# SmartSiren: Virus Detection and Alert for Smartphones

Jerry Cheng[1], Starsky H.Y. Wong[1], Hao Yang[2] and Songwu Lu[1]
Dept. of Computer Science, UCLA, 4732 Boelter Hall, Los Angeles, CA 90025 [1]
IBM T.J. Watson Research, 19 Skyline Drive, Hawthorne, NY 10532 [2]
{chengje,hywong1,slu}@cs.ucla.edu[1], haoyang@us.ibm.com[2]

## ABSTRACT

Smartphones have recently become increasingly popular because they provide "all-in-one" convenience by integrating traditional mobile phones with handheld computing devices. However, the flexibility of running third-party softwares also leaves the smartphones open to malicious viruses. In fact, hundreds of smartphone viruses have emerged in the past two years, which can quickly spread through various means such as SMS/MMS, Bluetooth and traditional IP-based applications. Our own implementations of two proof-of-concept viruses on Windows Mobile have confirmed the vulnerability of this popular smartphone platform.

In this paper, we present *SmartSiren*, a collaborative virus detection and alert system for smartphones. In order to detect viruses, *SmartSiren* collects the communication activity information from the smartphones, and performs joint analysis to detect both single-device and system-wide abnormal behaviors. We use a proxy-based architecture to offload the processing burden from resource-constrained smartphones and simplify the collaboration among smartphones. When a potential virus is detected, the proxy quarantines the outbreak by sending targeted alerts to those immediately threatened smartphones. We have demonstrated the feasibility of *SmartSiren* through implementations on a Dopod 577w smartphone, and evaluated its effectiveness using simulations driven by 3-week SMS traces from a national cellular carrier. Our results show that *SmartSiren* can effectively prevent wide-area virus outbreaks with affordable overhead.

## Categories and Subject Descriptors

C.2.0 [**Information Systems Applications**]: General—*Security and Protection*

## General Terms

Design, Security

## Keywords

Security, Smartphone, Virus Detection, Alert, Privacy


*MobiSys'07,* June 11–14, 2007, San Juan, Puerto Rico, USA.


## 1. INTRODUCTION

Smartphones are a new type of communication device that combines the functionality of a traditional mobile phone (i.e. voice and messaging) with that of a handheld computing device such as PDA. Unlike traditional mobile phones, smartphones are data-centric and capable of running third-party software applications. The convenience of an "all-in-one" device makes the smartphone very attractive to a wide range of users. Since its introduction a few years ago, smartphones have taken up 5% of the mobile phone market share within the United States [17] and analysts have estimated a continuous strong growth rate as high as 156% per year [22].

Unfortunately, smartphone's increasing popularity and its capability to run third-party software have also attracted the attention of virus writers. The first proof-of-concept smartphone virus Cabir [21], which spreads through the Bluetooth interface, was introduced in 2004 by the virus writing group 29A [1]. Subsequently a large number of smartphone viruses have followed and attempted to exploit the unique vulnerabilities of smartphones. A few instances of smartphone virus outbreaks have been reported, e.g., one at the World Athletics Championships by Cabir [18] and another internally at a company by CommWarrior [12]. Despite their relatively small scales, these incidents foreshadow more severe threats to come as smartphones become more widely adopted.

Currently, the best defense against smartphone viruses mirrors the strategy against computer viruses with the inception of smartphone anti-virus software. While anti-virus is expected to be effective in addressing smartphone viruses, it also has several limitations. First, a smartphone typically has only limited processing power, storage capacity and battery power, which can hamper the effectiveness of an on-device anti-virus software [15]. Secondly, it is challenging to distribute virus signatures files to the smartphones in a timely manner, because a smartphone may not be always connected to the Internet or may induce additional costs for the Internet connection, e.g., through GPRS/EDGE. When combined, these issues result in a fertile ground for widespread infection of smartphone viruses, which can cripple the mobile phone users as well as the cellular and telephony infrastructure.

In this work, we propose *SmartSiren*, a collaborative virus detection and alert system for smartphones. *SmartSiren* targets a practical scenario where some smartphones are not equipped with anti-virus software or have not received the latest virus signatures. The goal of *SmartSiren* is to halt the potential virus outbreak by minimizing the number of smartphones that will be infected by a new released virus.

As such, *SmartSiren* is orthogonal to and complements the existing anti-virus solutions for smartphones.

In our system, each smartphone runs a light-weight agent, while a centralized proxy is used to assist the virus detection and alert processes. The benefit of such a proxy-based approach is to offload most of the processing burden from the resource-constrained smartphones, and to simplify the collaboration among the smartphones. Specifically, each smartphone agent keeps track of the communication activities on the device, and periodically reports a summary of these activities to the proxy. In cases where abnormal activities have been locally identified, a smartphone may also submit a report immediately to the proxy. On the other hand, the proxy performs joint analysis on the received reports and detects any single-device or system-wide viral behaviors. When a potential virus is detected, the proxy sends targeted alerts to both infected devices and a subset of the uninfected devices, which may be in direct contact with an infected device, based on the users' contact lists and mobility profiles.

One salient feature of *SmartSiren* is the protection of user privacy. In practice, most users are not willing to reveal the activities on their phones to the proxy. *SmartSiren* addresses such privacy concerns by an anonymous and ticketed report submission scheme. On the one hand, it prevents the proxy from knowing the activities of any user. On the other hand, it also prevents a virus or an attacker from abusing the privacy mechanism and injecting bogus reports in large amounts to mislead the virus detection results.

We have implemented a prototype of *SmartSiren* on a Dopod 577w smartphone [5] that runs the Windows Mobile operating system, an increasingly popular platform for smartphones. We also implemented two proof-of-concept viruses on Windows Mobile that demonstrated its vulnerabilities. The effectiveness of *SmartSiren* is evaluated using simulations that are driven by 3-week SMS traces obtained from a national cellular carrier in India. Our results have shown that *SmartSiren* can confine the outbreak of smartphone viruses with affordable overhead.

In summary, our contributions are three-fold. First, we demonstrate the vulnerability of Window Mobile smartphones through the implementation of proof-of-concept viruses. Second, we design and implement *SmartSiren* that can prevent virus outbreak via collaborative detection and targeted alerts. Lastly, we propose a ticketing scheme that can preserve the user privacy yet still enforce accountable reports.

The rest of the paper is organized as follows. Section 2 describes the existing smartphone viruses and reports our own virus implementation efforts. Section 3 discusses the challenges in combating smartphone viruses. Section 4 presents the design of *SmartSiren* framework, and section 5 describes the design enhancement of protecting user privacy. Section 6 summarizes our prototype implementation on the smartphone, and Section 7 evaluates the effectiveness of our design through trace-driven simulations. Section 8 compares to the related work, and Section 9 discusses several remaining design issues. Finally, Section 10 concludes the paper.

# 2. SMARTPHONE VIRUSES

In this section, we first describe and categorize the existing smartphone viruses, then provide insights into the vulnerability of Windows Mobile by implementing two proof-of-concept viruses on this popular smartphone platform.

## 2.1 Existing Smartphone Viruses

The first smartphone virus, Cabir, was released in 2004 by the virus writing group 29A [1] as a proof of concept: it can self-replicate but does no harm to the phones. Since then, more than a hundred smartphone viruses have come into existence, many of which contain malicious codes and cause various damages to the smartphones. The evolution of smartphone viruses is at a very fast pace, perhaps due to the experience virus writers have gained from the computer and Internet world. As estimated by [13], two years were sufficient for smartphone viruses to evolve to an equivalent of twenty years of work in computer viruses.

Several incidents of smartphone virus propagation have recently been reported [18, 12]. So far the outbreaks have been limited in scale due to the the lack of victims. Smartphone is still in its infancy and only accounts for about 5% of the entire mobile phones in use in the United States. However, smartphone's market share is expected to increase significantly over the next few years with projection as high as 156% annual growth rate, as compared to 10% for average mobile phones [22]. Such explosive growth of smartphones will provide a fertile ground for the viruses to spread.

An infected smartphone can inflict severe damages to both the users and the cellular service provider. To the users, the damage may include the loss or theft of private data, the disruption of normal phone usage and also monetary losses (e.g., the virus may secretly use the SMS/MMS services). On the cellular infrastructure side, the smartphone viruses present a serious threat of Denial of Service, e.g., one that can block the emergency 911 calls [16].

## 2.2 Virus Categorization

It is important to examine and categorize the various smartphone viruses in existence today, because such an understanding would enable us to decide what type of virus is most crucial for our solution to target. There are many ways to categorize smartphone viruses. For example, in [28], these smartphone viruses are categorized based on the targets that the virus attacks (e.g. the call center, the cellular base station), while in [26], the viruses are categorized based on the goal it tries to achieve (e.g. information theft, DoS).

We take on a different categorization approach. Instead of focusing on what the viruses seek to attack or achieve, we choose to categorize the smartphone viruses based on the multiple infection vectors that the virus enters and/or exits the device. The benefit of our approach is that it provides a generic view on how a virus penetrates into a smartphone and how easily it can spread in the smartphone population.

We have identified five categories of infection vectors for smartphone virus, which are listed in Table 1 in a decreasing order of their expected spreading capability. Table 1 also gives some representative viruses currently in existence for each infection vector. Below, we will describe these infection vectors in more detail.

