# VirusMeter: Preventing Your Cellphone from Spies

Lei Liu[1] and Guanhua Yan[2] and Xinwen Zhang[3] and Songqing Chen[1]

[1] Department of Computer Science
George Mason University
[2] Information Sciences Group (CCS-3)
Los Alamos National Laboratory
[3] Computer Science Lab
Samsung Information Systems America

**Abstract.** Due to the rapid advancement of mobile communication technology, mobile devices nowadays can support a variety of data services that are not traditionally available. With the growing popularity of mobile devices in the last few years, attacks targeting them are also surging. Existing mobile malware detection techniques, which are often borrowed from solutions to Internet malware detection, do not perform as effectively due to the limited computing resources on mobile devices. In this paper, we propose VirusMeter, a novel and general malware detection method, to detect anomalous behaviors on mobile devices. The rationale underlying VirusMeter is the fact that mobile devices are usually battery powered and any malicious activity would inevitably consume some battery power. By monitoring power consumption on a mobile device, VirusMeter catches misbehaviors that lead to abnormal power consumption. For this purpose, VirusMeter relies on a concise user-centric power model that characterizes power consumption of common user behaviors. In a real-time mode, VirusMeter can perform fast malware detection with trivial runtime overhead. When the battery is charging (referred to as a battery-charging mode), VirusMeter applies more sophisticated machine learning techniques to further improve the detection accuracy. To demonstrate its feasibility and effectiveness, we have implemented a VirusMeter prototype on Nokia 5500 Sport and used it to evaluate some real cellphone malware, including FlexiSPY and Cabir. Our experimental results show that VirusMeter can effectively detect these malware activities with less than 1.5% additional power consumption in real time.

**Keywords:** mobile malware, mobile device security, anomaly detection, power consumption

## 1 Introduction

With the ever-improving chip design technology, the computing power of microprocessors is continuously increasing, which enables more and more features on mobile devices that were not available in the past. For example, today many cellphones come with various data services, such as text messaging, emailing, Web surfing, in addition to the traditional voice services. Due to their all-in-one convenience, these increasingly powerful mobile devices are gaining a lot of popularity: It has been expected a mobile population of 5 billion by 2015 [1] and out

of 1 billion camera phones to be shipped in 2008, smartphones represent about 10% of the market or about 100 million units [2]. Moreover, the new generation of mobile devices provide a more open environment than their ancestors. They now can not only run sandbox applications shipped from original manufacturers, but also install and execute third-party applications that conform to the norms of their underlying operating systems.

The new features brought by exotic applications, although rendering mobile devices more attractive to their users, also open the door for malicious attacks. By the end of 2007, there were over 370 different mobile malware in the wild [21]. The debut of Cabir [3] in 2004, which spreads through Bluetooth connections, is commonly accepted as the inception of modern cellphone virus [4]. Since then, a number of malware instances have been found exploiting vulnerabilities of mobile devices, such as Cabir [9] and Commwarrior [8]. These mobile malware have created serious security concerns to not only the mobile users, but also the network operators, such as information stealing, overcharging, battery exhaustion, and network congestion.

Despite the immense security threats posed by mobile malware, their detection and defense is still lagging behind. Many signature- and anomaly-based schemes for IP networks have been extended for mobile network malware detection and prevention [12,30,31]. For example, Hu and Venugopal proposed to extract signatures from mobile malware samples and then scan network traffic for these signatures [20]. Similar to their counterparts on IP networks, however, signature-based approaches can easily be circumvented by various techniques, such as encryption, obfuscation, and packing. On the other hand, anomaly-based detection schemes often demand accurate and complete models for normal states and are thus prone to high false alarm rates.

Recently, behavioral signatures have also been proposed for mobile malware detection [10]. They have their own limitations. On one hand, monitoring API calls within an emulated environment and running sophisticated machine learning algorithms for detection are hardly practical for resource-constrained mobile devices due to high detection overhead, not mentioning that most manufacturers do not publicize all relevant APIs on commodity mobile devices. On the other hand, stealthy malware can mimic user behavior or hide its activities among normal user activities to evade detection by API tracking. For example, FlexiSPY[5]-like malware that perform eavesdropping does not show anomalies in the order of relevant API calls, since they are typically implemented as if the user has received an incoming call. To detect energy-greedy malware and variants of existing malware, Kim et al. proposed to use power signatures based on system hardware states tainted by known malware [22]. Their approach, however, is mainly useful for detecting known malware and their variants.

In this study, we propose VirusMeter, a novel and general mobile malware detection method, to detect malware on mobile devices without demanding external support. The design of VirusMeter is based on the fact that *mobile devices are commonly battery powered and any malware activity on a mobile device will inevitably consume battery power*. VirusMeter monitors and audits power consumption on mobile devices with a behavior-power model that accurately characterizes power consumption of normal user behaviors. Towards this goal, Virus-

Meter needs to overcome several challenges. *First*, VirusMeter requires a power model that can accurately characterize power consumption of user behaviors on mobile devices, but such a model is not readily available as yet. *Second*, VirusMeter needs to measure battery power in real time. Existing research however shows that precise battery power measurement is difficult due to many electrochemical properties. In addition, although in practice mobile devices commonly have battery power indicators, their precision varies significantly from device to device. Examining the battery capacity frequently also incurs high computational overhead, rendering it hardly practical in reality. *Third*, as VirusMeter aims to run on on-the-shelf mobile devices without external support, it must be lightweight itself, without consuming too much CPU (and thus battery power); otherwise, it can adversely affect the detection accuracy.

To overcome these challenges, we design a user-centric power model that, as opposed to a system-centric model which requires in-depth understanding of various system-level behaviors and states, has only a small number of states based on common user operations. VirusMeter is designed to run in two modes: It, when in a *real-time* detection mode, performs fast malware detection, but when in a *battery-charging* mode, applies advanced machine learning techniques to detect stealthy malware with high accuracy.

To demonstrate its feasibility and effectiveness, we implement a VirusMeter prototype on Nokia 5500 Sport and evaluate its performance with real-world smartphone viruses including Cabir and FlexiSPY. The results show that VirusMeter can effectively detect the malware by consuming less than 1.5% additional power in a real-time mode. In a battery-charging mode, VirusMeter, by using advanced machine learning techniques, can considerably improve the detection rate up to 98.6%.

