The Pennsylvania State University
The Graduate School

MITIGATING RAPIDLY PROPAGATING WORM THREATS IN EMERGENT NETWORKS

A Dissertation in
Computer Science and Engineering
by
Liang Xie

© 2008 Liang Xie

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

August 2008

The dissertation of Liang Xie was reviewed and approved* by the following:

Sencun Zhu
Assistant Professor of Computer Science and Engineering &
Information Sciences and Technology
Dissertation Advisor, Chair of Committee

Thomas F. La Porta
Distinguished Professor of Computer Science and Engineering

Guohong Cao
Associate Professor of Computer Science and Engineering

Zan Huang
Assistant Professor of Supply Chain and Information Systems

Raj Acharya
Professor of Computer Science and Engineering
Head of the Department of Computer Science and Engineering

*Signatures are on file in the Graduate School.

Abstract

This dissertation presents a series of techniques that help both client devices and network elements defend against a wide variety of worm attacks. These techniques can be deployed to secure emergent networks including peer-to-peer (P2P) file-sharing systems and wireless communication systems.

In recent years, worms have emerged as one of the most disastrous security threats to various information systems and network infrastructures. Although Internet worms have been extensively studied, worm issues in such emergent networks as peer-to-peer (P2P) systems and cellular networks have yet received due attention. This dissertation aims at designing *automated, realtime, and systematic* countermeasures, which leverage the existing internal communication mechanisms and network infrastructure to contain worm propagation. The proposed defenses consist of security solutions for both client and system software.

For P2P networks, this dissertation first proposes a partition-based scheme and a CDS-based scheme to contain ultra-fast topological worm spreads. These schemes leverage the underlying P2P overlay for distributing automated security patches to vulnerable machines. They are unique in adopting graph-theory techniques for containing fast spreading worms. This dissertation then proposes a P2P-tailored solution to combat file-sharing worms in P2P environments. Our solution consists of a download-based scheme and a search-based scheme. Both schemes utilize the existing file-sharing mechanisms to internally disseminate security patches to participating peers in a timely and distributed fashion.

For cell-phone networks, this dissertation proposes two device-level defenses for securing smartphone software, namely an access-control–based scheme and a GTT-based scheme. These schemes are unique in that they either enforce security policies in phone devices to identify and block worm attacks or leverage artificial intelligence (AI) methods to differentiate human or worm initiators of the phone applications. This dissertation also proposes a systematic countermeasure consisting of both terminal-level and network-level defenses for combating cell-phone worms. Unlike the existing solutions that split the collaboration between the terminal device and the network to throttle system-wide worm spreads, the proposed solution adopts an identity-based signature scheme at both the sender and the receiver side, and a detection-based automated patching scheme at the network side. Combining terminal-level and network-level defenses effectively speeds up the process of worm detection and victim disinfection.

This dissertation also provides solid mathematical analyses, extensive simulations and experiments to evaluate the effectiveness and show the applicability of the proposed defenses. In addition, it discusses some open issues related to the proposed solutions and suggests some interesting directions in combating the worm threats as the emergent networks evolve.

Table of Contents

List of Figures

List of Tables

Acknowledgments

There are many people who have contributed significantly to my work and I would like to acknowledge their contributions.

Thanks to the members of my thesis committee, Dr. Sencun Zhu, Dr. Thomas F. La Porta, Dr. Guohong Cao, and Dr. Zan Huang for their support and valuable suggestions. I would like to thank them for their comments on my thesis proposal and their help with the reviews of this dissertation. Special thanks to my advisor Dr. Sencun Zhu, for his help and suggestions during our weekly meetings. A significant portion of the ideas for this dissertation originated during our discussions there. I also appreciate his excellent guidance on doing research and writing papers. Words are not sufficient to thank all of you.

Dr. Xinwen Zhang in Samsung Information Systems American provided the OMAP-5912OSK smartphone environment in the Trust Computing Lab. Thanks to him for helping prepare the root file system and the tool chain on the OMAP board. Thanks to Dr. Trent Jaeger for giving useful comments on designing systematic countermeasures against cell-phone worms. Thanks to Dr. Hui Song (former CSE student) for discussions on the internal patching against file-sharing worms and some writing issues.

Major Portions of this work were funded by CAREER NSF-0643906 – support that is gratefully acknowledged.

Chapter 1

Introduction

1.1 Emergent Networks

This dissertation studies two different types of emergent networks: Peer-to-peer (P2P) networks and cellular networks. These networks are popular and they are playing important roles in people's daily life.

1.1.1 Peer-to-peer Networks

A peer-to-peer (P2P) computer network uses diverse connectivity between participating hosts in a network and the cumulative bandwidth of network participants rather than conventional centralized resources where a relatively low number of servers provide core services or applications to other client hosts. P2P networks such as Gnutella [2], KaZaA [3], and Chord [4] are typically used for connecting hosts via largely ad hoc connections. Such networks are useful for many purposes. Nowadays sharing content files containing audio, video, data or anything in digital format is very common in P2P systems. Realtime data, such as telephony traffic, is also passed using P2P technology [5].

A pure P2P network does not have the notion of clients or servers, but only equal peer nodes that simultaneously function as both "clients" and "servers" to the other nodes on the network. This model of network arrangement differs from the client-server model (e.g., the FTP service) where communication is usually to and from a central server. An important goal in P2P networks is that all clients provide resources, including bandwidth, storage space, and computing power. Thus, as nodes arrive and demands on the system increase, the total capacity of the system also increases. This is not true of a client-server architecture with a fixed set of servers, in which adding more clients could mean slower data transfer for all users. The distributed nature of P2P networks also increases robustness in case of failures by replicating data over multiple peers, and – in pure P2P systems – by enabling peers to find the data without relying on a centralized index server. In the latter case, there is no single point of failure in the system.

The P2P overlay network consists of all the participating peers as network nodes. There are links between any two nodes that know each other, i.e., if a participating peer knows the location of another peer in the P2P network, then there is a directed edge from the former node to the latter in the overlay network. Based on how the nodes in the overlay network are linked to each other, we can classify the P2P networks as *unstructured* or *structured*. In unstructured systems, file placement is random and has no correlation with the topology; in structured systems, placement of shared data and the topology characteristics of the network are tightly bound based on *distributed hash tables* (DHTs).

An unstructured P2P network is formed when the overlay links are established arbitrarily. Such networks can be easily constructed as a new peer that wants to join the network can copy existing links of another node and then form its own links over time. In an unstructured P2P network, if a peer wants to find a desired piece of data in the network, the query has to be flooded through the network to find as many peers as possible that share the data. The main disadvantage with such networks is that the queries may not always be resolved. Popular content is likely to be available at several peers and any peer searching for it is likely to find the same thing. But if a peer is looking for rare data shared by only a few other peers, then it is highly unlikely that search will be successful. Since there is no correlation between a peer and the content managed by it, there is no guarantee that flooding will find a peer that has the desired data. Flooding also causes a high amount of signaling traffic in the network and hence such networks typically have very poor search efficiency. Most of the popular P2P networks such as Gnutella [2], KaZaA [3], and BitTorrent [6] are unstructured.

Structured P2P network employs a globally consistent protocol to ensure that any node can efficiently route a search to some peer that has the desired file, even if the file is extremely rare. Such a guarantee necessitates a more structured pattern of overlay links. By far the most common type of structured P2P network is the distributed hash table (DHT), in which a variant of consistent hashing is used to assign ownership of each file to a particular peer, in a way analogous to a traditional hash table's assignment of each key to a particular array slot. Some well known DHTs are Chord [4], Pastry [7], Tapestry [8], and CAN [9].

1.1.2 Cellular Networks

A cellular network is a radio network made up of a number of radio cells (or just cells) each served by a fixed transmitter, known as a cell site or base station. These cells are used to cover different areas in order to provide radio coverage over a wider area than the area of one cell. Cellular networks are inherently asymmetric with a set of fixed main transceivers each serving a cell and a set of distributed transceivers (generally mobile phones) which provide service to the network's users. Cellular networks offer a number of advantages over other wireless communication systems. These advantages include increased user capacity, reduced power consumption, and better radio coverage.

The primary requirement for a cellular network is to develop a standardized method for each mobile phone to distinguish the signal emanating from its own transmitter (serving cell) from the signals received from other transmitters (neighboring cell). Currently there are two major solutions: frequency division multiple access (FDMA) [10] and code division multiple access (CDMA) [11]. FDMA works by using varying frequencies for each neighboring cell. By tuning to the frequency of a chosen cell mobile phones can avoid the signal from other cells. The principle of CDMA is more complex, but achieves the same result; mobile phones can select one cell and listen to it. In CDMA, multiple phones share a specific radio channel; the signals are separated by using a pseudo-noise code (PN code) specific to each phone. Another available method of multiplexing, time division multiple access (TDMA), is used in combination with either FDMA or CDMA in a number of systems to give multiple channels within the coverage area of a single cell.