**Cellular Network**: Smartphone viruses can use Multimedia Messaging System (MMS) to spread within the traditionally virus-free cellular network. The most well-known virus of such a kind is *CommWarrior* [21]. By the virtues of its core telephony functionalities, every smartphone is almost always on and always connected to the cellular network, making this infection vector extremely contagious.

**Bluetooth**: Bluetooth virus is innovative in that its spreading does not rely on the existence of any network infras-

| Infection Vector | Examples |
|---|---|
| Cellular Network | CommWarriors, Mabir |
| Bluetooth | Cabirs, CommWarrior |
| Internet over WiFi/GPRS/EDGE | Skulls, Doomboot |
| USB/ActiveSync/Docking | Crossover, Mobler |
| Peripherals | Cardtrap |

**Table 1: Smartphone viruses categorization based on infection vector.**

tructure. Instead, it leverages the mobility of the mobile users and the short range wireless connectivity to directly infect nearby Bluetooth users. It is especially contagious in a dense environment, as demonstrated the incidents of *Cabir* outbreak in the World Athletics Championships [18].

**Internet**: Most smartphones are capable of accessing the Internet (via WiFi, GPRS/EDGE or 3G network access), and run the risk of contracting viruses through file downloading from the Internet much like the desktop computers. However, a few distinctions from the desktop computer set this infection vector less potent than the above. First, the screen size and the large keyboard will help the computer remain as the primary web surfing device of choice in an office or home environment. Secondly, the limited bandwidth for Internet access over cellular networks and the cost of GPRS/EDGE/3G access are barriers to the use smartphone users in an outdoor environment. However, a smartphone user can still be lured into downloading files such as *Skulls* and *Doomboot*, disguised as games, and end up getting infected by a smartphone virus.

**USB/ActiveSync/Docking**: Frequently, smartphones are connected to a desktop computer in order to synchronize calendar events and new contacts. A smartphone virus could potentially penetrate the smartphone in the event of a synchronization as demonstrated by the *Crossover* virus [21]. However, to take this infection vector, the virus must first compromise the desktop computer before an attempt can be made onto the smartphone. This requirement makes it significantly more difficult for the smartphone virus to reach a large audience.

**Peripherals**: Similar to desktop computers where viruses used to exploit the floppy disk to spread, smartphone viruses also demonstrated that they are capable of going the same route, as shown by *Cardtrap*. However, similar to the floppy disk virus, this infection vector has limited spreading capability and most likely will fade out before a major outbreak.

In this work, we focus on the first two categories, i.e., those viruses that spread through cellular messaging systems or Bluetooth. These two infection vectors are not only the most popular ones among existing smartphone viruses, but also the most dangerous ones, because they are unique to smartphones and have strong spreading capability. Thus, it is critical to have a security solution that can effectively combat these viruses.

## 2.3 Implementing Viruses on Windows Mobile

Next we use real implementations to investigate, and provide insights on the difficulties for a novice to develop a virus on Windows Mobile smartphones. Unfortunately, our study shows that the increasingly popular Windows Mobile platform is just as vulnerable, if not more than Symbian OS smartphone.

### 2.3.1 Application Unlock

The first step in developing applications for a Windows Mobile smartphone is to configure the device into the Application Unlock mode, i.e., a privately developed software program can be installed and run on the phone. Before diving into the detail of Application Unlock, we will first provide some insight regarding application locking.

Microsoft (Mobile2Market [9]), along with Symbian (Symbian Signed [10], has taken the approach to implement security policies into the smartphone in the form of application certification. The goal of the application certification is to disallow unknown applications from installing, executing, or accessing certain privileged APIs on the smartphone. In theory, such an approach can prevent viruses from infecting a smartphone. Unfortunately, this approach, in practice, has several associated drawbacks. First, software bugs and vulnerability may still exist on the smartphone which allows viruses to bypass the application certification. Secondly, it faces the greatest deterrent of economic pressure. Getting an application certified by the proper authorities is a time consuming process and usually involves high fees. Thus, it comes as no surprise that the open-source community is strongly opposed to such certification requirements. The consumers also feel frustrated because they cannot install and run freewares that provide similar functionalities as the commercial software. Finally, the age-old and very popular practice of SIM unlocking the phone (i.e. to use a competing carrier's SIM card on the phone) usually requires the phone be first application unlocked. As a result of such pressures from the public, either the cellular carrier (e.g., Orange [8]) gives up and allows the users to application unlock their phones, or the open-source community develops its own methods to application unlock the phone.

We have found some online resources to application-unlock the particular smartphone used in our implementation, Dopod 577w [5] . The procedure is relatively simple: We downloaded regeditSTG and SDA_ApplicationUnlock, and follow the well-documented user guides [11]. The whole Application Unlock process takes no more than 10 minutes and requires no knowledge about the inner details of a smartphone. For most smartphone models, numerous application unlocking tools and user guides are available to the public on the Internet. As a result, the bar for unlocking the smartphone is set fairly low for general users.

### 2.3.2 Implementing Smartphone Viruses

The existing smartphone viruses mostly target the Symbian OS, and only a few of them can infect other smartphones running Windows Mobile. This is not surprising because Symbian has dominated the current smartphone market. In summary, virus writers seek as many victims as possible, while currently there are more Symbian-based smartphones than Windows Mobile-based ones. However, Windows Mobile has recently gained increasing popularity, and very likely this trend will continue in the foreseeable future. Looking into the future, we ask the following questions: How secure is Window Mobile , and how difficult is it is even a novice to write viruses on it?

In what follows, we describe our implementation efforts in writing viruses on Windows Mobile, which emulate some existing viruses on Symbian. In particular, our newly created virus mimics *Cabir* [21] and *Flexispy* [6]. In this sense, one can think that we are "exporting" these viruses from

Symbian to Windows Mobile. However, the actual implementation is not trivial because we do not have the source codes of those of Symbian viruses, and because Windows Mobile provides a very different set of capabilities and APIs from those in Symbian. Our experience shows that, as long as the device is application unlocked, it is not only feasible but fairly easy to write viruses on Windows Mobile.

For a Cabir-like virus, it can be achieved using the APIs associated with OBEX programming. The basic steps involve first turning on the Bluetooth interface using the *BthSetMode()* function call. Next, create and initialize an IObex object that will handle all the actual communication between two Bluetooth enabled devices. Once the IObex object is ready, it can be used to discover nearby devices using the *StartDeviceEnum()* function call. The returned list of devices includes each device name and Bluetooth MAC address of a potential victim. A victim is selected from the list and as the last step, the file for transmission, i.e. the virus executable, is retrieved from its current directory and sent using the IObex's *put()* function. To behave similars to Cabir, one would only need to include an additional for-loop to enclose the code from device discovery to the sending of the virus executable. The entire executable is under 8 KB and the code is tested against a HP iPaq with Bluetooth enabled. The test shows that the iPaq will receive the virus executable and display a message asking the users whether it should be saved. If the user respond with a Yes, the executable is saved in the "My Documents" folder. With some social engineering (or worst software exploits) a user may navigate to the folder and execute the received program. From what we observed on the Smartphone, once the program starts running after initial warning, there is no visual indication that shows the user that program is running in the background. Even navigating to the Task Manager, where typical applications can be stopped, there is no display of the Bluetooth virus program that we created.

The above virus implementation is just a proof of concept. As we do not intend to create a smartphone virus with real damages, we did not further enhance our codes. Currently, when the phone is rebooted, our Bluetooth virus will cease to take effect. However, it would be fairly easy to take additional steps and package our executables into a cab file, which is essentially an installer for smartphone programs. Running the cab file will allow us to place several components into the smartphone so that the virus program will automatically restart when the smartphone is rebooted.

For the messaging virus, we have tried to emulate existing Symbian-based viruses that exploit the SMS capabilities to target a specific destination for financial gain (e.g. Redbrowser) or steal private data (e.g. Flexispy). Such a messaging virus can be implemented in two steps: a) gathering information on the local device, then b) creating and sending out SMS messages. There is much valuable information on a smartphone that the virus may collect. For example, the ability to query the pocket outlook allows a virus to retrieve private information, such as the itinerary of the user from the user's "calendar", the to-do list from the "task" and most importantly, the information of each contact on the user's "contact list". While a typical mobile phone can only store the phone number of the contacts, a smartphone contact list typically contains much more information including, but not limited to, the contact's birthday, email address, home/work address, Company, Job title, etc.

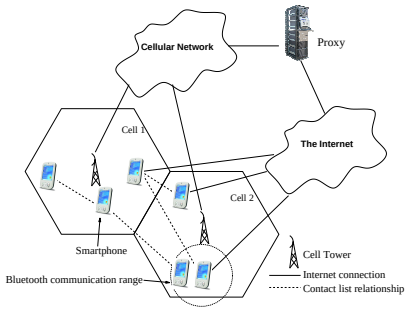


**Figure 1: The architecture of *SmartSiren***

Moreover, outside the pocket outlook, the users may store important PIN numbers or passwords for ATMs and financial accounts in text files. After gathering such valuable information, it is easy to compose an SMS message using the *SmsOpen()* and *SmsSendMessage()* API.