The remainder of the paper is organized as follows. Some related work is presented in section 2. We overview the VirusMeter design in section 3. We present our model designs, data collection, and model checking for VirusMeter in sections 4, 5, 6, respectively. We present our implementation in section 7 and evaluation results in section 8. We discuss some limitations and future work in section 9 and make concluding remarks in section 10.

## 2 Related Work

The increasing popularity of mobile devices with faster microchips and larger memory space has made them a lucrative playground for malware spreading. Existing studies [21] show that there are more than 370 mobile malware in the wild. As Symbian occupies the largest cellphone OS market share, it has been targeted by most of these mobile malware. Different approaches have been employed to classify existing mobile malware. For instance, mobile viruses have been classified based on their infection vectors, such as Bluetooth, MMS, memory cards, and user downloading [13,21]. Currently, user downloading, Bluetooth, and MMS are the most popular channels for mobile malware propagation. Several studies on mobile malware have focused on understanding their propagation behaviors. For example, an agent-based model has been developed to study worms spreading over short-range radio and cellular messaging systems [11]. A probabilistic

queuing model is proposed for the spreading of mobile worms over wireless connections [23], and a detailed mathematical model is also developed in [32] to characterize specifically the propagation process of Bluetooth worms. To detect Bluetooth worm outbreaks, Su et al. proposed to deploy monitors in high-traffic areas [29]. To simulate worm propagation in mobile phone networks, Fleizach et al. [17] developed a simulator with great details, including realistic topologies, provisioned capacities of cellular networks, and realistic contact graphs.

Some early studies on defense schemes against mobile malware have mainly focused on understanding their attack characteristics. For example, various potential attacks from a compromised cellphone and the corresponding defenses have been studied [15,16,19,26]. An algorithm based on user interactions is proposed to identify vulnerable users [12]. In [30], several schemes have been studied to mitigate DoS attacks via queuing in the network. To prevent cross service boundary attacks, a labeling technique is used to separate the phone interface from the PDA interface of a mobile device [24]. Sarat et al. [27] proposed to integrate commonwalk lengths and node frequencies to detect worms and determine their propagation origins. Recently, SmartSiren [13] showed how to use a proxy to detect malware by analyzing collected user communication logs. Bose et al. [10] proposed to extract behavioral signatures for mobile malware detection.

So far, existing schemes either have limited effectiveness by targeting particular situations (such as attacks through SMS), and/or demand significant infrastructure support, and/or demand non-trivial computing resources from mobile devices. By contrast, VirusMeter is a general approach, regardless of how malware invade into a system or whether they are known in advance. VirusMeter is also lightweight and can run on a mobile device without any external support. A previous approach that also aims to detect energy-greedy anomalies [22] is closest to VirusMeter. However, it is only effective to detect known malware and their variants, and works only for a single process mode which stealthy malware can easily evade by activating itself on when a user process is active.

## 3 Overview of VirusMeter Design

The rationale behind VirusMeter is the fact that any malware activities on a mobile device must consume some battery power. Hence, abnormal battery power consumption is a good indicator that some misbehavior has been conducted. Accordingly, VirusMeter monitors battery power usage on a mobile device and compares it against a pre-defined power consumption model to identify abnormal activities due to mobile malware.

Figure 1 shows the work flow of VirusMeter when executed on a mobile device. VirusMeter can run at either the system level or the application level (our current implementation is at the application level). Running at the system level is more robust against attacks since mobile OSes, such as Symbian and Windows Mobile, are often only accessible to device manufacturers or authorized parties. As shown in the figure, VirusMeter, using APIs provided by the underlying mobile OS, collects necessary information of supported services as well as the current remaining battery capacity. VirusMeter, based on the pre-defined power

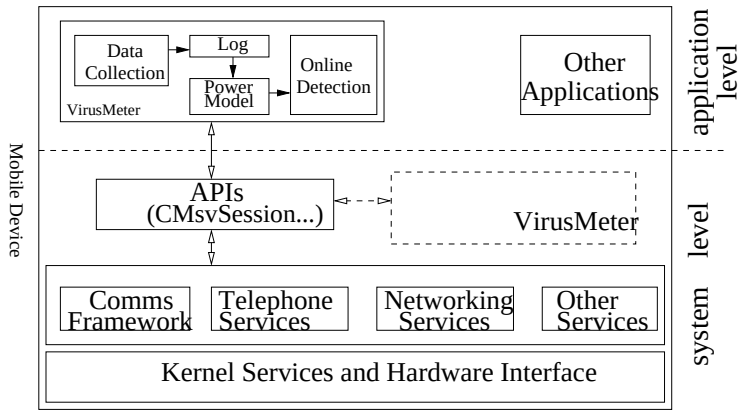


**Fig. 1.** VirusMeter Runs on a Mobile Device

model, calculates how much power could have been consumed due to these services and then compares it against the actually measured power consumption. The comparison result portends whether abnormal power draining has occured: If the difference exceeds a pre-specified threshold, VirusMeter raises an alarm indicating the existence of potential malware. Such comparison can be done in real time (a real-time mode) for fast malware detection, or when the battery is charging (a battery-charging mode) for high detection accuracy.

The alarms raised by VirusMeter are instrumental to further revealing malicious activities of the mobile malware. For instance, the user can check the communication records of her mobile device provided by the network operator to see whether there are any suspicious phone calls or text messages; she can also run more advanced virus removal tools to clean the mobile device. Hence, VirusMeter is a valuable tool to expose malware on mobile devices to their users at their early stages, thus preventing them from continuously compromising the service security or data confidentiality of the mobile device.

The pre-defined power model in VirusMeter is user-centric, which is relative to system-centric models that typically have too many system-level states. This model is constructed by VirusMeter itself when the device is in a clean state. VirusMeter consists of three major components: *User-Centric Power Model*, *Data Collector*, and *Malware Detector*. Figure 2 shows these components and the work flow of VirusMeter.

While the logic of VirusMeter is straightforward, we still need to overcome several challenges before VirusMeter becomes practically deployable on commodity mobile devices: **(1) Accurate power modeling:** An accurate yet simple power consumption model is crucial to the effectiveness of VirusMeter for malware detection. **(2) Precise power measurement:** Both model construction and data collection rely on precise power measurement. **(3) Low execution overhead:** For VirusMeter to be practically deployable, its own power consumption should not adversely affect the power-based anomaly detection.