Most existing cellular radio systems are still in their second generations (2G and 2.5G). For example, Global System for Mobile Communications (GSM) [12] focuses on circuit-based voice service and General Packet Radio Service (GPRS) [13] further provides low-speed (up to 150∼170 kbps downstream) packet-data service to users. Nowadays the fast-growing third generation cellular systems (3G systems), for example, Wideband Code Division Multiple Access (W-CDMA) networks, support higher data transmission speed (up to 8∼10 Mbps downstream) and larger user capacity. In addition, they allow different systems to inter-operate in order to attain global roaming across different networks [14]. The International Telecommunication Union (ITU) has a candidate for the international 3G standard known as IMT-2000; European Community [14] defines its version of 3G standard named UMTS (universal mobile telecommunication system).

As cellular networks grows rapidly, wideband packet-data services such as web browsing, image and video transmission, and Multimedia Messaging Service (MMS) have been quickly provided to mobile users with similar quality as in wired networks. This fact also contributes to the fast development of mobile phones, especially the quick surge of *smartphones* in the terminal market. Most smartphones today use an identifiable embedded operating system (e.g., Symbian [15], WinCE [16], and Linux), often with the ability to add new applications (e.g., for enhanced data processing, connectivity or entertainment) - in contrast to regular phones which only support very limited applications based on voice or text. Smartphone applications may be developed by the phone manufacturer, by the network operator or by third-party software developers.

In addition to the voice and packet-data transmission through the cellular network infrastructure, most existing smartphones support Bluetooth and even Wi-Fi communications. These capabilities enrich a smartphone's connectivity with the outside world. In terms of functionalities, most smartphones today support full-featured voice call, email, and messaging capabilities. Other additional functionalities might include software for contact management, navigation, media software for playing music and video clips, internet browsers, and additional hardware such as a miniature QWERTY keyboard, a touch screen, or a built-in camera, etc.

1.2 Security Problems

A worm is malicious code which can propagate from an infected system to other systems in an automated way. The highly automated nature of worms combined with software mono-cultures of the Internet and the uncontrolled Internet communication model have enabled a large number of systems in the Internet to be compromised within a matter of hours or even seconds [17]. In the last several years, worms have emerged as one of the leading threats to our information systems and critical infrastructure. Besides causing damages to infected hosts, worms are increasingly being used as the vehicles for installing remote-controlled zombies and botnets, which are responsible for large-scale network attacks such as Distributed Denial-of-Service (DDoS) attacks. Indeed, worm propagation itself effectively creates a denial of service in many parts of the Internet because of the huge amounts of scan traffic generated). Disruption of services caused by worm attacks could have catastrophic effects, including huge financial losses, disruption of essential services, and even loss of human life. For instance, the outbreak of the CodeRed worm infected more than 359,000 hosts, causing financial losses of approximately 2.6 billion dollars [18]. Because of the number of organizations and users on the Internet and their increasing dependency on the Internet to carry out day-to-day business, effectively containing worm propagation are of paramount importance in preventing the happening of catastrophic events with impacts on our safety, security, economy and society.

Recently, tremendous research effort has been taken to combat Internet worms. For example, there are techniques using such macro symptoms as Internet background radiation (observed by network telescopes) to raise early warnings of Internet-wide worm infection [19], techniques using such local traffic symptoms as content invariance, content prevalence and address dispersion to generate worm signatures and/or block worms (e.g., Earlybird [20], Autograph [21], Ploygraph [22]), and TRW [23], techniques using worm code running symptoms to detect worms (e.g., Shield [24], Vigilante [25], COVERS [26]), and techniques use anomaly detection on packet payload to detect worms and generate signature (e.g., [27, 28, 29]).

Despite the great number of approaches that have been proposed, our war against worms does not seem to end in the near future. New computer and system vulnerabilities are continuously reported, and new worm attacks still keep succeeding [30, 31]. Also we observe another significant trend in worm attacks: *the number of worm attacks against emergent networks is rapidly growing.* For example, with the increasing popularity of applications based on peer-to-peer networking, P2P networks have provided a great opportunity for worm propagation. A P2P worm may take advantage of the topological information of the participating hosts to expedite its propagation [32, 1], or disguise itself as an attractive file to get spread to many users in the network (e.g., the Benjamin.a worm, the Realmony worm, Duload worm, Togod worm, etc.). In cellular networks, various worms have been reported exploiting vulnerabilities in mobile phone software to propagate through MMS (Multimedia Messaging Services) or Bluetooth interface [33, 34, 35]. Because of the unique communication models and resource- constraints of the emergent networks, most of the existing solutions for defending against Internet worms are not directly applicable to these networks. Hence, there is an immediate demand on designing new techniques to effectively contain worm propagation in these emergent networks. This is the objective of the research in this dissertation.

1.3 Summary of Contributions

This dissertation focuses on two major types of worms that are threatening the emergent networks around people's daily life. Specifically, we study *P2P worms* in P2P networks and *cell-phone worms* in mobile communication networks. Our goal is to first analyze and model worm propagation characteristics in different network environments, and then to devise automated and systematic countermeasures. Our countermeasures will leverage the existing network infrastructures and internal communication mechanisms to actively and quickly launch protection against the worm attacks. Specifically, we propose the following solutions to achieve the above design goals.

For P2P networks,

- we first study *topological worms*, which exploit host software vulnerabilities and network topology information to propagate in an ultra-fast way. Faced with the challenge of designing an automated and systematic defense against these ultra-fast spreading worms, we study the feasibility of leveraging the existing P2P overlay infrastructure for distributing automated security patches to those vulnerable machines. We adopt graph theory in designing strategies for containing topological worms: a *partition-based* scheme and a Connected Dominating Set (CDS)-based scheme.

- We then study *file-sharing worms*, which are malware spreading via users' file-sharing activities in P2P systems. We propose a P2P-tailored patching system, which can be used as a complement of the existing automated patching tools such as Microsoft Windows Update and Symantec Security Update. We study the feasibility of utilizing the existing file-sharing infrastructure including the file download process and the file search process to internally disseminate security patches to participating peers in a systematic and automated way. We devise two corresponding distributed mechanisms for containing file-sharing worms in a wide variety of P2P systems.

For cell-phone networks,

- We first study the attack and propagation behavior of cell-phone worms and categorize them into different classes. Based on real implementations of two major forms of attacks on smarphone devices, we first seek *device-level* solutions to combat the security threats. Specifically, we design an access-control–based mechanism in smartphone to defend against worms which launch attacks through running malicious processes; considering such defense alone is not capable of blocking elaborated worms which hijack existing phone applications (e.g., messaging) to execute their malicious codes within a legitimate security domain, we further adopt artificial intelligence (AI) techniques such as Graphic Turing test (GTT) to provide more secure and comprehensive protection on those vulnerable smartphones.

- We then propose a systematic countermeasure involving defenses at both *terminal* and *network* levels to combat cell-phone worms. Our terminal-level defense has two components: *sender-side defense* and *message authentication.* The sender-side defense leverages device-level protection such as the GTT mechanism to identify and block worm attacks within smartphones; the message authentication consists of an identity-based signature scheme which helps both sender and receiver sides prevent unauthorized messages from leaving compromised phones and entering normal phones. At the network level, we propose a push-based automated patching mechanism for cleansing compromised phones once they have been identified. Our systematic countermeasure achieves *real-time*, *self-healing*, and *automated* defense against cell-phone worms.

1.4 Organization

The rest part of this dissertation is organized as follows: In Chapter 2, we review the preliminaries of the proposed work. In Chapter 3, we propose a partition-based scheme and a CDS-based scheme to combat ultra-fast topological worm propagation in P2P networks. In Chapter 4, we propose a download-based approach and a search-based approach to contain file-sharing worms in P2P systems. In Chapter 5, we propose device-level defenses including an access-control–based and a GTT-based defense to combat cell-phone worms within smartphone devices. In Chapter 6, we propose a two-level defense framework for cellular networks. Our solution is featured by an identity-based signature scheme at both the sender and the receiver side, and a detection-based automatic patching scheme at the network level. We summarize the proposed countermeasures and discuss some interesting open issues in Chapter 7.

Chapter 2

Preliminaries

2.1 Peer-to-Peer Systems

2.1.1 Network Topology

Modern P2P systems such as Gnutella [2] and KaZaA [3] typically adopt a *two-tier* overlay [36] in which a subset of peers, called *supernodes* or *ultra peers*, form a top-level overlay while other participating peers, called *leaf peers*, are connected to the top-level overlay through one or multiple supernodes. A supernode maintains a directory of files stored at its leaf nodes. When a leaf node queries a file, it sends the request to its supernode. If the latter knows the file location (i.e., one of its leaf nodes stores the file), it replies the requester directly. Otherwise, it floods the query to other supernodes.