## 3. CHALLENGES

Traditionally, the malwares on the Internet are handled by network-side defense, in the form of firewall and intrusion detection, and endhost-side defense using anti-virus software. The network-side defense in the smartphone context faces new challenges because viruses can spread without the reliance on the network infrastructure, e.g., through Bluetooth interfaces. While anti-virus software will continue to play a central role in defending against smartphone viruses, it has several limitations. In particular, the majority of existing anti-virus software rely on an up-to-date virus signature database to detect malwares. If the virus signatures is outdated, its effectiveness diminishes. This can happen when a new malware emerges and the anti-virus researchers have not yet identified its signature. Moreover, a smartphone may not have 24×7 Internet connectivity. As a result, even when the virus signature is available, the smartphone may not be able to obtain it in a timely fashion.

In general, there are several key differences between smartphones and computers that impact the anti-virus solutions. First, while a computer is primarily connected to the Internet via IP networks, a smartphone also connects to the cellular network through SMS/MMS services, as well as its Bluetooth interface that is frequently used to interact with other devices (e.g. headset, GPS, or even other smartphones). These interfaces are quickly becoming the new infection vector for viruses, which makes the smartphone susceptible to get infected even when it is disconnected from the Internet. Secondly, a smartphone is highly mobile and always on, resulting in a greater degree of difficulty in quarantining the virus in a local region. Lastly, a smartphone has limited processing power and storage capacity, which limits the effectiveness of a complex and on-device anti-virus solution.

## 4. DESIGN

In this section, we present *SmartSiren*, a collaborative virus detection and alert system for smartphones. Our basic idea is quite simple: The outbreak of viruses must affect many smartphones and cause noticeable changes in their behavior. Thus, we can achieve early detection of viruses by keeping track of the device activities even in a coarse granularity. However, this seemingly simple idea presents

many research challenges in terms of its *feasibility, effectiveness and efficiency*. For example, what activity information is available at the devices, and how is it affected by various viruses? How can we differentiate the behavior changes caused by viruses or inherent system dynamics? When a potential virus outbreak is identified, which devices should be alerted and how?

The rest of this section will address these questions in detail. We overview the architecture of *SmartSiren* in Section 4.1, then describe its information collection and virus detection processes in Sections 4.2 and 4.3. Finally, we describe the targeted alerting mechanism in Section 4.4.

## 4.1 Architecture and System Components

The architecture of *SmartSiren* is illustrated in Figure 1. The system consists of a large set of smartphones that want to be protected from potential virus outbreak and a proxy that interacts with the smartphones through either cellular networks or IP-based Internet connections. Each smartphone runs a light-weight agent that logs the device activities, e.g., the usage of cellular SMS service and Bluetooth interface. These logs are periodically reported to the proxy. Upon receiving such reports from the smartphones, the proxy performs per-device viral behavior analysis as well as aggregated system-wide viral behavior analysis, and identifies each smartphone as either healthy or infected. When the viral activity has been verified, the proxy alerts the infected smartphone users about the suspicious activities. In addition, the proxy also alerts other smartphone users that may immediately be vulnerable to infection attempts from those already infected devices.

The motivation of such a proxy-based architecture is twofold. First, the smartphones may have only limited resources in terms of computation, storage, battery power. By leveraging a powerful proxy, we can offload most of the processing burden from the devices to the proxy, thus minimizing the performance penalty on the smartphones. Secondly, for accurate virus detection and prompt alerts, the smartphones must collaborate with each other. The use of a centralized proxy can greatly simplify such collaboration. For example, the proxy can detect system-wide viral behavior by performing joint analysis on reports from different devices, which cannot be easily done at each single device.

Of course, the centralized proxy also manifests itself as a performance bottleneck and a single point of failure in the system. To improve the scalability and resiliency, one can extend the architecture with multiple proxies in a flat or hierarchical structure. However, we leave such extensions to future work, and focus on the single-proxy case in the remainder of this paper.

### 4.1.1 Smartphone Agent

In *SmartSiren*, each smartphone runs a lightweight agent that assumes minimal functionalities on the host device. The reason is because the smartphones on the current market have very diverse capability in terms of processing power, storage capacity and communication interface. A lightweight agent with minimum device requirements can allow most, if not all, smartphones to use our system. In return, the more smartphones participate in our system, the better the quality that the proxy can achieve in performing viral behavior analysis. Specifically, our smartphone agent consists of the following four modules:

**Logging**: This module logs assorted activities on the device. We focus on the viruses that spread through the use of messaging or Bluetooth interface, thus the logging module mainly records information that is related to communications over these two interfaces. In section 4.2, we will present more details on exactly what information is logged for messaging and Bluetooth respectively.

**Privacy Protection**: Simply reporting the phone activities to the proxy is unacceptable to many privacy-conscious users. The privacy protection module addresses this concern by allowing the device to report in an anonymous manner. On the other hand, it also prevents a virus or an attacker from abusing the privacy protection and injecting bogus reports. The details will be presented in Section 5.

**Reporting**: This module decides when the device should report its activities to the proxy. As we will show in Section 4.2, there are two types of reports: *daily* and *pro-active*. In normal cases, a device sends one report every day to the proxy; however, when abnormal behavior has locally been observed, the device immediately sends a pro-active report to the proxy.

**Communication**: This module handles the actual communication with the proxy. In our design, we make a minimal assumption about a smartphone's communication capability, as our primary communication methods are SMS messaging and IP channels. SMS has been available to all mobile phones since the 2G Cellular networks, and all smartphone nowadays are able to access the Internet through IP-based connections. Despite that such capability is already supported by the devices, different users may have different network availability. For GPRS/EDGE subscribers, IP communication can be achieved anytime, anywhere. However, non-subscribers may have to rely on WiFi access at home or at work.

The IP-based communication is easy to implement. For the SMS-based communication, we leverage the SMS gateway on cellular networks, which can convert a SMS message to/from the mobile phone and an email from/to an Internet mailbox. As such, the proxy only needs to send or receive emails from the specified address. For example, with T-Mobile, a mobile phone can send a SMS message to an email address by setting the recipient number as "500". The first word of the SMS body is the destination email address. Separated by a white space, the rest of the SMS body is the email content. The payload limit of a single SMS message can be up to 160 bytes. On the other hand, when the proxy needs to contact a smartphone, it can send an email to the phone's SMS address. The email is addressed to <phonenumber>@tmomail.net, which is received by T-Mobile's email gateway and then delivered to the mobile phone as SMS or MMS messages. Other cellular carriers, such as Cingular, also have similar mechanisms in place, though the setup (e.g., the domain name) may be different. Even better, a generic method to reach all mobile phones in US is currently available using <phonenumber>@teleflip.com.

### 4.1.2 Proxy

The primary functionality of the proxy is to perform analysis on the data collected from the smartphones, and to determine which smartphones should be alerted when a possible virus is detected. Specifically, the proxy consists of the following four modules:

| Category | Value |
|---|---|
| Phone Number | 555-123-4567 |
| Email Address | john.doe@domainname.com |
| Network | T-Mobile |
| Phone Model | Dopod 577w |
| Bluetooth | Yes |
| OS type | Windows Mobile |
| Contact List Info | Jane: 555-321-7654<br>Bob: 555-213-6745<br>.... |
| Mobility Profile | Cell ID 1234<br>Cell ID 9971<br>Cell ID 756 |

**Table 2: Static configuration information collected from a smartphone during the registration.**

**Report Collection**: This module interacts with the smartphone agents to collect information aggregated from each smartphone. The primary means of communication includes Cellular-Email and IP networks. Many cellular networks can support a smartphone to compose SMS and deliver the message to an email address [19]. If IP channel is available, the smartphone can also submit its report over IP networks. The proxy stores the received reports for further analysis.
**Privacy Protection**: This module works together with the privacy protection module on each smartphone agent. The details are in Section 5.
**Data Analysis**: This module performs joint analysis on the aggregated smartphones' reports, in order to detect whether a virus is currently spreading in the smartphone population. The specific analysis techniques will be presented in Section 4.3.
**Alerting**: Once a virus outbreak is identified, this module attempts to alert the infected devices using SMS, so that the respective user should take action to rectify the infected devices. In addition, it also alerts those uninfected devices that may be in direct contact with an infected device. Such an alert will instruct the smartphone agent to filter the virus infection attempts through SMS or change the Bluetooth interface into non-discoverable mode.

## 4.2 Collecting Information

In our system, a smartphone registers its static configuration with the proxy during the initial registration, while continuing to report its dynamic activity to the proxy during the operational phase. Below we describe the information collection process in detail.

### 4.2.1 Registering Static Configuration

When a user joins our system, he/she needs to first register with the proxy. Table 2 summarizes the list of information that the user may provide during the registration. The minimal information required is a valid mobile phone number and the cellular carrier. The mobile phone number will serve as the user identification, as well as the recipient address when the proxy needs to communicate with the phone (e.g., sending alerts) using SMS. The other information is optional but can help both the accurate detection for the proxy and the targeted alerting service for the user. In particular, the phone model helps the proxy to determine the general capabilities of the smartphone. The Bluetooth capability is indicated so that a Bluetooth-incapable device will never receive alerts for Bluetooth viruses. The operating system running on the smartphone is another parameter that decides the scope of the alerts.