In the following sections, we shall present more design and implementation details that address these challenges.

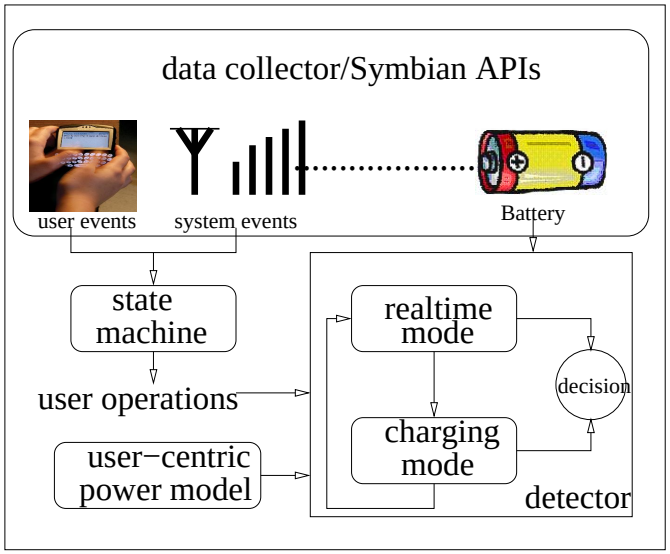


**Fig. 2.** VirusMeter Architecture

## 4 Building A User-Centric Power Model for VirusMeter

### 4.1 Existing Battery Power Models

Generally speaking, a battery's power consumption rate can be affected by two groups of factors, *environmental factors* such as signal strength, environmental noises, temperature, humidity, the distance to the base station, the discharging rate, the remaining battery power, etc., and *user operations* such as phone calls, emailing, text messaging, music playing, etc. Three types of power models have been suggested so far:

**(1) Linear Model**: In this simple model the remaining capacity after operating duration $t_d$ is given by

$$P_r = P_p - \int_{t=t_0}^{t_0+t_d} d(t)dt = P_p - I \times t_d, \tag{1}$$

where $P_p$ is the previous battery power, and $d(t)$ is the draining rate at time $t$. With the assumption that the operating mode does not change for $t_d$ time units, $d(t)$ stays the same during this period and is denoted as $I$. Once the operation mode changes, the remaining capacity is re-calculated [28].

**(2) Discharge Rate Dependent Model**: In this model, the discharge rate is considered to be related to the battery capacity. For this purpose, $c$ is defined as the fraction of the effective battery capacity $P_{eff}$ and the maximum capacity $P_{max}$, i.e., $c = \frac{P_{eff}}{P_{max}}$. Then the battery power is calculated as

$$P_r = c \times P_p - \int_{t=t_0}^{t_0+t_d} d(t)dt = c \times P_p - I \times t_d. \tag{2}$$

$c$ changes with the current; it becomes close to 1 when the discharge rate is low, and approaches 0 when the discharge rate is high [6,28].

**(3) Relaxation Model**: This model is based on a common phenomenon called relaxation [14,18], which refers to the fact that when a battery is discharged at a high rate, the diffusion rate of the active ingredients through the electrolyte and electrode will fall behind, and the battery reaches its end of life even if there are active materials available. If the discharge current is cut off or reduced, the diffusion and transport rate of active materials will catch up with the depletion of the materials [25]. Although this is the most comprehensive model characterizing a real battery, the model involves more than 50 electro-chemical and physical input parameters [25].

All these models calculate the battery power consumption from a physical and electrical perspective, although their inputs are remarkably different. The relaxation model can provide more accurate battery estimation than the linear model. However, even with aid of external instruments, measuring over 50 parameters could be difficult and expensive in practice. In addition, since VirusMeter aims to run on commodity mobile devices, it purely relies on publicly available system functions (without external support) to collect data; most of the 50 parameters in the relaxation model, however, cannot be captured with available APIs. Furthermore, a model with as many as 50 parameters is too cumbersome and thus not suitable for resource-constrained devices. The other two models model have similar problems, as the power draining rate and discharge rate are hard to measure without external power measurement instruments.

### 4.2 User-Centric Power Model

Due to the difficulties of measuring the input parameters of existing power models, we decide to build a user-centric power model for VirusMeter. In this model, the amount of power consumed is characterized as a function of common user operations and relevant environmental factors. Moreover, this model has only a few states, which is in contrast to those system-centric power models that need cumbersomely profile all system behaviors and are thus difficult to build without in-depth understanding of the mobile OS and its underlying hardware.

To derive a user-centric model from scratch, we investigate the power consumption of common types of user operations on mobile devices in different environments. The following types of user operations are now considered: *(1) Calling*: its power consumption is mainly dependent on the conversation duration. VirusMeter treats incoming and outgoing calls separately. *(2) Messaging*: its average power consumption depends on both the sizes and the types of the messages. MMS and SMS are the two message types being considered. Also, sending and receiving messages are treated as different activities. *(3) Emailing*: its power consumption is mainly decided by the amount of traffic, which we can get by querying the email message size. *(4) Document processing*: we assume that the duration of the operation is the deciding factor. *(5) Web surfing*: Web surfing is more complicated than the above as a user may view, download, or be idle when surfing the Web. Currently we calculate the average power consumption simply based on the amount of traffic involved and also the surfing duration. *(6) Idle*: for a large amount of time, a user may not operate on the device for anything. During this period, however, system activities such as signaling may

still take place. Under such a state, the power consumption is intuitively relevant to its duration. *(7) Entertainment and others*: currently, we simply assume the average power consumption is determined by the duration of the activities. This, admittedly, is a coarse model and further study is required.

For environmental factors, the following two types are being considered: *(1) Signal strength*: signal strength impacts the power consumption of all the above operations. The weaker of the signal strength, the more power consumption is expected. *(2) Network condition*: for some of the operations, network conditions are also important. For example, the time, and thus the power, needed to send a text message depends on the current network condition.