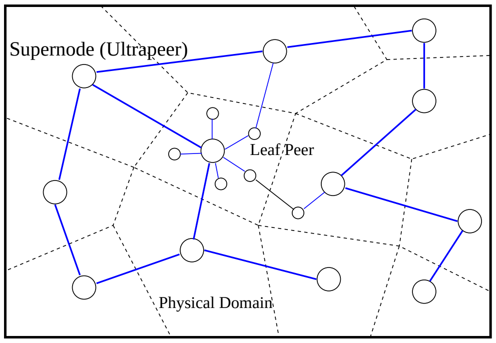


Figure 2.1. A network graph of modern P2P systems

In our work, we generalize a P2P overlay topology as an undirected random graph, in which each vertex corresponds to a peer and each edge reflects the current neighboring relationship between two peers. If the overlay adopts the two-tier hierarchy, it usually follows *power law*. A P2P graph is dynamic because it updates when peers join or leave the system. However, it is not influenced by the P2P traffic, and it remains relatively static during the very short time period of topological worm spread. We note that the P2P graph is a logical concept. Physically, peers connect to an Internet routing infrastructure whose key part consists of hundreds of thousands of routers. Therefore, each edge in the topology graph has a latency, which is largely determined by the hop-count between the two end hosts.

2.1.2 File-sharing Applications

Many P2P file-sharing systems are available these days (a nice comparison can be found in [37]), very popular ones including eMule, KaZaA, Gnutella, and BitTorrent. For concreteness, however, our discussion will focus on the unstructured networks such as Gnutella and KaZaA.

We describe the file-sharing process using Gnutella as an example. Each node uses a *shared folder* to store the files it wishes to share. When a node initiates a download request for a specific file, it places a search for the target node(s) responsible for the given file ID. The search request is routed through a 2-tiered system of ultrapeers and leaves. In response, the user collects a list of peers (*search list*), each of which contains a file copy. The user then connects to a target node in the list and downloads the copy. Having successfully acquired the file and stored it in the local machine, she usually opens it for use.

2.2 Messaging in Cellular Networks

2.2.1 MMS Communications

Multimedia Message Service (MMS) [38] evolves from Short Message Service (SMS), a text-only messaging technology in mobile networks. With MMS, a cell phone can send and receive multimedia messages containing graphics, video clips, application software, etc.. MMS is designed to work in mobile packet data networks such as GPRS and UMTS systems. Fig.2.2 shows an abstract view of the MMS network. A MMS Center (MMSC) typically contains an MMS proxy-relay and an MMS server. The former is responsible for message routing between MMSCs and the latter provides message storage and retrieval. A typical MMS data flow starts when a subscriber uses a smart-phone to compose, address and send an MMS message to another subscriber. The initial submission by an MMS client to the *home MMSC* is accomplished using HTTP with specialized commands and encodings. Upon receiving an MMS message, the *recipient MMSC* notifies the receiver using either a SMS notification, HTTP Push or WAP Push.

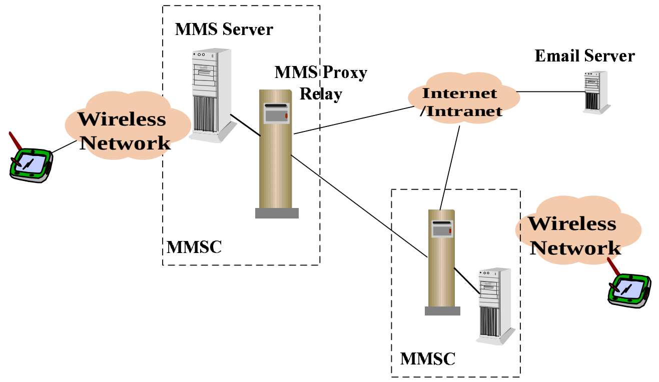


Figure 2.2. An abstract view of cellular network supporting Multi-media Messaging Service (MMS).

Two delivery modes are available in a recipient MMSC: *immediate* or *deferred*. In the immediate mode, when the receiver gets the notification, it immediately retrieves the message content from the MMSC; in the deferred mode, the receiver is alerted and allowed to choose whether and when to retrieve the new message. The former method hides the network latencies from the user but is less secure because of its instant retrieval. Our work is based on the latter mode due to its popularity in cellular networks.

2.2.2 Blue-tooth Communications

Unlike the MMS which utilizes the network infrastructure to deliver messages among users, the Blue-tooth technique helps cell phones set up localized wireless connections for message dissemination. Specifically, two Blue-tooth–enabled cell phones that are in close proximity with each other can set up a secure communication channel through pairing (a symmetric key authentication process). Through this secure channel, cell phones may exchange data (e.g., video clips, applications) with a throughput of 700K∼2.1M bps. Typically, Blue-tooth uses 2.4 GHz radio waves and has a coverage of 10 meters (Class II) or 100 meters (Class I).

2.2.3 Identity-based Signature

In 1984, Shamir [39] proposed an extra twist on a public key cryptosystem: in stead of generating a random pair of public/secret keys and publishing one of these keys, a use may choose her identity information as her public key. This enables any pair of users to verify each other's signature without exchanging public keys, hence reducing the message overhead in a communication system. Since Boneh et al. [40] provided the first practical IBE system based on *bilinear paring* in 2001, several IBS variations [41, 42] have been proposed. Typically, an IBS scheme consists of four algorithms:

- **Setup**(k): This algorithm is executed by a *Private Key Generator* (PKG), which takes a random parameter k and generates a master key K_m and a set of public parameters *params* (typically include the PKG's public key, some known functions and groups of numbers).

- **Keygen**($ID_i, K_m, params$): Based on a user i's identity ID_i, the PKG uses K_m and *params* to compute for user i a string Q_{ID_i} and the corresponding private key d_{ID_i}.

- **Sign**($M, params, d_{ID_i}$): This algorithm is run by user i to sign a message M, the user takes *params* and the private key d_{ID_i} to compute a signature σ for M.

- **Verify**($M, \sigma, ID_i, params$): A user j runs this algorithm to verify a message M and a signature σ sent from user i. j first derives Q_{ID_i} from ID_i and *params*, it then performs a test $VERIFY(M, \sigma, Q_{ID_i}, params)$. j accepts the signature if the result is $TRUE$ and rejects otherwise.

Chapter 3

Combating Ultra-fast Topological Worms in P2P Networks

3.1 Introduction

Recent Internet worms outbreaks indicates that hundreds of thousands of Internet servers and client machines can be infected within a few minutes. Worm threats also become imminent and devastating to distributed P2P systems, which are featured by huge population and frequent data accesses. P2P worms may first compromise client machines, and then propagate through such strategies as scanning IP addresses [43], harvesting email addresses, discovering P2P neighbors from those victims, or carried in shared files.

Internet worms adopt three major scanning strategies in identifying new victim targets: *random, hit-list–based, and topological* scanning. A random scanning worm selects targets' IP addresses at random. It needs to probe if a host with the specific IP address really exist; a hit-list–based scanning worm collects a preference list of susceptible nodes (based on node importance or the overlay topology) and choose new targets from the list; a topological scanning worm exploits topology information from infected hosts and accurately locate new targets. Topological scanning worms typically combine the hit-list–based and the topological scanning strategy to propagate. Specifically, a worm starts by attacking initial targets chosen from a hit-list, it then scans the neighbor sets or the routing tables of these victims and identifies all susceptible neighbors as new targets. In this way, the worm continues its spread cycle and eventually spread to the entire overlay. Note that for the tractability of the problem, we assume the size of a hit-list is up to 1.0% of the population. For example, a worm may initiate a file search and obtain a list of nodes, to which it launches attacks. To prevent an attacker from building a huge hit-list, we assume some mechanism has been (or will be) deployed to reject unauthorized crawling of the P2P topology.

Our focus here is on the *topological scanning worm* (or topological worm in short), which exploits neighborhood information from the overlay to locate new targets for the system-wide spread. Compared with an IP scanning worm that randomly probe IP addresses to discover new targets, a topological worm does not need interactions, thus are less likely to be detected by an IDS and will be more accurate in target-seeking and faster in spread. These facts make it more difficult to containment topological worm propagation in P2P networks. Although no instance of a topological worm has been witnessed in a real P2P network, there are strong evidences that such worms could happen. For example, worms have been reported in [44, 45] to exploit the buffer overflow vulnerability in FastTrack, KaZaA, iMesh and other P2P client programs to launch denial-of-service attacks on supernodes and potentially other machines.

Recently some attention has been drawn to addressing security problems in P2P networks. However, The most deadly and imminent topological worm threats in P2P systems somehow have largely been ignored. Vojnovic et al. [46] demonstrated that effective automatic patching is feasible for an overlay network if combined with mechanisms to bound worm scan rate and with careful engineering of the patching system. Their work reveals a race between the worm spread and the patch dissemination. Zhou et al. [1] proposed an end-host based defense infrastructure for containing topological worms. Their scheme adopts the self-certifying alerts [32] and their preliminary results have shown effectiveness in protecting various overlays. Cai et al [47] suggests a collaborative way of worm containment in a DHT-based overlay. This approach automatically generates worm signatures by analyzing payload contents and address dispersions to identify alerting traffic. Considering the fact that users often delay or ignore installing security patches due to complications of application/system restarts, Altegar et al. [48] designed a dynamic patching tool which applies fixes to C programs at run time.