During the registration, the user has the option to provide additional information about his/her contact list (i.e., the phone book ) and mobility profile. The contact list is valuable to the proxy in detecting viruses that spreads through the messaging channels. Moreover, in cases where the user's smartphone is infected with a virus, the proxy can issue alerts to the entries on its contact list. The mobility profile is a list of locations that a user frequently visits. Here the location is identified based on the cellular network's cell tower id. The proxy uses such mobility profiles to issue targeted alerts to the users in a specific area when a Bluetooth virus is discovered. Thus, registering the mobility profile helps users to receive only targeted alerts for the relevant locations instead of receiving a wide range of alerts.

Each registration must be authenticated by having the mobile phone directly reply to the proxy's authentication request through a SMS message. This step ensures the existence of a valid SIM card and prevents the malicious attacker from registering a large number of fake identities in order to submit fabricated reports to confuse the proxy.

### 4.2.2 Reporting Dynamic Activity

As discussed earlier, each active smartphone in *SmartSiren* runs an agent that logs the communication activities and reports to the proxy. For now we simply assume that a user is willing to disclose his/her phone usage and activities to the proxy. In Section 5, we will describe the mechanism to protect the user privacy.

The information carried in each report is summarized in Table 3. Basically, it describes the communications that have occurred over the SMS messaging and the Bluetooth channels, which are the primary means for the viruses to infect the device. In our system, the SMS logging is done by checking the "Sent" folders in the MMS, SMS and Email stores on the device. These folders contains all the messages that the user have composed using standard messaging composer available on the phone. On the other hand, the logging of Bluetooth activities is achieved using the base properties of the *ConnectionsBluetoothCount* and *ConnectionBluetoothDescriptions* in the Windows Mobile operating system.

There are two cases in which the device submits a report. First, the device submits a report daily that describes past communication activities on both the messaging and the Bluetooth interfaces. The daily report can be submitted either using SMS or over the Internet, e.g.,at night, when the user has Internet connection at home. Secondly, each device also keeps a long-term average (i.e. 7 days moving average) of its traffic volume over both messaging and Bluetooth channels. Whenever the traffic since the last daily report exceeds the long-term average of daily usage plus one standard deviation, the device immediately sends a report to the proxy using SMS, which is always available and provides instant delivery.

The reason for having two different reporting methods is to handle two types of viruses, namely trojans and worms, separately. A trojan virus may hide in the device and passively utilize the resources over an extended period of time. For example, *Flexispy* [6] records the user activities and tries to send such information to the *Flexispy* server using SMS. The periodic reports can be used to detect such passive tro-

| Category | Value |
|---|---|
| Identity | 555-123-4567 |
| Authentication | digital signatures |
| [Optional Privacy] | Submission Tickets |
| Log Date | Dec 4, 2006 |
| Mobility Profile | CellID 1234<br>CellID 756<br>CellID 3215 |
| Message Sent | 555-111-1111<br>555-222-2222<br>555-333-3333<br>.... |
| Bluetooth Sent | 11:11:11:50:11:11<br>22:22:22:22:22:22<br>33:33:33:33:33:33<br>.... |

**Table 3: Dynamic activity information reported by a smartphone during the operational phase.**

jans, which do not exhibit strong epidemic behavior. On the other hand, a worm virus may aggressively replicate itself in a short period, thus pro-active reporting is necessary for the system to respond in a timely fashion and quarantine the outbreak.

## 4.3 Joint Detection

The detection phase consists of the proxy sifting through the collected data and trying to pinpoint suspicious behaviors that are atypical of normal user activities. Our proposed solution includes the use of statistical average and threshold to discover the rise in average messaging volumes. In addition, we lure the virus into performing abnormal activities by inserting bait entries in the user contact list. Note that, leveraging users with antivirus software installed to report an incident is also an viable option for use with *SmartSiren* framework.

In this work, our proposed framework aims to detect those malwares that exploit the communication capabilities of a smartphone. More specifically, we target the viruses that make access to either the messaging channel or the Bluetooth channel. Smartphone viruses utilizing these two communication channels have generated a tremendous amount of buzz due to their novelty as well as their potential to spread to a large number of handsets. Our general approach is to, first, systematically analyze the goals of a communication-based smartphone virus, then comprehensively evaluate what communication target the virus must select to achieve its goal. Finally, based on the enumerated communication target selection strategies for the smartphone virus, we will respond with two detection plans, *statistical monitoring* and *abnormality monitoring*. When used in conjunction with the information collected from the smartphone, it can accurately achieve viral activity detection.

The communication-based malwares can be loosely divided into two categories: those that spread (i.e. communication contains viruses) and those that do not (i.e. communication contains content). However, these two categories are not mutually exclusive as it may be the case that a spreading virus carries a malicious payload that further accesses the communication channels for other purposes (e.g. a spreading virus that cause the smartphone to spam other mobile phone users). The goal of the content-based virus can be further divided into two subcategories: those that steal

| Targets | Virus Communicated | Content Communicated | |
|---|---|---|---|
| | | Personal Information | Unrelated Information |
| From-Virus | DoS | [Flexispy] | [Redbrowser] |
| From-Device | [CommWarrior] | Info Leak | Spam |
| Randomized | Infection | [PBstealer] | Spam |

**Table 4: Examples of existing viruses are shown in brackets, otherwise the potential damage is stated.**

personal information from the device and those that deliver device unrelated contents for either spamming or causing usage charges to the users (e.g. to a premium number).

Independent of the virus goals, there are three general strategies for viruses to acquire communication targets: From-Virus, From-Device and Randomized. From-virus means the virus brings a pre-defined set of targets with it to the victim. From-Device indicates that the virus extract the set of target from the infected smartphone. Randomized does not include a specific set of the target. Instead, the target may randomly generated or happens to connect by chance. Table 4 breaks down the specific types of attack that can be achieved and illustrates their ideal target acquiring strategy. Our solution responds to various virus goals through logging the communication end point of the smartphone and perform a general *statistical monitoring* based on the volume of the communication and a more specific *abnormality monitoring* leveraging the user contact list and user mobility profile.

### 4.3.1 Statistical Monitoring

*Statistical monitoring* is a general approach that attempts to discover communication-based smartphone viruses when these viruses over-utilize the smartphone's communication capabilities. This approach is geared toward fast spreading worms as such malware would typically result in wild fluctuation from normal usage. Specifically, for each user, based on the user-submitted communication log, the proxy would keep track of the average number of communications that each user initiates each day using a 7 days moving average window. We define $U_{thresh}$ as the summation of the 7 days moving average and its standard deviation. $U_{thresh}$ captures the normal usage of each users. In addition, each day the smartphone users agent will count the number of communication that the users has initiated as $U_{today}$. When the user's daily usage $U_{today}$ exceeds $U_{thresh}$, the user would be moved from normal state into over-usage state and a report would immediately be submitted to the proxy over the SMS channel. While the communication access of a smartphone virus can certainly help push $U_{today}$ over $U_{thresh}$, the over-usage state does not guarantee that a particular handset is infected.

In addition to monitoring the $U_{thresh}$ for each user, the proxy also monitors a global equilibrium, $P_{avg}$. $P_{avg}$ is defined as, on average, each day, of the entire smartphone population, how many users would exceeds their $U_{thresh}$. $P_{avg}$ is an indicative equilibrium. When the daily count of $P_{today}$ exceeds wildly from $P_{avg}$, it can suggest that an aggressive viral outbreak has occurred. To allow some natural fluctuation of the $P_{today}$ from $P_{avg}$, we apply a Detection Threshold Multiplier (DTM) of 2 to $P_{avg}$. So for a particular day, when $P_{today}$, the count on the number of users

exceeding their $U_{thresh}$, becomes greater than $2 * P_{avg}$, the proxy would deem this as an aggressive viral outbreak and proceed to the alert phase on the group of smartphone exceeding their $U_{thresh}$ in hopes of halting the outbreak.

### 4.3.2 Abnormality Monitoring

Statistical monitoring is effective in observing sudden and wild change in behavior from the user population, and thereby possesses the ability to detect fast spreading worms. Slower worms, on the other hand, may evade detection by flying under the radar. To combat slow infecting worms, we further propose abnormality monitoring to complement statistical monitoring.

In abnormal activity monitoring for messaging, we observe that, of all current smartphone viruses that utilize the messaging system, they often need to select a communication target to contact. The goal of the communication typically can be divided into two categories. First, the virus may wish to achieve some form of financial gain or information theft, as in the case of Redbrowser [21] and Flexispy [6]. Secondly, the virus would like to replicate and infect other smartphones such as CommWarrior [21]. Note that viruses need not be mutually exclusive as CommWarrior could be designed to have a payload that aims to achieve targeted messaging attack for financial gain or information theft. We further observe that, to setup the target, the virus either hardcodes a communication destination into the body of the virus, or retrieves the victim's information from the user's contact list. To address viruses belonging to the first group, the proxy would actively maintain a top messaging destination list. For each destination, a counter will keep track of the number of messages directed to this particular destination. When a destination becomes highly ranked, the proxy would move onto the alert phase. For the second group of viruses that utilize the on-device contact list, we utilize a dummy entry approach to bait the virus into sending a message into a non-existent mobile phone. Since normal users would not typically message an invalid entry, logging such an event will provide a strong indication that a virus has infected this smartphone.