In VirusMeter, the battery power consumed between two measurements can be described as a function of all these factors during this period:

$$\Delta P = f(D^i_{call}, SS^i_{call}, T^j_{msg}, S^j_{msg}, SS^j_{msg}, N^j_{msg}..., D^k_{idle}, SS^k_{idle}), \qquad (3)$$

where $\Delta P$ represents the power consumption, $D$ the duration of the operation, $SS$ the signal strength, $T$ the type of the text message, and $N$ the network condition. $i$,$j$, and $k$ represent the index of the user operation under discussion.

To this end, what is missing in this user-centric power model is the function itself in Equation 3. This is derived from the following three different approaches:

**Linear Regression:** Linear regression generates a mathematical function which linearly combines all variables we have discussed with techniques such as least square functions; it can thus be easily stored and implemented in a small segment of codes that run on commodity mobile devices with trivial overhead. While linear regression may incur little overhead, which makes it suitable for real-time detection, its accuracy depends on the underlying assumption of the linear relationship between variables.

**Neural Network:** An artificial neural network (ANN), often referred to as a "neural network" (NN), is a mathematical or computational model inspired by biological neural networks. It consists of an interconnected group of artificial neurons that process information using a connectionist approach for computation. Neural networks are commonly used for non-linear statistical data modeling. They can be used to model complex relationships between inputs and outputs or to find patterns in data. In VirusMeter, we use neural network as a regression tool, in which the neural network model, unlike the linear regression model, cannot easily be presented as a mathematical function.

**Decision Trees:** A decision tree is a predictive model that maps the observations of an item to conclusions of its target value. In a decision tree, branches represent conjunctions of features that lead to leaves that represent classifications. In VirusMeter we build a classification tree in which branches represent normal or malware samples. We train the decision tree with both normal and malware data samples. When a new piece of data sample is fed into the decision tree, it can tell if the new data is normal or not, as well as which malware most likely caused the abnormal power consumption.

## 5 Constructing State Machines for Data Collection

To train the three power models presented in the previous section, VirusMeter needs to collect some data. For the linear and neural network model construction, only clean data are needed. For decision tree construction, both clean data and dirty data (the data when malware programs are present) are needed. In this section, we present how VirusMeter collects these data to train the models.

Currently, we mainly consider the user operations defined in the previous section and their corresponding power consumption in VirusMeter. Although the power consumption can be queried using public APIs, there is no interface that could be directly called for the user operations. As it is common for commodity devices to provide some APIs for third parties to query, register, and monitor system-level events or status, we construct a state machine to derive user operations (which we also call *external* events) from system events (which we also call *internal* events). In this state machine, state transitions are triggered by internal events when they appear in a certain order and satisfy certain timing constraints. For example, during a normal incoming call, a ring event must precede another answer key event, but cannot happen more than 25 seconds before the answer key event, because ringing lasts for less than 25 seconds in our experimental cellphone before the call is forwarded to the voicemail service.

One may wonder whether we can simply use these state machines to detect malware without power auditing. This is possible but can potentially miss some malware for two reasons. On one hand, stealthy malware can easily evade detection by mimicing normal user behaviors that can be derived from the state machine. On the other hand, it is difficult, if not impossible, to build a state machine that exhaustively characterizes all possible user operations. The state machine in VirusMeter covers only the internal events corresponding to those common user operations that we have defined. Due to these concerns, we still need the power model for mobile malware detection.

**Algorithm 1** State Machine Construction for Each User Operation

1: Run a monitor program on the clean cellphone.
2: Execute a defined user operation, such as a phone call.
3: Monitor and record all related internal events during the test period and their properties.
4: Find the correlation between a user operation and the internal events, their dependency and sequences.
5: Query and record all parameters of the events.
6: Repeat the experiment.
7: Abstract the common event sequence from the recording. These internal events are used to build the state machine.

VirusMeter performs Algorithm 1 to construct the state machine for each user operation defined previously. Figure 3 shows an example of the obtained state machine for receiving a phone call. In this figure, the triggering events are marked on the transition arrows. Starting in the `Idle` state, the state machine transits

to the `Ring` state after a `ring` event. If the user decides to answer the call by pressing the answer key, the answer key event is generated, which makes the state machine move to the `Answer` state if the answer key event happens half a second to 25 seconds after the `Ring` state. On a Symbian cell phone, we can observe an *EStatusAnswering* event. At this time, the state machine starts a timer. When the user terminates the call by pressing the cancel key or hanging it up, the state machine turns to the `End` state followed by a Symbian *EStatusDisconnecting* event. The state machine now stops the timer and calculates the calling duration. Finally the state machine returns to `Idle` state and generates a `receiving call` operation with the call duration. In a similar approach, we conduct experiments to build state machines for other user operations we have defined.

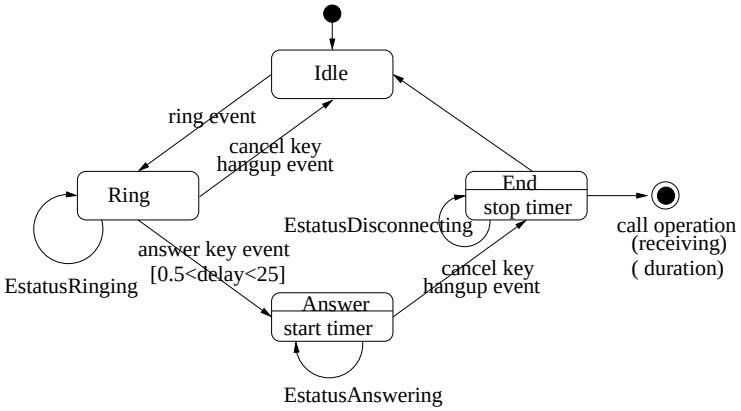


**Fig. 3.** State Machine for Receiving a Phone Call

## 6 Model Checking for Malware Detection

With the power model and the state machines available, VirusMeter can perform malware detection in a straightforward manner: we use the power model to predict how much power should be consumed and then compare it against the measured power consumption. If abnormal power consumption is observed, an alert is raised. Here, VirusMeter is designed with two running modes:

- *Real-time mode*: VirusMeter uses the linear regression power model to predict power consumption due to its low computational cost.
- *Battery-charging mode*: Although linear regression is easy to perform, it may generate false detection results since (1) it implicitly assumes a linear relationship among all variables, and (2) power measurements may have fluctuations due to electro-chemical battery properties. Thus, VirusMeter accumulates power consumption measurement data and uses the neural network model and the decision tree algorithm to perform malware detection when the battery is charging.