In our research, we address a challenging question: how can we contain the ultra-fast topological worms when they start to surge in a real network? Given that today's Internet worms such as Slammer may infect millions of machines in minutes and topological worms could propagate even faster, human response is clearly too slow. In other words, we must rely on a *systematic* and *automatic* worm containment system. The research challenge is: Is this type of automatic containment system at all feasible, considering such a system must provide (1) automatic worm detection, (2) automatic patch generation, and (3) automatic patch dissemination, verification and application? The first two issues are being actively studied [24], and here we assume they will be completely addressed someday soon. Through the appropriate choice of the number of initially infected nodes, we may simulate the delay caused by the first two steps. We will focus on addressing the third problem: how can we disseminate the security patches to the vulnerable hosts before the worm reaches them? We note that although automatic update mechanisms such as Windows Automatic Updates and Symantec security updates have been in place for directly acquiring the latest security patches from vendor sites, as shown in [46], the time to disseminate patches to hundreds of millions of Windows PCs would be of the order of hours, which is much longer than what is needed for worm spread. Thus, a faster way of patch dissemination, at least comparable to the speed of worm spread, is needed. Clearly, nothing can we do if the worm has already finished its spreading, and nothing should be further done if all the hosts have received and applied the proper patch. As such, we are interested in a quantitative study of the speeds of worm and patch when they are in an arms race.

While the solution space could be huge and better solutions could exist, in this work as an initial study of this important and challenging problem, we study the feasibility of constructing a defense infrastructure within a P2P system for rapid patch dissemination and examine its effectiveness. We observe that the order of patch dissemination among hosts plays a critical role in combating worm propagation. Network-wide flooding is not necessarily the fastest and most effective way. On the other hand, security patches do not have to reach all the hosts in such a short racing period if the worm could be first quarantined in a small island. Based on these observations, we examine the effectiveness of two types of strategies. First, we propose to proactively select a set of worm-immune nodes in the overlay to block the possible worm spreads. Second, by exploiting the P2P network topology information, we may first disseminate the security patch to a select set of nodes, which will further flood it to local neighbors. Specifically, we propose a *partition-based scheme* in which immune nodes are chosen in a way that they partition the overlay graph into many nearly-balanced sub-graphs and each blocks the worm spread within its area. We also propose a *CDS-based scheme* in which a dominating set of nodes are chosen from the overlay and these nodes are utilized to disseminate security alerts in a timely fashion.

3.1.1 System Model

Network Model There are two categories of P2P systems: *unstructured*, and *structured*. In unstructured systems, such as Gnutella and KaZaA, file placement is random and has no correlation with the topology; in structured systems, such as Chord and Pastry, placement of shared data and the topology characteristics of the network are tightly bound based on distributed hash tables (DHTs). Our main focus in this dissertation is the unstructured P2P systems.

Modern unstructured systems such as Gnutella typically adopt a *two-tier* overlay [36] in which a subset of peers, called *supernodes* or *ultra peers*, form a top-level overlay while other participating peers, called *leaf peers*, are connected to the top-level overlay through one or multiple supernodes. A supernode maintains a directory of files stored at its leaf nodes. When a leaf node queries a file, it sends the request to its supernode. If the latter knows the file location (i.e., one of its leaf nodes has the file), it replies the requester directly. Otherwise, it floods the query to other supernodes.

In our work, we generalize a P2P overlay topology as an undirected random graph, in which each vertex corresponds to a peer and each edge reflects the current neighboring relationship between two peers. A P2P graph is dynamic because it updates when peers join or leave the system. However, it is not influenced by the P2P traffic, and it remains relatively static during the very short time period of topological worm spread. We note that the P2P graph is a logical concept.

Physically, peers connect to an Internet routing infrastructure whose key part consists of hundreds of thousands of routers. Therefore, each edge in the topology graph has a latency, which is largely determined by the hop-count between the two end hosts.

Node States We define three security states for P2P hosts: *susceptible*, *infected* or *immune*. A susceptible host is not protected against the worm. It either gets infected when exposed to the worm attack or becomes immune when a protection is in place *before* the worm arrives. By protection we mean the host installs a security patch and activates it. We note that an infected host may get recovered when, for example, the user activates a patch and scans the machine to cleanse the infection. However, such recovery is usually too late because the worm may have caused serious damages (e.g., system crash) to the victim. We should avoid infections on the hosts.

Attack Model Internet worms adopt three major scanning strategies in identifying new victim targets: *random, hit-list–based, and topological* scanning. A random scanning worm selects targets' IP addresses at random. It probes if a host with the specific IP address really exists; a hit-list–based scanning worm collects a preference list of susceptible nodes (based on node importance or the overlay topology) and choose new targets from the list; a topological scanning worm exploits topology information from infected hosts and accurately locate new targets. Topological worms typically combine the hit-list–based and the topological scanning strategies. Specifically, a worm starts by attacking initial targets selected from a hit-list (acquired by simply initiating a file search), it then scans the neighbor sets or the routing tables of these victims and identifies all susceptible neighbors as new targets. In this way, the worm continues its spread cycle and eventually spread to the entire overlay. Note that for the tractability of the problem, we assume the size of a hit-list is up to 1.0% of the population. To prevent an attacker from building a huge hit-list, we assume some mechanism has been (or will be) deployed to reject unauthorized crawling of the P2P topology.

3.2 System Overview

As described earlier, we study the topological worms for which the patches to address the security vulnerabilities have been generated, for example, by the security vendors such as Microsoft and Symantec. We assume for security purpose, such vendors may deploy a group of *security servers* in P2P systems, which form a separate overlay. These servers are publicly known. They are well maintained by the vendors so that they keep the up-to-date worm definitions and the security patches. In addition, these servers may collaborate and share security information with each other. The number of security servers is typically determined by the size of the P2P system. As P2P networks are distributed, security servers can be deployed in various locations, each taking care of a certain proportion of the network. In our schemes, these servers also secretely construct periodic snapshots of the P2P overlay and utilize this information to help secure other hosts. On the other hand, these servers do not participate the file-sharing process, so they do not have to be very powerful.

We take two defense principles. The first is to proactively select a set of worm-immune nodes in the overlay to block the worm spread; the second is to compete with the worm so that susceptible nodes may be alerted and immunized before the worm attacks reach them. Along the first design principle, we propose a partition-based scheme, in which some worm-immune nodes (key nodes) are systematically chosen by the security servers in a way that they partition the overlay graph into as many nearly-balanced sub-graphs as possible. Thus, worm propagation can be effectively contained within these sub-graphs. Along the second design principle, we propose a CDS-based scheme, in which the security servers select a small proportion of hosts to form a *connected dominating set* for the overlay. Once a topological worm has been detected, the servers disseminate security alerts through this set to notify vulnerable hosts of the surging worm attack and urge them to launch the protection.

Previously, Zhou et al. [1] introduced a random patch dissemination scheme, which falls in the second category of the design principles. It assumes each node in the system has an independent probability p of being a guardian node. When a worm starts propagating and reaches a guardian, the guardian immediately detects the worm and notifies other nodes by disseminating a self-certifying alert (SCA) [32] in a flooding mode. Since this scheme does not make use of any topological information, it is much less effective than ours in containing topological worms. We will use this scheme as the base-line approach and show the comparisons with our schemes.

3.3 Partition-based Scheme

Basic Principle In this approach, security servers periodically collect and assemble the latest snapshot of the overlay from those supernodes. According to this topological information, security servers construct a small group of worm-immune hosts (named *key nodes* hereafter) which protect the susceptible hosts by stopping the worm spread within the overlay graph.

The issue here is how to decide such a small set of key nodes from the overlay topology. If security servers can predict the worm hit-list, the choice of key nodes may be biased towards the neighbors of those initial victims. However, in real applications the worm's hit-list varies with link latency, node degree and host vulnerability. We propose a heuristic for choosing key nodes: *Key nodes should partition the overlay graph into as many separate pieces as possible and block worm propagation within each partition.* A partition contains non-zero number of nodes and (ideally) worm propagation from an infected node inside its partition to the rest of the region has to go through a key node, which blocks the propagation. Thus, an occurrence of the worm attack will be confined within a partition, leaving other partitions of the overlay intact. This approach is optimal if we assume that servers have no knowledge on the origins of the worm attacks. Worm containment now becomes a graph problem and we may utilize graph-theory techniques to solve it.
Acquiring Overlay Topology In the partition-based scheme, security servers adopt a scalable *parallel crawling* technique [36, 49] featured by a master-slave architecture, to collect topology information in a P2P system. Specifically, a master process running in the security server coordinates multiple slave processes in a selected list of supernodes that crawl disjoint proportions of the network in parallel. Each security server is responsible for managing the list of supernodes and constructing the final topology graph. Each supernode in the list receives some

initial points from the security server and start discovering the network topology around these points. We note that a distributed crawler (e.g., Cruiser [36]) is able to accurately capture a complete snapshot of a Gnutella network with more than one million hosts in just a few minutes. Moreover, the overlay graph is relatively static during the short period of worm spread and the security servers do not have to frequently reconstruct the overlay graph. We note that the crawling interval is determined by the overlay dynamics, which reflect the frequency of node joins and departures. We will evaluate the impact from overlay dynamics in our experiments.