## 4.4 Alert

In the alert phase, once the detection has been confirmed, the proxy will begin issuing targeted alerts to the smartphone population based on the information that they registered in the bootstrap phase. In particular, the proxy will alert the users of their infected status so that users can take action to disinfect the unit. For each infected user, the proxy will also attempt to warn the smartphone units that can come into contact with the infected units either physically (Bluetooth) or logically (messaging).

In the previous section, we presented the joint detection scheme for our proposed *SmartSiren* framework. Once the proxy has identified viral behavior in the smartphone population, the proxy will begin issuing alerts to the smartphone population. A total of four types of alert will be issued as summarized in Table 5. The message type $B_I$ and $M_I$ will be directly issued to the units that reported viral behavior. For connected units, those that are logically linked to the units through contact list or physical proximity, the message of $B_C$ and $M_C$ is issued.

Our alerting scheme for the Bluetooth connected units using $B_C$ is as follows: based on the infected unit's mobility profile $MP_{infected}$, the proxy will alert all other units with mobility profile $MP_{uninfected}$ that share the same location as those in $MP_{infected}$. For example, if infected unit I's mobility profile includes location A, B, C and an uninfected unit U's mobility profile includes location B, F, H, then U will be alerted since U and I share location B.

| | Bluetooth Virus Alert | Messaging Virus Alert |
|---|---|---|
| Infected Units | $B_I$ | $M_I$ |
| Connected Units | $B_C$ | $M_C$ |

**Table 5: Types of alert messages.**

The alerting scheme for the messaging connected units using $M_C$ is as follows: for each infected units, all mobile phone entries on its contact list that have also registered with the proxy for service are alerted with message $M_C$. Appearing on the list of devices that the proxy monitors indicate that the entry is indeed a smartphone and has the capability to process message $M_C$. Otherwise, the unknown mobile phone entries are sent a generic SMS warning message.

The alert will reach the smartphone in the form of SMS messages. The smartphone user agent will react to the alert differently based on the type of alert received. Table 5 shows the type of alert messages that a smartphone user agent would receive.

When a smartphone user agent receives a message of type $B_I$, it means that this smartphone is believed to be infected with a virus. The smartphone user agent will attempt to shut down the Bluetooth interface and display a visual alert to the user showing the type of Bluetooth activities that has been recorded. If the activities are inconsistent with the smartphone user's Bluetooth usage, actions should be taken by the user to rectify the problem.

When smartphone user agents receives a message of type $B_C$, it means that this user frequents a location that is also visited by one of the infected users. Upon reception of the message, the user agent will record the region indicated by the proxy. The smartphone agent will keep a watch on the current user location. When the user enters the indicated region, the smartphone user agent will switch to non-discoverable mode to avoid communicating with the infected unit.

Reception of message $M_I$ indicates that the unit is infected. Ideally, an infected unit should shut off the outgoing messaging interface to prevent the further spread of the virus. However, Window Mobile currently does not offer an API to intercept outgoing messages. Therefore, we have opted to simply display a warning message along with the recorded messaging log for users to verify the suspicious activities.

Reception of message $M_C$ indicates that this smartphone is on the contact list of a user that is infected. Windows Mobile does offer an API to intercept incoming messages and since the infected unit is unable to stop outgoing messages, we have opted to have the $M_C$ receiver to intercept and only filter out infected users messages. Furthermore, the smartphone user agent will try to remove any unread message from the infected users.

# 5. PROTECTING USER PRIVACY

Privacy is a critical issue that impacts the incentive of the users to join *SmartSiren*, because the information it col-

lects from the smartphone may contain sensitive information, e.g., call records, SMS records, and network usage. In practice, many users are reluctant to reveal such private information to the proxy. Without proper protection of user privacy, it is difficult for *SmartSiren* to attract enough users and reach the critical mass for accurate virus detection.

It is important to define what privacy we can protect. Clearly, to perform any meaningful detection or alert operation, the users must sacrifice some privacy. For example, in order to be notified when an acquaintance's phone or a frequently visited spot is polluted by viruses, users must register their contact list and mobility profile with the proxy. Therefore, *SmartSiren* does not attempt to provide perfect privacy. Instead, it only ensures that the proxy cannot infer any user's daily activity from the collected data.

This, by no mean, implies that the privacy of static configuration data is not important. If a user is concerned about the privacy of such data, he/she can simply choose not to register it with the proxy, at the cost of sacrificing some targeted alert services. As such, *SmartSiren* leaves users the flexibility to trade off the services they receive for the privacy they are willing to sacrifice for the static configuration.

The privacy of dynamic activity, on the other hand, is non-trivial to achieve due to the need for joint detection. There are two basic approaches to protecting privacy: *obfuscation* that hides the actual data in the reports (e.g., by encryption or hashing), and *anonymization* that hides who submitted these reports respectively. Clearly, obfuscation is not suitable in our context, because the proxy can hardly perform joint analysis on the obfuscated data. However, anonymization is also problematic because it may be abused by viruses or attackers to inject bogus reports and damage the trustworthiness of detection results.

To ensure both user privacy and authenticated reports, we present an anonymous and ticketed report submission scheme. In this scheme, each smartphone can submit its reports in an anonymous manner, but each report must carry a unique cryptographic ticket. To prevent the proxy from linking a ticket to its owner's identity, the smartphones exchange their tickets using a proxy-oblivious scheme. In the rest of this section, we describe anonymous report submission in Section 5.1, the ticketing mechanism in Section 5.2 and the anonymous alert strategy in Section 5.3.

## 5.1 Anonymous Report Submission

To protect the user privacy, we allow a smartphone to submit its activity reports in an anonymous manner, i.e., without revealing its phone number or user identity. This is achieved by leveraging the diversified communication channels available to the devices. As discussed in Section 4, a smartphone can communicate with the proxy directly using SMS, Email, WiFi, GPRS, or indirectly with the assistance of a computer. Based on how the device obtains its network address, these channels can be classified as either *cellular-based*, such as SMS and Email, or *IP-based*, such as WiFi, GPRS, and indirection from computers.

To ensure anonymity, the device submits its activity reports only through those IP-based channels, unless no such channel exists. Because a user never registers any associated IP addresses with the proxy, the proxy cannot differentiate who sent a report if it was received from IP networks. The proxy may record the source IP addresses in the received IP packets, but in many cases, a device acquires its IP address from DHCP, thus very likely uses different addresses for different reports. Note that it is possible that the device may temporarily have no IP-based connection. In such cases, it can either defer the report submission until an IP-based channel is available, or immediately submit the reports using cellular-based channels such as SMS. The tradeoff is between the privacy and the timeliness of report delivery, because the cellular-based channels are always available but reveal the sender's phone number.

## 5.2 Ticketed Report Submission

While the anonymous report submission protects user privacy, it may be abused by the viruses or attackers to inject bogus reports, possibly in large amounts, to disrupt the virus detection. For example, an attacker may forge many reports of abnormal activities and trigger system-wide false alarms. Moreover, a virus may forge many reports of seemingly normal activities to disguise its actual behavior. In order to defeat these attacks, the proxy must be able to differentiate a forged report from the legitimate ones. However, we cannot employ the traditional authentication techniques such as MAC or digital signatures, because they reveal the sender's identity, hence the user privacy is lost.

To address the dilemma between the accountability of the reports and user privacy, we propose a ticketing scheme that can achieve both weak report authentication and user privacy. It is important to note that our design assumes the proxy is *curious but honest*. That is, the proxy is allowed to analyze the traces and try to infer private information, but it must correctly follow the protocol. If the proxy is malicious or compromised, the entire *SmartSiren* system is broken, the defense is beyond the scope of this paper.

**Ticket Distribution** In our scheme, each legitimate report must carry a valid ticket; otherwise, the proxy will not accept it. The tickets are generated by the proxy and distributed to the smartphones. For this purpose, the proxy picks up a secret key $K_r$ and a secure hash function $H$. A ticket is simply the tuple

$$(tid, t_e, H_{K_r}(tid|t_e))$$

where $tid$ is the unique ticket id, $t_e$ is the expiration timestamp, and $H_{K_r}(tid|t_e)$ is the hash result of applying $H$, keyed by $K_r$, on the concatenation of $tid$ and $t_e$.

Each ticket can be used only once. To sustain the submission of reports, the proxy periodically distributes a set of tickets to each smartphone. For example, if a smartphone submits one report everyday, the proxy can distribute 30 tickets to it once per month. This can amortize the overhead of distributing tickets. With such a local ticket repository, a smartphone simply appends an unused ticket before it submits a report, and then discards this ticket. On the other hand, for each distributed ticket, the proxy keeps its status of either "fresh" or "used" until its expiration time is reached. Upon receiving a report, the proxy first checks the validity of the embedded ticket, and accepts the report only if the ticket carries correct hash value, has not expired and has not been used before.

The use of cryptographic tickets ensures weak authentication in that each device can only submit one valid report in each reporting period. However, it cannot ensure privacy because the proxy can record which device each ticket is distributed to. This way, when the proxy receives a report, it can find the sender's identity based on the ticket. Fortu-

nately, in our scheme, the tickets are not associated with the devices, thus we can shuffle the tickets among the devices using the following ticket exchange scheme.

**Ticket Exchange** The goal of ticket exchange is to prevent the proxy from knowing which device holds what tickets. One could possibly have a distributed protocol for the devices to exchange tickets, e.g., using P2P networks. However, for simplicity and efficiency, we take a centralized approach in which the exchange is assisted by the proxy in an oblivious manner.