It is noted that both modes can also run off the mobile device. For example, the device manufacturer or the service operator may provide such a service

that a user can submit the collected measurement data to a server for malware detection. This however will increase the communication cost on the mobile device.

## 7 Practical Issues in VirusMeter Implementation

As Symbian is the most popular mobile OS, we implement a prototype of VirusMeter on Nokia 5500 Sport, supported by Symbian OS 9.1. Figure 4 shows the modules of VirusMeter implementation. Currently, it is implemented as a user-level application and a user can choose to start it or to shut it down manually. The implementation program uses a client/server architecture which is widely used for Symbian applications. Figure 5 shows the user interface once VirusMeter is installed on our experimental device.

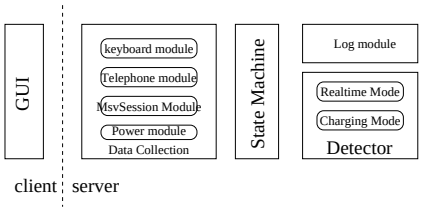


**Fig. 4.** Modularized VirusMeter Implementation


**Fig. 5.** VirusMeter Implemented on a Nokia 5500 Sport

### 7.1 Power Measurement Precision and Power Model Construction

The power consumption data are collected through the APIs provided by Symbian for power status changes. In the prototype implementation, however, we find that the precision of the power capacity measurement is not sufficient. In fact, the precision returned by the APIs of mobile devices varies significantly. For example, iPhone can return the current power capacity at 1% precision. Many other devices, including the one in our experiments, return the power consumption data only at the level of battery bars shown on the screen. On the Nokia 5500, these bars are at the 100, 85, 71, 57, 42, 28, 14, and 0 percent of the full capacity. We call the battery supply between two of these successive values as a power *segment*. To overcome the precision challenge, we perform experiments long enough so that the power consumption is sufficient to cross a segment. Assuming a constant draining rate during the experiments, we expect the power measurement through this method is more accurate.

Accordingly, it is necessary to transform the power model in Equation 3 because if we were to still follow Equation 3, many experiment samples would have the same constant dependent value $\Delta P$, which is bad for the linear regression and neural network regression. To make the regression as accurately as possible, we transform the function as follows. Because in all our experiments, the signal strength is always good (at level 6 and 7) but the duration of idle time has a

large range, we select idle time at the best signal strength as the dependent variable, and transform our model to

$$D_{idle} = f'(D^i_{call}, SS^i_{call}, T^j_{msg}, S^j_{msg}, SS^j_{msg}, N^j_{msg}..., \Delta P, SS^k_{idle}). \quad (4)$$

For environmental factors, VirusMeter is currently only concerned about the signal strength and network condition. Through the API, VirusMeter can directly query the current signal strength. There are 7 levels of signal strength on Nokia 5500, from 1 to 7. We, however, cannot directly query APIs for network conditions when a user performs a certain operation, such as text messaging. In the experiments, we have observed that if the network congestion is severe, the duration for sending or receiving messages increase significantly. Therefore, to make the power model more accurate, we introduce the sending time into it, and the duration is measured as follows. In Symbian, sending a message leads to a sequence of events that can be captured by VirusMeter: first, an `index` is created in the `draft` directory; when the creation is complete, the `index` is moved to the `sending` directory; when sending is successful, the `index` will be moved to the `sent` directory. Hence, the operation time can be measured from the time when the `index` is created to the time when it is moved to the `sent` directory. Following the similar idea, we further refine the parameter input for receiving messages and other networking operations.

*Note that our power model is built in such a way due to insufficient power precision, but a malware does not need to be active throughout a segment of battery power to be detected by VirusMeter. Instead, no matter how long the malware is active, we can always feed the runtime data collected during an entire power segment for malware detection, and our experiments in the next section will show that it is still very effective.*

### 7.2 Data Collection Rules

To construct the power model, we need to collect not only the power consumption data under normal user operations (clean data) for the three power models, but also dirty data when malware is present for training the decision tree. Constrained by the precision of the battery power measurement offered by Symbian OS, we treat all user operations conducted in one battery segment as a batch to achieve more accurate detection. As our goal is to detect malware whose activities lead to abnormal power consumption no matter how long they are active, we collect clean data under various circumstances for model construction: (1) In some experiments, our data collection just focuses on a single user operation. For example, in a battery segment, we only send SMS text messages, and in another one, we only receive SMS text messages; (2) In some experiments, mixed user operations are conducted. For example, in a battery segment, we make phone calls and also receive text messages; (3) For each user operation, we consider various properties of the activity. For instance, we send text messages with different sizes ranging from ten bytes to a thousand bytes; and (4) In all experiments, we avoid abnormal conditions, which decrease the accuracy of our power models.

Dirty data are also necessary to train the decision trees. The power consumption of a malware program may vary significantly in different environments. For

example, different usage frequencies or spy call durations on FlexiSPY cause great difference in power consumption. In another example, the power consumed by the Cabir worm depends on how many Bluetooth devices exist in the neighborhood. Based on such considerations, we collect dirty data as follows: (1) During dirty data collection, we conduct experiments to cover as many different scenarios as possible, including both high power consumption cases and low power consumption cases; and (2) For the purpose of model training, the fraction of high and low power consumption data samples are randomly selected.

### 7.3 Stepwise Regression for Data Pre-Processing and Time-Series Data Analysis

The data we have collected, including both clean and dirty data, have 41 variables that are measurable through the Symbian APIs. To simplify the model by eliminating insignificant factors, we first use the `stepwise regression` technique [7] to pre-process the collected data. Stepwise regression is a statistical tool that helps find the most significant terms and remove least significant ones. Besides that, stepwise regression also provides information that help to merge variables. Using stepwise regression, we found that the idle time with signal strength level 6 is insignificant. This is because in our experimental environment, we often have good signal strength at level 7. The signal strength 6 is relatively rare. Thus, we merge the signal strength 6 to the signal strength 7.