Partitioning an Overlay Graph Once the servers have constructed the snapshot of the overlay graph, they may partition the graph and choose key nodes. We start with a simple example. Suppose we have a undirected graph $G = (V, E)$, where V is a finite set of vertices and E is a finite set of edges. We partition V into two sub-sets, A and B. As a result we get a sub-set C which consists of edges spanning A and B and incident vertices of these edges. We define edges in C as *cut edges* and the vertex set $\{1,2\}$ as the *vertex separator* of G. Clearly, deletion of the vertex separator will disconnect A and B. Our goal is to obtain a minimal vertex separator while maintaining an appropriate balance (e.g., the same number of vertices) between A and B. To achieve this goal, we first need to obtain a minimal set of cut edges, based on which we may further derive the minimized vertex separator.

The problem now becomes to partition the vertices of an overlay graph G into k subsets such that each of them has a nearly equal size of vertices, while the number of cut edges spanning the subsets is minimized (see Fig.3.1). We define it as a *relaxed k-way graph partitioning problem* as follows.

Definition 1 (Relaxed K-way Graph Partitioning) *Given a graph $G = (V, E)$ with $|V| = n$, partition V into k subsets V_1, V_2, ..., V_k such that $V_i \cap V_j = 0$ for $i \neq j$, $|V_i| \approx n/k$, $\cup_i V_i = V$, and the number of cut edges is minimized.*

Compared with [50], we relax the strict balance requirement, so that vertex numbers in subsets do not have to be exactly equal. This allows security servers to provide an approximated overlay graph as the input. Besides, this produces a nice feature that we may always derive a good vertex separator from a minimized set of cut edges [51] (we show an efficient algorithm to achieve this later). Finding an exact minimized set of cut edges for a given graph is NP-hard. Fortu-

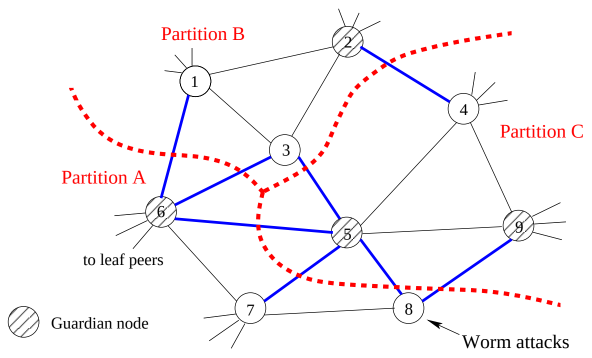


Figure 3.1. Partitioning the overlay graph to contain worm spread (within partition A)

nately, we may use some approximations because we do not require partitions to be precisely balanced. A popular technique in practice (especially when graph G is complex) is the *multilevel k-way partitioning* [50] approach. The basic structure of this approach is as follows. The graph $G = (V, E)$ is first coarsened down to a small number of vertices and a k-way partitioning of this small graph is computed, typically using the Kernighan-Lin algorithm [52] in a recursive way (i.e., divide and conquer), and then this partition is projected back towards the original graph (finer graph), by successively refining the partitioning at each intermediate level.

Now we may apply the partition-based algorithm in a P2P system. We use an undirected graph $G_O = (V_O, E_O)$ to denote the overlay graph, where V_O denotes the vertex set and E_O denotes the edge set (Section 2.1). The proposed algorithm is executed periodically in the security servers and consists of the following four steps:

1. Security servers collect the overlay topology and map it to a graph G_O.

2. Security servers execute the multi-level k-way partitioning algorithm and obtain the cut edges of G_O.

3. Security servers run Algorithm 1 to compute a minimum vertex separator from the set of cut edges.

4. Security servers choose members from the vertex separator as the key nodes and push the security alerts to them.

Algorithm 1 Minimum Vertex Separator Algorithm

```
Input: $E_c = \{e_1, e_2, ..., e_m\} \in E_O$: a set of cut edges;
Procedure:
1: $V_s \leftarrow \phi$
2: while $E_c \neq \phi$ do
3:     select $v \in V_O$ which is shared by the most number of cut edges in $E_c$
4:     add $v$ to $V_s$
5:     remove from $E_c$ any cut edge whose end point is $v$
6: end while
Output: $V_s = \{v_1, v_2, ..., v_n\}$: the vertex separator of $G_O$;
```

We propose Algorithm 1 for further deriving a minimum vertex separator from the cut edges (obtained from step 2). Using the overlay graph in Fig.3.1 as an example, a typical sequence of adding nodes to the vertex separator is: node $5 \rightarrow 6 \rightarrow 2, 9$. Accordingly, the sequence of deleting cut edges (in bold line) between the partitions is: edge $(5,3),(5,6),(5,7),(5,8) \rightarrow (6,1),(6,3) \rightarrow (9,8),(4,2)$.

Protecting Key Nodes In the partition-based scheme, security servers require a key node to launch the protection once it has received the security alert. Each alert typically contains the specific worm information, a vendor patch and the vendor's signature for verification. A key node is aware of its importance in the overlay and is urged to launch immediate protection to protect itself and others. However, in some special cases when a selected node refuses to be a key node or ignores the patch, several unseparated partitions could be merged into a larger one and a worm will eventually escape from a single partition. To address this problem, we propose that security servers pre-define a threshold, say S_t, as an upper-limit of node population in each partition. Based on the responses from those key node candidates, security servers leverage the graph knowledge to investigate the population s in each merged partition. Once they find s exceeds S_t, the servers will repartition this unbalanced piece and set up new key nodes to launch the protection.

We notice that the set of key nodes may overlap with those supernodes in the overlay. However, we do not simply choose the latter to partition the overlay. Sometimes there exist far more supernodes than necessary key nodes. On the other hand, the overlay evolves with time and some non-supernodes may become key nodes. Moreover, some P2P systems [53] make supernode selections very dynamic and adaptive to reduce the flooding cost of queries.

3.4 A CDS-based Scheme

Basic Principle In this section, we consider to flood the security patch to all the hosts in a race with the worm. Clearly, if we inject the patch to one host, which then propagates the patch to its neighbors, which recursively in turn forward to their own neighbors, just as a worm does, it will not be effective because the worm started earlier. Randomly selecting a small set of key nodes to start the propagation will be better, however, it is still not optimal because the distance (in terms of number of hops) of a node from a key node is not bounded–some nodes may wait long to receive the patch. To achieve the best performance, we must exploit the P2P network topology information to start the dissemination process, so that the patch latency is more guaranteed.

We propose that the security servers compute a small group of nodes named *dominating set* each time when the overlay graph has been constructed. A group of nodes is a dominating set if every node not in the subset is adjacent to at least one node in the subset. Also, security servers prefer nodes in this subset to be *connected* for the ease and reliability of alert delivery. These selected hosts are called *CDS nodes* hereafter. The CDS nodes are *only* derived by the servers, which securely construct and maintain the topology graph. We note that the P2P overlay usually has rich node connectivity, hence the set of CDS is relatively small. An example in Fig.3.2 shows that, when the security server pushes a patch into the CDS nodes $\{1, 2\}$, everyone receives it within 1-2 hops from the server.

Finding CDS Nodes in the Overlay We show how security servers derive the set of nodes to which they directly push the security alert. We first define an *Overlay CDS* problem, which aims at finding a feasible set of CDS nodes based on an overlay graph. We give the following definitions.

Definition 2 (Dominating Set) *Given an overlay graph $G_O = (V_O, E_O)$, a dominating set D of G_O is a subset of V_O such that for every node $v \in V_O$, either $v \in D$ or there exists a node $u \in D$ such that $(u, v) \in E_O$.*

Definition 3 (Overlay CDS problem) *The Overlay CDS problem is to find a*

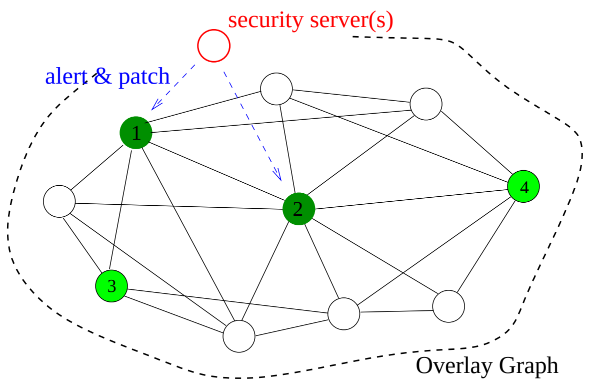


Figure 3.2. CDS node 1, 2 are found in an well-connected overlay. Node 3, 4 form a weakly connected dominating set.

minimal subset S of nodes, such that the subgraph induced by S is connected and S forms a dominating set in the overlay graph G_O.