In our ticket exchange process, there are two types of devices, namely *traders* and *tradees*. When a smartphone registers with the proxy, it randomly picks up a type with equal probability. That is, roughly half of the devices are traders, and the other half are tradees. The proxy maintains a list of currently active tradees. When a tradee needs to exchange tickets, it simply notifies the proxy, which then adds it into the list. On the other hand, when a trader needs to exchange tickets, it queries the proxy for active tradees using an IP-based channel. Upon receiving such query, the proxy randomly picks up a tradee from the current list and returns it to the requesting trader. The trader then contacts the returned tradee and exchanges tickets through SMS. Once the transaction is completed, the tradee notifies the proxy and has itself removed from the list of active tradees.

Essentially, the traders and the tradees use the proxy's assistance to find each other and facilitate ticket exchange. However, because the traders query the proxy over an IP-based channel, the proxy does not know the identity of the requesting trader. This way, the proxy cannot track the ticket exchange transactions, hence does not know what tickets a device has after the exchange.

However, the above protocol works only if all smartphones are benign. A malicious or infected smartphone can easily disrupt it by repeatedly querying for tradees and exchanging tickets with all of them. As a result, the malicious device can collect a large number of valid tickets and then successfully inject bogus reports. To defeat such cheating behavior, we enhance the ticket exchange process as follows.

**Cheating Prevention** To prevent cheating in ticket exchange, our basic idea is to ensure each device can only trade once within a fairly large amount of time, say one day. This is difficult because the traders' actions (i.e., querying for tradees and exchanging tickets) must be oblivious to the proxy; otherwise, the proxy is able to track the transactions. In what follows, we describe how the proxy can limit the trading rate of a trader, without knowing whom it exchanges tickets with.

Our cheating prevention mechanism is based on *commutative encryption* [24]. For any message $M$ and two keys $K_1$ and $K_2$, a commutative cipher $E$ always satisfies:

$$E_{K_1}(E_{K_2}(M)) = E_{K_2}(E_{K_1}(M)) \quad (1)$$

That is, with a commutative cipher, the order of encryption operations does not affect the result. An example of commutative ciphers is the Pohlig-Hellman algorithm [32]. Let $E^{-1}$ denote the decryption function for $E$.

In our scheme, the proxy and the smartphones each chooses a secret key and never reveals the key to other entities. Let $K_P$ be the proxy's key and $K_A$ be a smartphone $A$'s key. As before, the proxy maintains a list of active tradees, and answer queries from the traders. However, when a trader $A$ receives the reply, which contains the identity of a tradee $B$, it must construct a Transaction Description (TD) message as follows:

$$M = (A, B, t)$$

where $t$ is the current time. $A$ encrypts the message using its own key $K_A$, and sends the ciphertext $E_{K_A}(M)$, *together with its identity*, to the proxy in a Request-for-Encryption (RE) message. The proxy keeps a cache of all REs received in the past $T_m$ seconds, which is the minimum time allowed between two consecutive exchanges for a device. When the proxy receives a RE from $A$, it checks whether the cache already has a RE from $A$. If RE already exists, it simply drops the newly received RE. Otherwise, encrypts the received $E_{K_A}(M)$ using its own key $K_P$. The result, i.e., $E_{K_P}(E_{K_A}(M))$, is denoted by $X_1$. The proxy returns $X_1$ to A.

Now $A$ is ready to initiate ticket exchange with $B$. To do so, $A$ decrypts $X_1$ using $K_A$ and sends the result $E^{-1}_{K_A}(X_1)$ to $B$. On the other hand, $B$ first encrypts the received $E^{-1}_{K_A}(X_1)$ using $K_B$. The result, i.e., $E_{K_B}(E^{-1}_{K_A}(X_1))$, is denoted by $X_2$. Next $B$ sends $X_2$ to the proxy in a Request-for-Decryption (RD) message. The proxy then decrypts $X_2$ using $K_P$ and returns the result $E^{-1}_{K_P}(X_2)$ to $B$. Finally, $B$ decrypts the received $E^{-1}_{K_P}(X_2)$ using $K_B$. The result, i.e., $E^{-1}_{K_B}(E^{-1}_{K_P}(X_2))$, is denoted by $X_3$.

At this time, $B$ is able to verify whether $A$ is indeed a valid trader, because based on the commutative property,

$$X_3 = E^{-1}_{K_B}(E^{-1}_{K_P}(E_{K_B}(E^{-1}_{K_A}(E_{K_P}(E_{K_A}(M)))))) = M$$

This result is not surprising because $A$, $B$ and the proxy each applies one encryption and one decryption operation, using their respective keys. Due to the commutative property, the final result should be exactly the original Transaction Description. Thus, $B$ can check the final result $X_3$ and exchange tickets with $A$ only if it includes both $A$ and $B$.

Clearly, without the assistance from the proxy, a trader cannot complete the above process and successfully exchange tickets with any tradee. Since the proxy only allows a trader to invoke the encryption once every $T_m$ seconds, a trader can exchange tickets no faster than once per $T_m$ seconds. Despite its conceptual complexity, the cheating prevention mechanism is actually light-weight. For each transaction of ticket exchange, it incurs only 5 messages, 3 encryption operations and 3 decryption operations. Such overhead can be further amortized by exchanging multiple tickets in one transaction, so that it is invoked less frequently.

## 5.3 Anonymous Alert Strategy

The proposed anonymous report submission scheme can protect the privacy of the users, but it takes away the proxy's ability to identify and reverse contact the users, which is important in determining how targeted alert should be issued. To overcome this short coming, two strategies can be employed with each having its own tradeoff. First, instead of *pushing* the alert, the proxy allow the users to *pull* down the alert using the anonymous communication channel. The cost of such a scheme is that user must more frequently contact the proxy. Another approach is to include a transient anonymous *contact-back* method when submitting the report. Such an approach will require the users to setup an email address that can perform indirect forwarding of the proxy's message to to the mobile handset's SMS address.

# 6. IMPLEMENTING SMARTPHONE AGENT

We have implemented a prototype system of *SmartSiren* to demonstrate the feasibility of our design. Due to limited space, we only describe the smartphone-side implementation here. The proxy-side implementation is relatively straightforward, as it is based on a PC running Linux.

The smartphones used in our prototype system are Dopod 577w [5], also known as i-Mate SP5/SP5m and Qtek 8300/8310[1]. These phones are equipped with a TI OMAP 850 processor (200 MHz), 64MB ROM, 64MB RAM, and both Bluetooth (v1.2) and WiFi (802.11b) wireless interfaces. They are also capable of SMS/MMS messaging and GPRS/EDGE data connection over cellular networks, and we choose T-Mobile USA [20] as our cellular service provider. The OS on these phones is Windows Mobile 5.0 Smartphone Edition, and we use Visual Studio as development tool.

The smartphone agent needs to perform logging for messaging and Bluetooth communication, as we have described in Section 4. While "Sent Box" already logs the outgoing SMS messages, it is unreliable as its content may be altered. Instead, we leverage the *messaging delivery report* mechanism provided by the cellular network. By turning on the mechanism, through a registry value on the phone, we can request, intercept and then log the delivery report sent back by the cellular network about the success or failure of each SMS message. For Bluetooth interfaces, we leverage the State and Notification Broker API to gather Bluetooth-related information using the base properties of *ConnectionsBluetoothCount* and *ConnectionsBluetoothDescriptions*.

In addition to logging communication activity, the smartphone agent also maintains a mobility profile of the user by logging the locations that the user frequently visits. The location information is based on cell ID that is readily supplied by the cellular network. There are two methods in retrieving cell ID from the smartphone. First, it can be achieved through a hack that reads the data from a specific memory location of the smartphone. In the case of the Dopod 577w, the cell Id information is stored at the address space of 0x8ffb0174. However, the location where each smartphone stores its cell ID is device-dependent. The generic approach that is suitable to most smartphone model and the one that we used in this work is to directly communicate with the GSM modem on the device by issuing AT commands [2]. The AT command for querying the cell ID is "AT+creg?".

One issue in using cell ID to identify location is that different cellular carriers have their own cell tower infrastructure. A user using T-Mobile would perceive a different cell ID from a user using Cingular even when they stand in the same location. To overcome such problem, it is necessary for the proxy to obtain the physical locations the of cell towers from different carriers, so that the cell IDs from different carrier can correlate to a same geographical location. The cell tower locations may be directly provided from cellular carriers, but there are extensive public efforts to create such a database of cell tower locations [3].

In our implementation, the smartphone agent utilizes SMS as the primary means of communication with the proxy. To send a SMS message, the smartphone agent would compose the message and send it using the *SmsOpen()*, *SmsSendMessage()* and *SmsClose()* API provided by Windows Mobile. On the reverse path, the proxy can compose an Email and have it delivered to the smartphone agent as a SMS message as described in Section 4. When the SMS message arrives, the reception is done using Window Mobile's *MessageInterception* APIs. Using these APIs, we can construct a *MessageInterceptor* object that describes the conditions for which the message should be intercepted. In our implementation, the condition for intercepting SMS messages is specified as the inclusion of the proxy's email address in the SMS message body. Only such SMS messages are intercepted by our smartphone agent, while the rest of incoming messages are directly delivered to the Text Messaging Inbox.