To further improve the model accuracy, we collect data samples from multiple segments and use the average to smooth out the fluctuations due to the internal electro-chemical battery properties. Based on this idea, we generate three sets of input for each power model. If a model is built from data samples collected in a single battery power segment, we call them "short-term" experiments. If a model is built from data samples from seven segments, we call them "middle-term" experiments. Note that Nokia 5500 only has seven battery segments. We can further feed data samples collected in more than one battery lifecycle. In our experiments, we use four battery lifecycles, which correspond to 28 segments, and we call them "long-term" experiments. A stealthy malware that does not consume much power in one segment may not be caught in a short-term detection, but can be caught in the middle- or long-term detection.

## 8 Evaluation Results

In this section, we use actual mobile malware, including FlexiSPY, Cabir, and some variants of Cabir, to evaluate the effectiveness of VirusMeter. `FlexiSPY` is a spyware program that runs on either Symbian OS or Blackberry handhelds. Once installed, it conducts eavesdropping, call interception, GPS tracking, etc. It monitors phone calls and SMS text messages and can be configured to send them to a remote server. We test three major types of misbehaviors supported by FlexiSPY: *eavesdropping (spy call)*, *call interception*, and *message (text message and email) forwarding*. Figure 6 shows the information flow of FlexiSPY. The `Cabir` malware exploit Bluetooth to spread themselves. We obtained 3 Cabir

variants and in the experiments, we used two of them for decision tree training and the other one for testing.

We have several sets of experiments to examine common malware behaviors that consume low (such as Cabir), medium (such as text-message forwarding), and high battery power (such as eavesdropping). We also evaluate false positives and the runtime overhead, i.e., power consumption, of VirusMeter.

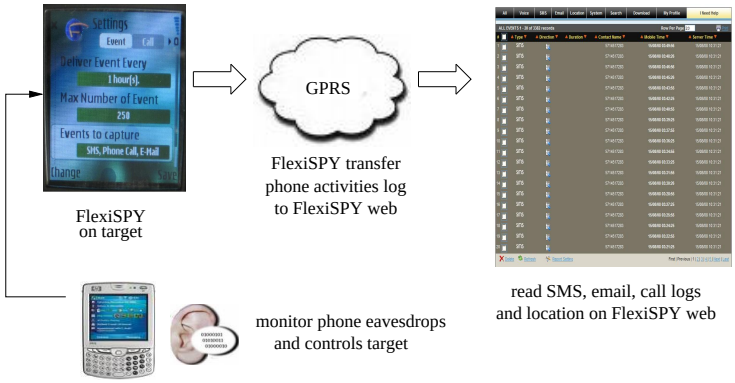


**Fig. 6.** FlexiSPY Running on Nokia 5500 Sport and the Information Flow

### 8.1 Experiments on Eavesdropping Detection

When using FlexiSPY to eavesdrop on a cellphone, the attacker makes a call to a previously configured phone number and then the phone is activated silently without user authentication. Our power measurement shows that eavesdropping has a similar power consumption rate as a normal call. In our experiments, we make spy calls of different time durations uniformly ranging from 1 minute to 30 minutes. More than 50 samples are collected in this and each of the following detection rate experiments. Table 1 shows the detection rates (true positives).

**Table 1.** Detection Rate (%) on Eavesdropping

| | Short-Term | Middle-Term | Long-Term |
|---|---|---|---|
| Linear Regression | 85.1 | 89.9 | 87.1 |
| Neural Network | 89.3 | 90.9 | 93.0 |
| Decision Tree | 89.8 | 90.2 | 88.9 |

The results show that for eavesdropping, both middle-term and long-term experiments can improve the detection rates for linear regression and neural network, compared with short-term detection. In fact, even the short-term linear regression achieves a detection rate over 85%. This is because eavesdropping consumes a lot of power, which makes short-term detection quite accurate. Surprisingly, the long-term detection based on linear regression generates a worse

result than mid-term detection. Our conjecture is that due to the inaccurate linear relationship between variables, more errors may be accumulated in the long-term experiments, which leads to worse results. This may apply to long-term decision tree as well.

### 8.2 Experiments on Call Interception Detection

FlexiSPY can also perform call interceptions, which enables the attacker to monitor ongoing calls. A call interception differs from eavesdropping in that the call interception can only be conducted when a call is active. After FlexiSPY is installed, when the victim makes a call to a pre-set phone number, the attacker will automatically receive a notification via text message and silently call the victim to begin the interception.

**Table 2.** Detection Rate (%) on Call Interception

| | Short-Term | Middle-Term | Long-Term |
|---|---|---|---|
| Linear Regression | 66.8 | 79.5 | 82.4 |
| Neural Network | 82.9 | 86.0 | 90.5 |
| Decision Tree | 84.8 | 86.8 | 86.9 |

In our detection experiments, we again perform call interceptions with different time durations uniformly ranging from 1 minute to 30 minutes. Table 2 shows the detection rate. The short-term linear regression detection results are not very good when compared to neural network and decision tree. This is because the call interception only consumes slightly more battery power than a normal phone call and it only works when a call is active. But middle-term and long-term experiments can significantly improve the detection rate for linear regression. The results confirm that for stealthy malware that consumes only a small amount of power, a more accurate model or a longer detection time can help improve the detection accuracy.

### 8.3 Experiments on Text-message Forwarding and Information Leaking Detection

FlexiSPY can also collect user events, such as call logs, and then deliver collected information via a GPRS connection periodically at a pre-configured time interval. Clearly, transferring data through GPRS consumes power and the power consumption depends on the time interval and the characteristics of user operations such as the number of text messages sent during each interval.

**Table 3.** Detection Rate (%) on Text Message Forwarding

| | Short-Term | Middle-Term | Long-Term |
|---|---|---|---|
| Linear Regression | 89.5 | 93.0 | 96.4 |
| Neural Network | 90.3 | 94.8 | 98.6 |
| Decision Tree | 88.7 | 89.1 | 90.7 |

In our detection experiments, we set the interval from 30 minutes to 6 hours, with an interval of 30 minutes. Under each setting, we keep sending and receiving text messages of different sizes ranging from 10 bytes to 1000 bytes.Table 3 shows the detection results. We can see all three approaches achieve detection rates above 88%. The long-term detection with linear regression and neural network can achieve a detection rate up to 98.6%. Our analysis shows that this is because such a FlexiSPY functionality consumes additional power other than communication: although when the interval is large, FlexiSPY may not transfer data for a while, FlexiSPY still needs to monitor and save information related to user activities, which also consumes battery power. Thus, even in short-term experiments, the detection rate is quite high. To our surprise, decision tree does not achieve comparable results to linear regression and neural networks for middle-term and long-term detection. We believe that the performance of decision tree is highly related to the training dataset, for which we are currently constrained by a limited number of malware samples.