The CDS problem was originally defined in [54], and it has already been shown that finding the CDS in a general graph is a NP-complete problem [55]. However, a good approximation can always be found for an overlay graph G_O, which typically has high connectivity among the supernodes. We adopt the following two-phase algorithm from Guha and Khuller [54] to derive the CDS nodes:

1. Initially all nodes are colored *white*. Each time we include a vertex in the dominating set, we color it *black*. Dominated nodes are colored *gray* (once adjacent to a black node). In the first phase, we pick a node at each step and color it black, coloring all adjacent white nodes gray. A *piece* is defined as a white node or a black connected component. At each step we pick a node to color black that gives the maximum (non-zero) reduction in the number of pieces.

2. In the second phase, we have a collection of black connected components. We recursively connect pairs of black components by choosing a chain of vertices, until there is only one black connected component. The final solution is the set of black vertices that form the connected component.

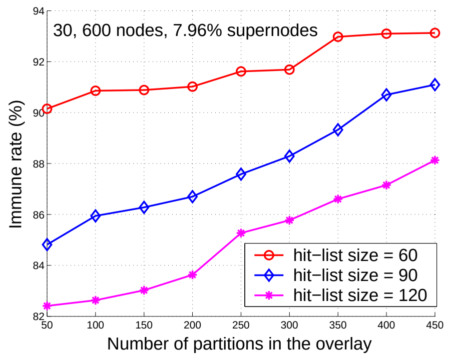
We further adapt the solution to the P2P environment. Due to the complexity and the randomness of an overlay graph, sometimes the size of the resulting CDS nodes could be unreasonably large. Considering the bandwidth limit, it is not scalable for the servers to simultaneously push the security alert to a full set of CDS nodes. Moreover, such a message may contain multiple patch payloads, which typically happens when the server fails to infer a specific patch from the worm notification it receives. To balance the trade-off between scalability and security, we propose security servers push the alert to a random subset of the CDS nodes. We note that CDS nodes are well connected in the P2P overlay. Therefore, our strategy only slightly increases the receiving latency (i.e., nodes receive the alert through a few more hops). Currently we do not consider alternatives such as the *k-dominating set* [56] or the *weakly connected dominating set* [57] in which the resulting nodes typically are less connected.

3.5 Evaluation of Effectiveness

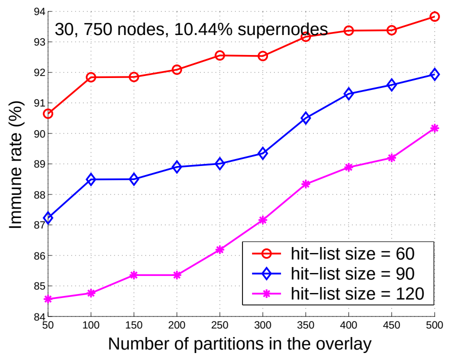
Environmental Setting To construct overlay graphs for P2P systems such as Gnutella 0.6 and KaZaA, we implemented two distributed crawlers [49, 36]. The crawlers walk over the network and adopt the membership protocol (PING/PONG mechanism) to collect topology information. They also make use of the two-tier overlay structure to reduce the crawling time. In our experiments, we set up a security server (a PC with Intel Pentium 4 2.5GHz CPU) to run a server thread, which coordinated the client crawler threads launched on 8 different PCs (with the similar configuration as the server PC). Snapshots of the overlay graph were assembled in the server side. According to our observations, in Gnutella 0.6, about 15% of the supernodes depart from the overlay within less than $3 \sim 4$ hours. To reflect network dynamics, we used a 4-hour crawling interval. Note that if the interval is too long, the topology graph in the security servers will become distorted; if it is too short, much overhead will be added to the crawling hosts and the normal P2P traffic.

To obtain various overlay networks, we selected different subnets (each has a population of 40,000 nodes) from the constructed topology graphs. Note that these subnets depict the logical connections between the participating hosts. We then randomly placed the hosts in a subnet into an Internet environment with 5050 hierarchical routers (generated using Georgia Tech's Transit-Stub Internet

Topology Generator [58]). The latency between each pair of hosts was computed according to the routers between them. We further designed a worm simulator in which a remote worm [45] exploits a common buffer overflow vulnerability and causes denial of service attacks to those unprotected hosts in the system. Initially, a number of supernodes are chosen by the attacker to form a hit-list. These initial victims accept incoming requests from the worm that overflow their stack buffers. Before these victims crash, they are hijacked to execute the malicious code provided by the requests and forward the same message to their neighbors. Each of our tests takes 50 runs. We report the mean of test data. To evaluate the schemes on their tolerance against node dynamics, we keep the graph knowledge unchanged while perform a number of rewirings on the underlying topology, according to the speed in which the network evolves.



(a) Topology I (7.96% supernodes)



(b) Topology II(10.44% supernodes)

Figure 3.3. Test results when applying the partition-based scheme in Gnutella 0.6 overlays (partial snapshots). (*a*) and (*b*) each shows the immune rate as a function of the number of partitions and the worm hit-list size, repectively

Partition-based Scheme Our results show that the server divides an overlay (40,000 hosts) into 300 $\sim$ 500 nearly-balanced pieces in one second and with an averaged CPU usage below 17%. Fig.3.3 (a)$\sim$(b) illustrates the results when the partition-based scheme is applied in various Gnutella 0.6 overlays. The figures indicate that a higher immune rate can be achieved when there are more partitions in the overlay. They also demonstrate that the size of worm hit-list has a negative influence to the defense. A further comparison reveals that an overlay with a higher percentage of supernodes achieves a higher immune rate. Fig.3.4 illustrates the proportions of key nodes needed in various overlays. The result indicates that we may nicely partition the overlay graph and achieve an immune rate higher than 90% by simply utilizing less than 10% of the population as key nodes.

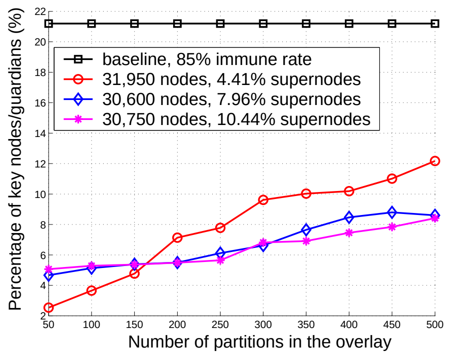


Figure 3.4. Fraction of key nodes needed in the partition-based scheme (Gnutella 0.6); it shows the percentage of key nodes needed to achieve an immune rate $\geq$ 85% (hit-list size = 90). The baseline scheme [1] requires more guardians.

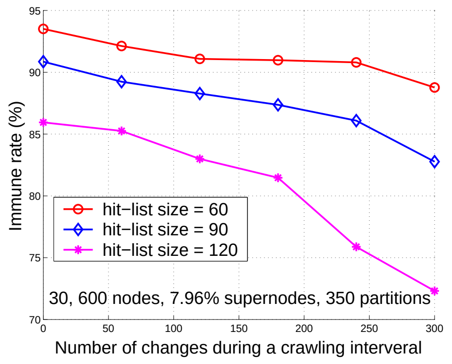


Figure 3.5. Impacts of node dynamics on the partition-based scheme (Gnutella 0.6)

Fig.3.5 shows the impacts of node dynamics on the partition-based scheme. Here the number of changes (e.g., node joins, departures, reconnections) reflects a degree of overlay dynamics. We notice that the partition-based scheme keeps a good result even when the servers' topology information becomes outdated due to the crawling delay. However, as the worm hit-list size increases, this tolerance deteriorates a little bit. In our test, 250 changes of supernodes typically happens within a 3-hour time period and the figure shows a good tolerance to a 3-hour crawling interval.

CDS-based Scheme We evaluated the performance of the CDS-based scheme in Gnutella 0.6 systems. Our results show that using the Guha and Khuller's algorithm [54], a security server can derive the CDS nodes of an overlay (40,000 hosts) in 0.5 second, with its averaged CPU usage below 23%. Fig.3.6 illustrates the containment result when security servers choose different proportions of CDS nodes to deliver worm containment alerts. Clearly, a larger subset results in a higher immune rate. For example, when 1/2 CDS nodes (12% of the total population) are selected, a worm with an hit-list of 500 nodes can be successfully contained (with a final infection rate below 5%). However, considering the bandwidth limit

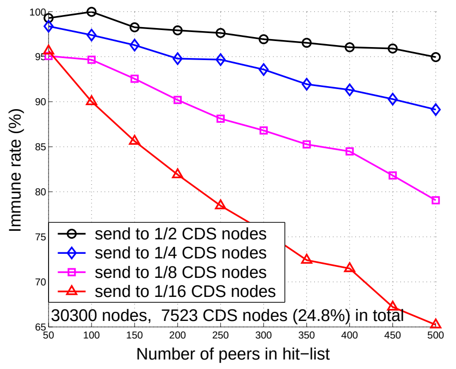


Figure 3.6. Immune rate vs. hit-list size and proportion of CDSs used in the CDS-based scheme

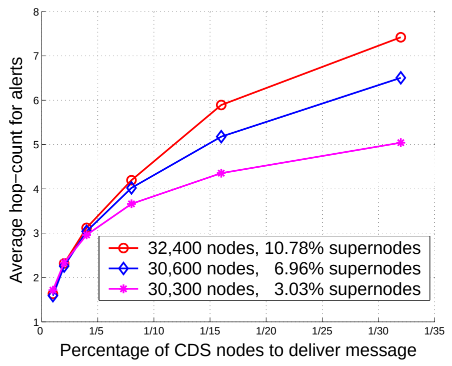


Figure 3.7. Average hop-count of alert dissemination as a function of percentage of CDSs selected

and the impact to the normal P2P traffic (this becomes worse when the messages contains heavy payload), security servers may want to push the message directly to a smaller node set. Fig.3.6 indicates that choosing 1/4 CDS nodes keeps a good balance between immune rate and overhead.