[1]Although the name differs under different brands, they are all manufactured by HTC [7] under the code name of HTC Tornado.

# 7. EVALUATION

In this section, we evaluate the effectiveness of *SmartSiren* against both messaging and Bluetooth viruses using trace-driven simulations.

## 7.1 Messaging Viruses

We first consider the messaging viruses that spread themselves over the cellular messaging system. After infecting a smartphone, such viruses may launch further attacks, e.g., sending SMS messages to premium numbers. To drive the simulations, we leverage the 3-week SMS trace that we have collected from a national cellular service provider in India. The trace contains approximately 3.91 million users that have sent or received SMS messages during a 3-week data collection period. The trace records the sender/receiver pair for each SMS message that has been transmitted and should reflect a normal, virus-free cellular messaging system as smartphone virus is not yet a widespread phenomenon.

### 7.1.1 Preventing Messaging Virus Outbreak

We first study whether *SmartSiren* can effectively prevent the outbreak of messaging viruses through early detection and alerting. To understand how a messaging virus propagates in the cellular network without *SmartSiren*, we have conducted several simulations, in which a virus outbreak starts from one single infected user randomly chosen from the user population. The simulated infection behavior of the virus is similar to that of CommWarrior [21]. Once it has infected a smartphone, the virus will send out a copy of itself to each smartphone on the user's contact list. One limitation of our trace is that it does not contain the user's contact list information. To model such contact relationship, we pre-processed the trace and define a contact list for each user as all other users that he/she ever contacted in the trace. We believe this assumption is reasonable and captures most of the contact relationship in practice. Once the virus is delivered to a recipient via SMS, the recipient will open the message after a random delay, from 5 seconds up to 2 hours. This delay reflects the reality that a user may only check his/her phone occasionally, and gives us a conservative estimate on the propagation speed of the virus. Once the user opens a message sent by the virus, his/her smartphone will become infected with a probability of *P(infection)*.

Figure 2 shows the simulation results of virus propagation speed with P(infection)=1, which resembles a virus that exploits software bugs or OS vulnerabilities, or P(infection)=0.5, which resembles a virus that relies on social engineering. It is interesting to note that in Figure 2, a virus with P(infection)=1 can quickly infect the entire smartphone population in less

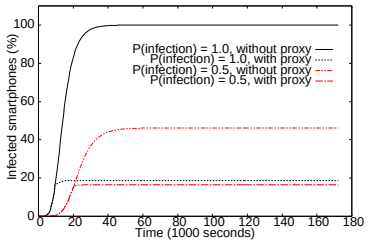


**Figure 2: The spreading trend of messaging virus over time**

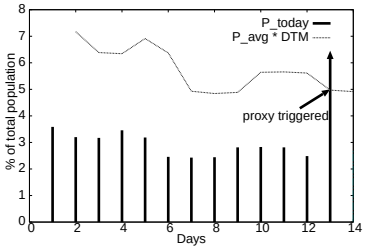


**Figure 3: The top line is the detection threshold and the bars show the percent users exceeding $U_{thresh}$**

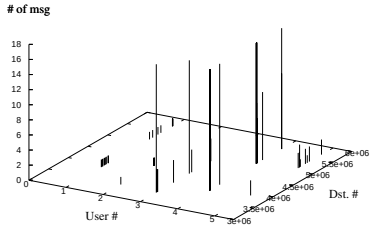


**Figure 4: Daily usage of 5 random users**

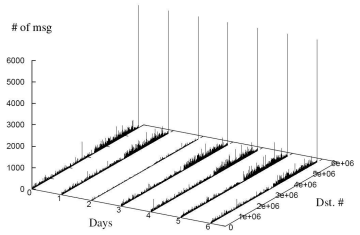


**Figure 5: No. of msg seen by the proxy with 0.1% infected users**

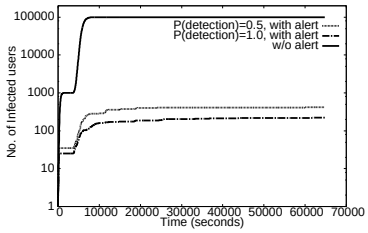


**Figure 6: The spreading trend of Bluetooth virus over time**

than 10 hours, based on our real traces. On the other hand, if a virus mainly relies on social engineering, its spreading capability is limited and often be capped. For example, with P(infection)=0.5, the virus can only infect less than 50% of the smartphones. This is because a significant portion of users is more security conscious and will not open suspicious files embedded in the received messages, which can effectively protect their phones from being infected.

Now we turn on the protection of *SmartSiren* and see how the virus infection speed changes. In the simulation, the proxy uses $P_{avg} * DTM$, where $DTM = 2$ as the threshold for virus activity detection. We empirically chose the DTM value as 2 to tolerate some inherent dynamics in the traffic volume. In the SMS trace we collected, during days when infection has not taken place, we do no observe any instance that $P_{today}$ exceeds the detection threshold. However, on the day (day 13) when the virus is injected, $P_{today}$ count quickly rises above the threshold, resulting in the proxy moving onto the alert phase. Such variation of $P_{avg} * DTM$ and $P_{today}$ is shown in Figure 3.

The performance of *SmartSiren* in containing the virus is shown in Figure 2. The alerts issued by the proxy help the warned smartphone unit to filter out potentially harmful messages coming form the infected users. For P(infection)=1, the smartphones that the virus can eventually infect is limited to only 19.7% of the population, as opposed to the entire population when *SmartSiren* is not available. For P(infection)=0.5, the infection strength of the virus is 18.18% and 44.38% of the smartphone population with and without *SmartSiren*, respectively.

To study how much messaging overhead the *SmartSiren* has consumed, we compare the number of messages that has been delivered in the cellular system. The result is shown in Table 6. Without *SmartSiren*, during the virus outbreak, approximately 15.8 million messages were sent and more than 68% of which are generated by the virus. With *SmartSiren* in place, the total number of message sent is reduced to about 13.1 million. As the virus was quarantined, only about 3.7 million viral messages are injected into the system. *SmartSiren* related messages only consumed about 33.6% of the total messages. Note the large amount of alert messages is necessary in order to "catch up" the virus spread.

| Message Type | without proxy | with proxy |
|---|---|---|
| User | 5027903 (31.72%) | 5027903 (38.2%) |
| Virus | 10823617 (68.29%) | 3703404 (28.2%) |
| Detection | Nil | 879606 (6.7%) |
| Alert | Nil | 3539081 (26.9%) |
| Total | 15851520 | 13149994 |

**Table 6: Message distribution for P(infection)=1.0 and DTM=2.0**

### 7.1.2 Detecting On-Device Abnormality

Once a messaging virus infects a smartphone, it is capable of inflicting various damages on the device, e.g., using the messaging system to deliver private information to a remote third party. Flexispy [6] is a representative example of such privacy-leaking viruses. Other viruses also exist, such as Redbrowser [21], which do not steal personal information but send messages from an infected phone to a premium number in hope of achieving financial gain. Regardless of

their actual intent, these viruses share the same strategy in acquiring the recipient number for its SMS messages, because the virus must bring with it a recipient number, so that the attacker can benefit from the stolen information or receive financial gains.

The benefit of *SmartSiren* in such cases is the provision a bird-eyes view of the messaging behavior of the entire smartphone population, based on the reports submitted by individual smartphones. This bird-eyes view can reveal abnormality that can not be observed locally by each individual smartphone. Assuming that a virus, which acquires information from infected smartphone, has spread out in the smartphone population. Those infected smartphones will only submit a daily report to the attacker. This extremely low volume of traffic will be hard for a single user to detect.

Figure 4 shows five randomly selected users from the trace and their messaging activities for a randomly selected day. In Figure 4, we have inserted a rogue message into each of the five users' record and it is hardly noticeable which message is being sent by the virus. On the other hand, based on the aggregated reports submitted by the smartphone, the SmartSiren proxy can draw a system wide picture of which destination is suspicious due to the high volume of incoming traffic. Figure 5 provides a system-wide messaging activity view as seen by the SmartSiren proxy for five randomly selected days. The erected long line for that particular destination is in fact the rogue message that we have injected into only 0.1% of infected users. The comparison between Figure 4 and Figure 5 shows that *SmartSiren* can easily identify such suspicious activities.

## 7.2 Bluetooth Viruses

Next we evaluate the effectiveness of *SmartSiren* against emerging Bluetooth viruses. Unlike the message viruses that can virtually spread between any two smartphones, a Bluetooth virus only spreads in the host's local vicinity due to the short communication range of Bluetooth channels. Thus, our simulations must be driven by a mobility profile that precisely specifies where the users are at any time. Unfortunately, our SMS traces do not contain such information due to privacy concerns. To overcome this barrier, we leverage the metropolitan-area mobility model developed by [36]. In this model, a city is divided into multiple regions. Each user follows random waypoint mobility model within each region, and probabilistically travels from one region to another (i.e. going to work/home). The probability of such inter-region traversal is extracted from the results in [36].