### 8.4 Experiments on Detecting Cabir

Cabir, a cellphone worm spreading via Bluetooth, searches nearby Bluetooth equipments and then transfers a `sis` file to them once found. The power consumption of Cabir mainly comes from two parts: neighbor discovery and file transferring. Because Bluetooth normally does not consume significant battery power, we conduct the experiments in an environment full of Bluetooth equipments, in which Cabir keeps finding new equipments and thus consumes a non-trivial amount of power. To control the frequency of file transferring, we repeatedly turn off Bluetooth on these devices for a random amount of time after a transfer completes and then turns it on again.

**Table 4.** Detection Rate (%) on Cabir

| | Short-Term | Middle-Term | Long-Term |
|---|---|---|---|
| Linear Regression | 84.6 | 89.8 | 92.9 |
| Neural Network | 88.6 | 93.4 | 93.5 |
| Decision Tree | 86.8 | 87.6 | 88.7 |

Table 4 shows the detection results. For linear regression, the middle-term and long-term detection can remarkably improve the detection result. The table also indicates that although Bluetooth discovery and file transferring only consume a limited amount of battery power, it can be detected with a reasonably high rate by VirusMeter at real time.

### 8.5 Experiments on Detecting Multiple Malware Infections

Previous detection experiments all involve only one malware program running on the cellphone. It is possible that a mobile device is infected by more than one malware program and each malware program could perform different attacks simultaneously. To test such cases, we activate both FlexiSPY and Cabir on our experimental cellphone and randomly conduct various attack combinations.

**Table 5.** Detection Rate (%) on Multiple Malware Infection

| | Short-Term | Middle-Term | Long-Term |
|---|---|---|---|
| Linear Regression | 84.8 | 87.9 | 88.1 |
| Neural Network | 88.9 | 90.2 | 92.0 |
| Decision Tree | 72.6 | 76.3 | 73.6 |

Table 5 shows the detection rates. The results show that both linear regression and neural network still have reasonably high true positive rates. But decision tree results in a much higher false negative rate than in single malware infection experiments. Although it is seemingly counterintuitive, the underlying principle of these three approaches can explain this: linear regression and neural network regression only predict the power consumption of normal user operations rather than describing power consumption of specific malware activities, which is the objective of decision tree. However, our decision tree model is not trained with a mixture of malware samples. Thus for data samples collected when multiple malware programs are active, its performance is the worst.

### 8.6 False Positive Experiments

In addition to the detection rates, we also conduct experiments to evaluate false positives. By feeding power models with a clean dataset, we can get the prediction result and calculate the false positive rate. For this purpose, we collect more than 100 clean data samples for experiments.

**Table 6.** VirusMeter False Positive Rate (%)

| | Short-Term | Middle-Term | Long-Term |
|---|---|---|---|
| Linear Regression | 22.4 | 14.2 | 10.3 |
| Neural Network | 10.0 | 5.1 | 4.3 |
| Decision Tree | 15.2 | 15.1 | 14.4 |

Table 6 shows the false positive rates. The results show that linear regression in short-term detection has the highest false positive rate. This is due to the inaccuracy of the underlying assumption of linear regression model. However, both middle-term and long-term experiments can significantly reduce the false positive rates. With a more accurate power model, neural network achieves the best results among the three for all three terms.

### 8.7 VirusMeter Overhead Measurement

As VirusMeter targets to run on commodity devices, its power consumption overhead is a great concern. As we cannot directly measure the power consumption of VirusMeter, we conduct experiments as follows: with and without VirusMeter running on the cellphone, we conduct the same set of user operations and then compare the operating durations under these two scenarios. Figure 7 shows our experimental results. We can see that with and without VirusMeter, the duration of various user operations is very close. The average duration when VirusMeter

is off is 109.5 minutes across our experiments, while the average duration when VirusMeter is on is 108 minutes. This indicates the VirusMeter running overhead is less than 1.5%. Note the above results show the overhead when VirusMeter uses the linear regression model. For the other two approaches, we do not evaluate their power consumption because they run in a battery-charging mode.

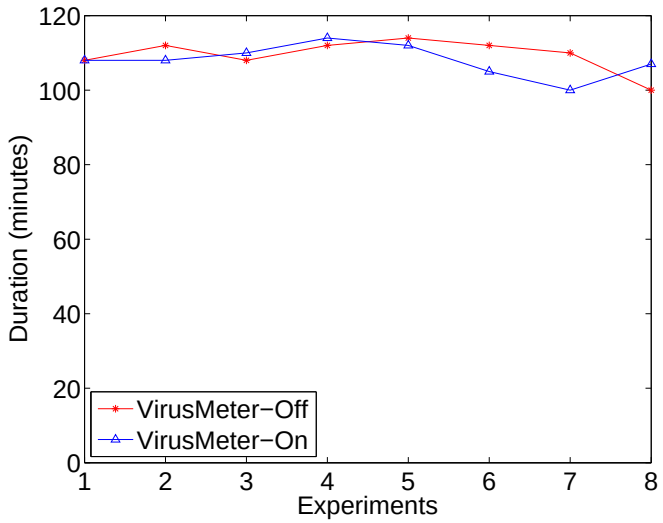


**Fig. 7.** VirusMeter overhead evaluation

## 9 Further Discussion

VirusMeter, in principle, should have potential to detect any misbehavior with abnormal power consumption as long as the battery power metering is sufficiently accurate. Currently, precisions of battery power indicators vary significantly among different mobile OSes, which can affect the detection efficiency of VirusMeter. This is particularly important for real-time detection. Practically, on our experimental device, this changes the real-time detection mode of VirusMeter to a near-real-time mode.