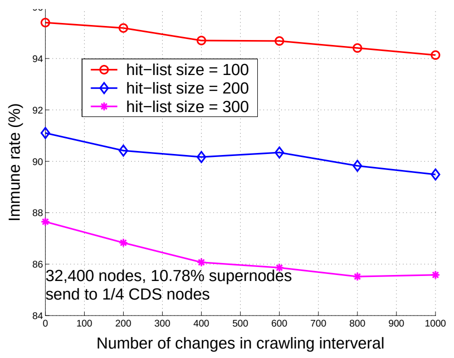


Figure 3.8. Impacts of node dynamics on CDS-based scheme

Fig.3.7 shows the average hop-count (logical) through which security alerts are forwarded. Different subset of CDS nodes were tested and the result demonstrates that a full set of CDS nodes results in the lowest hop-count (less than 2). As the subset becomes smaller, the hop-count increases, i.e., a security alert has to go through more forwarding nodes before reaching the destination. When the overlay has a higher fraction of supernodes, more forwarding nodes exist between a pair of end hosts and the average hop-count increases. Figure 3.8 illustrates the impacts of network dynamics on the CDS-based scheme. Compared with the partition-based scheme which depends more on an exact overlay topology (Fig.3.5), the CDS-based scheme shows more tolerance to the topology distortions: it keeps good performance even when there are 1000 changes among the supernodes. The crawling interval, therefore, is set to 12 hours.

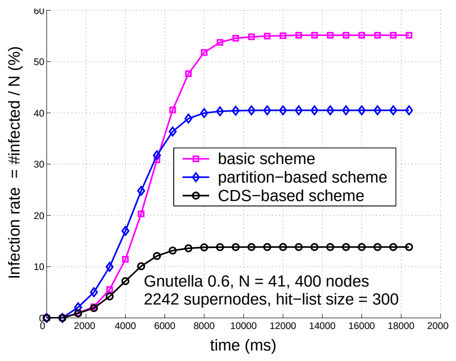


Figure 3.9. Infection rate vs. time

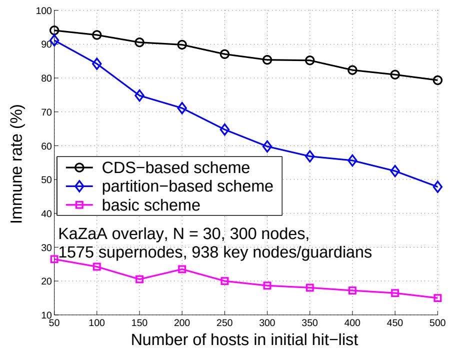


Figure 3.10. Immune rate vs. hit-list size

Comparing the Schemes We used the scheme [1] as the base-line approach and compared it with our schemes in different overlays (Gnutella and KaZaA). The comparisons were based on the same fraction of key nodes/guardians and the same size of initial worm hit-list. Fig.3.10 demonstrates that in a KaZaA system, both our schemes achieve a higher immune rate than the baseline approach. When the worm hit-list is relatively small, two schemes attain the similar immune rate. As the hit-list size increases, the CDS-based scheme outperforms the partition-based scheme. Fig.3.9 shows the change of the infection rate when applying two schemes in a Gnutella system. Initially the infected population increases as the worm launches the attacks, it is then slowed down as the defenses take effects. Eventually the system reaches a stable state. Both our schemes outperform the basic approach in that they help the system retain a lower infected population.

Chapter 4

Combating File-sharing Worms in P2P Networks

4.1 Introduction

P2P file-sharing programs such as KaZaA, iMesh, Morpheus are popular Internet applications that allow users to download and share electronic files. As one of the most popular networks, KaZaA has four million simultaneous users. The powerful data access feature however creates unique privacy and security threats in the system. Besides adware and spyware may be introduced into hosts, these machines may also be placed at the risk of viruses and other malicious codes [59].

Our focus in this work is *file-sharing worms*[1], which also are malware spreading through file-sharing applications in P2P systems. A file-sharing worm usually copies itself to a host's shared folder and publish it with an attractive name, for example, as a popular song or movie. Sometimes attackers replace real movie or sound files with their copies or add executable extensions to such files. When a host searches the system for some file and finds a match on an infected machine, it downloads and uses the file without being aware of the threat. Thus, the worm is activated and it copies/attaches itself to all the files in the shared folder(s) of this new victim host. In this way, the worm continues its spread cycle. Recently, many file-sharing worms have been reported attacking P2P systems, e.g., the Benjamin.a

[1]Like mass-mailing worms, file-sharing worms also require human's operations to propagate. As such, they are sometimes referred to as viruses or malware.

worm, Franvir worm, Bare.a worm, Darby.m worm, and Duload worm in KaZaA, iMesh, Morpheus and eDonkey2000 file exchange networks [60, 61]. These worms all replicate by copying themselves into the shared file folder(s) of a victim machine. One experimental study reports that 44% of the 4,788 executable files downloaded through a KaZaA client program contain malicious code [62]; another experimental study reports that over 12% of KaZaA client hosts were infected by over 40 different worms in February and in May, 2006 [63]. Some consequences of worm infections are opening backdoors, changing system registries and client configurations, and collecting client confidential data [61].

So far, more than 200 file-sharing worms have been reported already [60]. It would not be surprising that attackers will increasingly exploit P2P file-sharing as an additional attack vector. Clearly, without any defense against the worm attacks, they may eventually spread to the entire system. As such, before they cause havoc, a timely study on defenses against file-sharing worms is highly demanding. However, so far the problem has not received due attention. Currently, the focus of the research community is on combating Internet worms. Moreover, the earlier study was mainly on modelling the propagation of file-sharing worms, but few defense mechanisms have been proposed — they merely rely on some simple and passive means such as individual users detecting and cleansing the infected machines or reducing the sizes of shared folders (this apparently affects normal file-access services).

It would not be surprising that worms will increasingly exploit file-sharing applications as their major infection vector. However, research on these imminent threats has just started and most existing work focused on modeling worm propagations. Dimitriu et al. [64] used analytical models and simulations to study the resilience of P2P file sharing systems. They demonstrated that the effectiveness of file-sharing attacks is highly dependent on the clients' behaviors such as willingness to share files and quickness in removing infected files. Kumar et al. [65] developed a suite of fluid models that model pollution proliferation in P2P systems. These fluid models lead to systems of non-linear differential equations. They captured a variety of user behaviors and described how the injection of multiple versions of corrupted content impacts a client's ability to receive a valid copy. Their analysis revealed intelligent strategies for attackers as well as strategies for clients seeking to recover non-polluted content within large-scale P2P networks. Thommes et al. [66] derived deterministic epidemiological models for the propagation of file-sharing viruses through a P2P network and the dissemination of pollution. They showed by simulations that the models remain sufficiently accurate despite variations in individual peers and provided insight into the behavior of the file-sharing system. They also used the models to examine some mitigation techniques such as deploying the Credence system [67] in the network to increase the elimination rate of infected files.

Since most of the above work focuses on modelling the evolutions of file infection or pollution in various P2P systems, none of them are directly applicable to the study of file-sharing worm *containment*. Using Thommes's epidemiological models [66] as an example. They failed to consider that a node could become immunized to the same worm attack once it has the security patch installed; they also failed to reveal the evolution of files and its influence to the node states, and their assumption that the uninfected files in the system remain the same during worm propagation is problematic. For these reasons, we need to derive a more accurate worm propagation model without the above limitations. Other closely related work is on security patch and alert distribution. Zhou et al. [1] explored the feasibility of applying a mitigation mechanism which disseminates self-certifying alerts (SCA) [32] to immunize vulnerable hosts under the threat of scanning worms. SCAs are proofs of vulnerability that can be inexpensively verified by any receiver. Vojnovic et al. [68] studied the efficacy of patching and filtering techniques in protecting a network against scanning worms. Their study shows that whether alerts and patches can be propagated fast enough to limit the spread of worm attacks depends on the speeds of different processes, namely, worm spread, alert spread, and patch download from trusted sites.