Specifically, we simulate $100K$ users in a city, which is divided into $10\times10$ zones (each zone is $1000m\times1000m$). These users follow the above metropolitan-area mobility model. Initially, one randomly chosen phone is infected by the Bluetooth virus. An infected phone will infect its local neighbors within 10 meters after 5 seconds if such a neighbor has not turned its Bluetooth interface into non-discoverable mode. Once a smartphone makes an infection attempt, the proxy has a chance of $P(detection)$ to detect it. In case the proxy successfully detects the infection attempt, it sends alerts to the zone where the infection occurs. As a result, all phones in that infected zone will eventually switch their Bluetooth interfaces to non-discoverable mode.

Figure 6 shows the simulation results of number of infected phones over time with P(infection)=1.0. In this setting, we also evaluate the number of infected users under different P(detection). One thing to note is that in figure 6, a virus can quickly infect the entire smartphone population in less than 3 hours, based on the mobility pattern in [36]. On the other hand, the proxy can always catch the infection attempt (P(detection) = 1.0), our proposed solution can limit the number of infected users to 222, which is only 0.222% of the population. Even if our proxy cannot always catch the infection attempt, i.e. P(detection) = 0.5, our proposed solution can still effectively limit the number of infected use to 420. This shows that using mobility profile can efficiently quarantine Bluetooth virus outbreak.

# 8. RELATED WORK

Computer viruses have been plaguing the Internet for many years, and a number of detection and defense mechanisms have been proposed [29, 30, 35, 38]. However, only recently have the smartphone viruses attracted much attention in the research community. The initial studies on smartphone viruses [28, 26, 33] mainly focused on understanding the threats and behavior of such emerging viruses. For example, [28] examines various types of attacks that the infected smartphones can launch, and suggests potential defense mechanisms. The threats of smartphone viruses are further analyzed in [26], along with a categorization based on the goal of these viruses. [33] demonstrates the vulnerability of the SMS system, which can be exploited to launch battery exhaustion attacks.

There are also several recent works on modeling the propagation of mobile viruses. For example, [31] proposes a new probabilistic queuing framework to model the spreading of mobile viruses over short-range wireless interfaces, (e.g., Bluetooth). [25] provides an in-depth examination on the vulnerabilities of smartphones and proposes another model for viruses spreading over short-range radio and cellular messaging system. [34] investigates worms in Bluetooth environment and recommends deploying monitoring systems in high traffic area. Our solution is inline with their recommendations, as each smartphone user agent also serves as a monitoring entity at their respective location.

The vulnerability of SMS services is demonstrated in [27], which that shows the cellular infrastructure can be easily crippled by automated attacks launched from either the Internet or a set of infected smartphones. In a followup work [37], several techniques have been proposed to mitigate the damage of such DoS attacks through queuing techniques inside the cellular infrastructure. While these techniques can mitigate the effects of a congested SMS channel, they cannot eliminate the DoS attacks or quarantine the viruses.

The above studies provided valuable insights on the threats and the behavior of the smartphone viruses. However, the problem of quarantining smartphone virus outbreaks, to our best knowledge, has not been addressed. Our proposed *SmartSiren* system differs from, and complements, the existing solutions [28, 37] in that it requires limited support from the cellular network and protects the smartphones from virus infection through early detection and alerting services.

# 9. DISCUSSION

**User Agent Integrity** The smartphone user agent in our design may potentially become the target of virus attack. The attack may be manifested in one of the two forms: destruction or compromise. For destruction of smartphone

user agent, it can be detected by the lack of reporting from the smartphone user agent. For compromise of the smartphone, an integrity checker [23] can be employed to detect the modification made to the smartphone user agent.

**False Positive** During legitimate events, such as natural disaster, false positive may emerge in our system. These forms of unusual events will be difficult to distinguish from virus outbreak based only on statistical analysis. To address this issue, an user survey system can be employed to query a subset of the suspected user on whether the suspicious activity is intentional. If users confirm the suspicious activity then the proxy will refrain from reaction.

**Contact List Privacy** While some users may not be willing to reveal their contact list. In practice, many users are indeed doing so, provided there are enough incentives. Danger Inc. [4] is better known for its popular handset, T-Mobile Sidekick [20]. All data on the phones, including pictures, contact list and messages are all transparently backed up on the Danger server to allow users to (1) conveniently access them over the Internet using desktop PC, and (2) restorate the data on a new phone in case the old handset is damaged or lost.

**Cell Tower ID** In our design, we leverage cell tower ID as the means to marking user location. Cell tower range can vary from a few kilometers to tens of kilometers resulting in location imprecision. However, this problem can be mitigated by considering multiple cell tower IDs. In fact, as we discovered through our implementation, it is possible to retrieve up to seven cell tower IDs and their respective signal strength from the smartphone device. If combined with some prior knowledge about the location of the cell tower, it is possible to produce higher precision location information. Navizon [14] is one example.

## 10. CONCLUSION

The era of smartphone is on the horizon, and so is smartphone virus. The smartphones are particularly vulnerable to viruses due to their versatile communication capabilities, yet are difficult to harness due to their resource constraints and intermittent network connectivity. As a result, the viruses can easily spread out and cripple both the smartphone users and the cellular and telephony infrastructures.

To this end, we have designed and implemented *SmartSiren* that can prevent smartphone virus outbreaks through early detection and targeted alerts. *SmartSiren* requires limited assistance from the cellular infrastructure and poses minimal processing overhead to the smartphones. While the users can enjoy the targeted virus alert services, their privacy is also protected. The feasibility and effectiveness of *SmartSiren* have been confirmed by both real implementations and trace-driven simulations. We believe that *SmartSiren* can be readily used to provide a first line of defense for the smartphone population to combat the emerging viruses.

## 11. ACKNOWLEDGMENTS

We appreciate the constructive comments by our shepherd, Dr. Richard Han and the anonymous reviewers.

## 12. REFERENCES

[1] 29A lab: http://vx.netlux.org/29a/.
[2] 3GPP AT commands: http://www.3gpp.org/.
[3] Cell Spotting: http://www.cellspotting.com/.
[4] Danger Inc.: http://www.danger.com/platform/exp.php.
[5] Dopod 577w: http://www.dopodasia.com/.
[6] Flexispy: http://www.flexispy.com.
[7] HTC: http://www.htc.com/.
[8] http://developer.orangews.com/orgspv/comdefq.aspx.
[9] http://msdn.microsoft.com/windowsmobile/.
[10] https://www.symbiansigned.com/app/page.
[11] http://wiki.spv-developers.com/.
[12] http://www.f-secure.com/weblog/archives/archive-082005.html.
[13] Mobile malware evolution: An overview: http://www.viruslist.com/.
[14] Navizon: http://www.navizon.com.
[15] Phone viruses: how bad is it?: http://www.newscientist.com/article.ns?id=dn7080.
[16] Prank directs phones to call police: http://news.zdnet.com/.
[17] Securing consumer-friendly smart phones: http://news.com.com/.
[18] Sports fans in Helsinki falling prey to Cabir: http://news.zdnet.com/.
[19] T-Mobile SMS-to-Email: http://wiki.howardforums.com/index.php/.
[20] T-Mobile USA: http://www.t-mobile.com/.
[21] Virus Library: http://www.viruslibrary.com/.
[22] Windows mobile business value for mobile operators: http://download.microsoft.com/.
[23] www.sans.org/resources/idfaq/integrity_checker.php.
[24] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *ACM SIGMOD '03*, pages 86–97. ACM Press.
[25] A. Bose and K. G. Shin. On mobile viruses exploiting messaging and bluetooth services. In *SecureComm 06*.
[26] D. Dagon, T. Martin, and T. Starner. Mobile phones as computing devices: The viruses are coming! *IEEE Pervasive Computing*, 2004.
[27] W. Enck, P. Traynor, P. McDaniel, and T. L. Porta. Exploiting open functionality in sms-capable cellular networks. In *ACM CCS '05*.
[28] C. Guo, H. J. Wang, and W. Zhu. Smart-phone attacks and defenses. In *HotNets III*, 2004.
[29] H. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *Usenix Security Symposium*, CA, 2004.
[30] C. Kreibich and J. Crowcroft. Honeycomb - creating intrusion detection signatures using honeypots. In *HotNets II*, Boston, November 2003.
[31] J. W. Mickens and B. D. Noble. Modeling epidemic spreading in mobile environments. In *ACM WiSe '05*.
[32] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. *IEEE Transactions on Information Theory*, IT-24:106–110, 1978.
[33] R. Racic, D. Ma, and H. Chen. Exploiting mms vulnerabilities to stealthily exhaust mobile phone's battery. In *SecureComm 06*.
[34] J. Su, K. K. W. Chan, A. G. Miklas, K. Po, A. Akhavan, S. Saroiu, E. de Lara, and A. Goel. A preliminary investigation of worm infections in a bluetooth environment. In *WORM '06*.
[35] G. V. Sumeet Singh, Cristian Estan and S. Savage. Automated worm fingerprinting. In *OSDI '04*, 2004.
[36] D. Tang and M. Baker. Analysis of a metropolitan-area wireless network. *Wirel. Netw.*, 2002.
[37] P. Traynor, W. Enck, P. McDaniel, and T. L. Porta. Mitigating attacks on open functionality in sms-capable cellular networks. In *ACM MobiCom '06*.
[38] C. C. Zou, W. Gong, D. Towsley, and L. Gao. The monitoring and early detection of internet worms. *IEEE/ACM Trans. Netw.*, 2005.