Since VirusMeter relies on the user-centric power models to detect malware, the accuracy of the models themselves is important. Our experimental results have shown that linear regression, although consuming trivial additional power, may generate high false negative rates due to the inaccurate underlying assumption between variables. On the other hand, in a battery-charging mode, neural network often improves the detection rate remarkably due to lack of such an assumption. The decision tree model does not perform as effectively as neural networks in our experiments. We believe that our limited malware samples may adversely affect its performance, and we plan to collect more samples in the future to further evaluate this method. In addition, for some types of user operations, such as entertainment and Web surfing, more fine-grained profiling can further improve the accuracy of the power model.

As we suggested, VirusMeter can also run in the battery-charging mode to improve the detection accuracy. Malware may leverage this as well since when the battery is charging, there is no way for VirusMeter to accurately measure the power consumption without any external assistance. To capture this kind of malware, VirusMeter needs external devices to measure how much power is charged and how much power is consumed. On the other hand, currently most mobile OSes are only accessible to manufacturers. If we place VirusMeter in the mobile OS, VirusMeter becomes more resilient to those attacks that could fail signature- or anomaly-based detection schemes. But if the mobile OS is also cracked and the malware knows how to inject faked events, VirusMeter will also fail because the data collected by VirusMeter cannot be trusted any more.

## 10 Conclusion

The battery power supply is often regarded as the Achilles' heel of mobile devices. Provided that any activity conducted on a mobile device, either normal or malicious, inevitably consumes some battery power, VirusMeter exploits this to detect existence of malware with abnormal power consumption. VirusMeter relies on a concise lightweight user-centric power model and aims to detect mobile malware in two modes: While the real-time detection mode provides immediate detection, running VirusMeter under the battery-charging mode can further improve the detection accuracy without concerns of resource consumption. Using real-world malware such as Cabir and FlexiSpy, we experimentally show that VirusMeter can effectively and efficiently detect their existence.

## References


1. http://www.wellingtonfund.com/blog/2007/02/19/gmp-3gsm-wrapup/.
2. `http://en.wikipedia.org/wiki/Smartphone`.
3. `http://www.viruslibrary.com/`.
4. `http://vx.netlux.org/29a/`.
5. `http://www.flexispy.com/`.
6. `http://www.panasonic.com/inustrial_oem/battery/battery_oem/chem/lith/lith.htm`.
7. `http://en.wikipedia.org/wiki/Stepwise_regression`.
8. Commwarrior. `http://www.f-secure.com/v-descs/commwarrior.shtml`.
9. Sprots fans in helsinki falling prey to cabir. `http://news.zdnet.com`.
10. A. Bose, X. Hu, K. Shin, and T. Park. Behavioral detection of malware on mobile handsets. In *Proceedings of Mobisys*, Breckenridge, CO, June 2008.
11. A. Bose and K. Shin. On mobile virus exploiting messaging and bluetooth services. In *Proceedings of Securecomm*, 2006.
12. A. Bose and K. Shin. Proactive security for mobile messaging networks. In *Proceedings of WiSe*, 2006.
13. J. Cheng, S. Wong, H. Yang, and S. Lu. Smartsiren: Virus detection and alert for smartphones. In *Proceedings of ACM MobiSys*, San Juan, Puerto Rico, 2007.
14. C. Chiasserini and R. Rao. Pulsed battery discharge in communication devices. In *Proceedings of MobiComm*, Seattle, WA, August 1999.
15. D. Dagon, T. Martin, and T. Starner. Mobile phones as computing devices: The viruses are coming! In *IEEE Pervasive Computing*, 2004.

16. W. Enck, P. Traynor, P. McDaniel, and T. Porta. Exploiting open functionality in sms-capable cellular networks. In *Proceedings of CCS'05*, November 2005.
17. C. Fleizach, M. Liljenstam, P. Johansson, G. Voelker, and A. Mehes. Can you infect me now? malware propagation in mobile phone networks. In *Proceedings of WORMS*, Alexandria, VA, November 2007.
18. T. Fuller, M. Doyle, and J. Newman. Simulation and optimization of the dual lithium ion insertion cell. In *Journal of Electrochem. Soc.*, volume 141, April 1994.
19. C. Guo, H. Wang, and W. Zhu. Smart-phone attacks and defenses. In *Proceedings of HotNets III*, San Diego, CA, November 2004.
20. G. Hu and D. Venugopal. A malware signature extraction and detection method applied to mobile networks. In *Proceedings of IPCCC*, April 2007.
21. M. Hypponen. http://www.usenix.org/events/sec07/tech/hypponen.pdf.
22. H. Kim, J. .Smith, and K. Shin. Detecting energy-greedy anomalies and mobile malware variants. In *Proceedings of Mobisys*, Breckenridge, CO, June 2008.
23. J. Mickens and B. Noble. Modeling epidemic spreading in mobile networks. In *Proceedings of ACM WiSe*, 2005.
24. C. Mulliner, G. Vigna, D. Dagon, and W. Lee. Using labeling to prevent cross-service attacks against smart phones. In *Proceedings of DIMVA*, 2006.
25. S. Park, A. Savvides, and M. Srivastava. Battery capacity measurement and analysis using lithium coin cell battery. In *Proceedings of ISLPED*, August 2001.
26. R. Racic, D. Ma, and H. Chen. Exploiting mms vulnerabilities to stealthily exhaust mobile phone's battery. In *Proceedings of SecureComm'06*, August 2006.
27. S. Sarat and A. Terzis. On the detection and origin identification of mobile worms. In *Proceedings of WORMS*, Alexandria, VA, November 2007.
28. T. Simunic, L. Benini, and G. Micheli. Energy-efficient design of battery-powered embedded systems. In *Proceedings of ISLPED*, August 1999.
29. J. Su, K. Chan, A. Miklas, K. Po, A. Akhavan, S. Saroiu, E. Lara, and A. Goel. A preliminary investigation of worm infections in a bluetooth environment. In *Proceedings of WORM*, 2006.
30. P. Traynor, W. Enck, P. McDaniel, and T. Porta. Mitigating attacks on open functionality in sms-capable cellular networks. In *Proceedings of Mobicom'06*, 2006.
31. D. Venugopal, G. Hu, and N. Roman. Intelligent virus detection on mobile devices. In *Proceedings of ACM PST*, Markham, Ontario, Canada, October 2006.
32. G. Yan and S. Eidenbenz. Modeling propagation dynamics of bluetooth worms. In *Proceedings of ICDCS'07*, 2007.