We have to admit that the problem of quarantining P2P worms has not been adequately addressed by security vendors (e.g., Microsoft and Symantec, McAfee) either. Currently the best defense mirrors the strategy against Internet computer viruses with the inception of security patches from vendors. The method of automated security update is widely employed by security servers to automatically push the latest security patches to Internet hosts[68]. This generic solution certainly helps protect Internet hosts at the earliest stage of worm spreads, but it is not P2P-oriented. That is, security servers either have to blindly deliver P2P patches to all the Internet hosts (including those non-P2P machines and those who have installed P2P software but are not executing it), or have to scan for currently running P2P client programs within each Internet host before sending a patch to it. In both cases, system resources and network bandwidth could be greatly wasted. Moreover, unnecessary security updates could cause annoying machine reboots (sometimes required for a complete security update), which unavoidably interrupt those non-related users' on-going tasks running on their hosts.

In this work, *we study the feasibility of utilizing the existing file-sharing infrastructure to internally push security updates to the participating nodes in P2P systems.* We propose two BitTorrent-like mechanisms for distributing the security patches. In file-sharing networks such as Gnutella, a very small fraction (5%) of hosts usually provide a large fraction of the shared files (70%) [69]. Exploiting this asymmetry in file-sharing, we consider first disseminating the security patches to these popular hosts, such that most of the other participating hosts can receive the patches from these popular hosts when they actively download files from them. Our second approach is based on the belief that P2P users as a community should help each other in combating worm attacks. Therefore, when a host detects worm infection from a downloaded file, it first re-performs a search on the infected file to identify those hosts possessing the same file, and then it collaboratively notifies these hosts of the worm information as well as the security patch. Based on a modified fluid model, we analyze worm spreads and evaluate the effectiveness of our approaches in unstructured networks. Our result demonstrates that both schemes can help a file-sharing system with 20, 000 hosts achieve a high immunity rate (90%) within a few dozens of hours after the initial worm surge.

Our solutions are not a substitution for the existing automatic patching systems but rather a nice complement to them. Our proposed techniques are not necessarily very complex, but our work, backed up with solid analytic modelling and extensive experiments, makes a concrete movement towards solving the important security problem facing many P2P users. Also, our solution is scalable and easy to deploy by leveraging the existing P2P infrastructure, without involving a dedicated Content Distribution Network (CDN) (e.g., Akamai).

4.2 System Model

Network Model Many P2P file-sharing systems are actively running in these days (a comparison can be found in [37]). The most popular ones include eMule, KaZaA, Gnutella, and BitTorrent. For concreteness, however, our discussion will focus on those unstructured networks such as Gnutella and KaZaA.

We use Gnutella as an example to describe the file-sharing process. Specifically, each node uses a *shared folder* to store those files it wishes to share. When a requesting node initiates a download request for a specific file, it places a search for the target node(s) responsible for the given file identifier. The search request is routed through a two-tiered system of ultra-peers and leave nodes in the Gnutella overlay. In response, the requester collects a list of peers, each of which contains a file copy (probably with different versions). The requester then connects to one target node in the list and downloads the copy. Finally, she opens the downloaded file for use.

Attack Model A file-sharing worm usually copies itself to a host's shared folder and publishes it with an attractive name, for example, as a popular song or movie. Sometimes attackers replace real movie or sound files with their malicious copies or add executable extensions to such files. When a host searches for some file and finds an match from an infected machine, it downloads and opens the file without being aware of the threat. Consequently, the worm is activated and it copies/attaches itself to all the files in the shared folder(s) of this new victim. In this way, a file-sharing worm continues its spread cycle.

We define two states for a file in P2P systems: *normal* and *abnormal*. A file is normal when it is valid and clean, and it becomes abnormal once malicious codes have been injected or attached to it. Also, we define three states with respect to a surging worm for each host in the systems: *vulnerable*, *infected*, and *immune*. A vulnerable host is not well-protected against the worm, hence it gets infected when exposed to the attack. For example, when a user opens an downloaded file which is abnormal, all files inside the shared folder(s) of this infected machine consequently become abnormal. A vulnerable/infected node becomes immunized once the protection (e.g., a patch) has been in place. Fig 6.2 illustrates the node state transitions. We note that in real applications, some P2P users could voluntarily install the vendor patch on their machines. Therefore, these nodes are initially immunized to the worm. For simplicity, we assume no such individual recoveries occur during the period of defense.

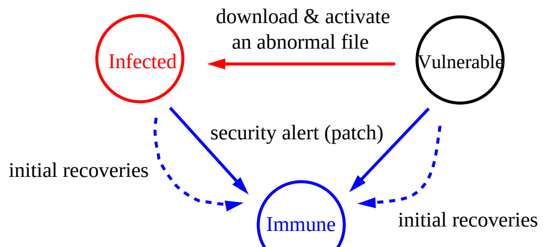


Figure 4.1. Node state transition during the defense of internal patching. Initial recoveries include individual updates from security vendors; its percentage is relatively low as a new worm surges.

We notice that a few elaborated worms such as Worm.Win32.Hofox were recently reported to be able to block the anti-virus protection services or kill anti-virus programs on P2P hosts. Clearly, at the system level, some local countermeasures will be devised to protect defense tools from being eliminated, and the arms race will continue. In this dissertation, however, we assume that P2P worms cannot disable the patching protocol deployed in end hosts, so that an infected host can receive patches and become immunized as it is expected. Note that this assumption will not affect the correctness of our approaches, and our analysis model presented later can be slightly changed for the case when this assumption does not hold.

4.3 System Overview

To effectively combat file-sharing worms, we cannot merely rely on users' precaution and worm elimination skills; rather, we need *an automated and systematic approach to disseminate security patches to the P2P users.* Existing automated patching systems can be utilized to secure P2P hosts as well by simply treating them as normal Internet hosts; however, they are not necessarily the best-fit choices because not all Internet hosts are equally exposed to those P2P worms. For P2P users who often download files, their machines are more likely to be affected by P2P worms, whereas for non-P2P users, their machines will not be affected by worms exploiting P2P applications. Moreover, a traditional centralized model of patch distribution could cause single-point failure or overloading on the patch servers.

As such, we are motivated to study a P2P-tailored automated patching mechanism as a supplement to existing solutions and examine its effectiveness. Our approach utilizes the existing file-sharing infrastructure to internally push security updates (alerts) to the participating peers. It has several good features. First, it is customized for P2P environments and delivers security updates to P2P hosts only. This avoids unnecessary consumption of network/computer resources. Second, it adopts a distributed manner to disseminate security patches to those vulnerable peers in need and no longer strains the central servers. Third, our push-based scheme delivers security updates more promptly than the traditional once-a-day update adopted by existing patching systems.

Internal patching should leverage the existing file-sharing infrastructure for distributing security patches. Approaches utilizing IP address scanning or topology exploration to locate alive patching targets bring extra computation and communication overhead to P2P systems. Moreover, these methods could be easily exploited by malicious users and used as the vehicle for rapid worm spread and denial-of-service attacks.

We choose to study two two push-based patching mechanisms for P2P systems. We first examine a download-based approach, in which a small fraction of popular nodes (also referred as *key nodes*) act as early patch distributors and a node which downloads a file from a key node will also be offered with a security patch/alert. Thus, the patch is propagated to many active hosts along with the file-download process. We then examine a search-based approach, in which once a key node detects worm infection in a downloaded file, it re-performs a file search to identify those active hosts that possibly possess the abnormal file and disseminates the patch to immunize/disinfect them.

In the following sections, we will address the issues related to the realization of these approaches and evaluate their effectiveness. We assume each node keeps simple file-sharing records such as its file uploading and downloading rates.

4.4 A Download-based Approach

In our download-based approach, a small set of key nodes internally push the security patches to participating peers through the file-downloading process. Nodes that are notified of the approaching threat hence have a good chance of being immunized/disinfected against the worm. The design of our scheme involves answering the following questions.

- Which hosts in the system should be decided as the key nodes so that they distribute security patches to others in a most efficient and timely manner? Key nodes cannot be determined in a centralized mode because no node in the system holds a global knowledge of file-sharing activities of others.

- What is the user's strategy to choose a download source and what is its impact on the patch dissemination? How should the existing P2P file transfer protocol be utilized to support the patch dissemination?

- How could a receiver verify the authenticity of the patch messages? In a distributed P2P system, even if a public key infrastructure (PKI) may be deployed to provide sender authentication, it cannot prevent malicious peers from injecting worms instead of security patches. In other words, a node cannot fully trust others in the system. Moreover, how does the receiver process the patch and how does her decision affect the immunity level of the system?

4.4.1 Scheme Description

Bootstrapping Key Nodes The first important issue is the choice of key nodes. An immediate thought is that vendors such as Symantec and Microsoft deploy some dedicated patch distribution servers in the system and hope users to contact them for security patches. However, a user may not be interested in actively searching and obtaining security patches from these machines, given that some more automatic and trustable channels (such as directly downloading from a vendor site) exist; even if they are deployed, to be effective in distributing patches, many of them are needed, causing high deployment and maintenance costs. As such, a better alternative is to utilize some ordinary nodes in the system as the patch distributors (key nodes). In most decentralized systems (e.g., Gnutella, KaZaA), downloading traffic is highly focused around a small minority of popular items and these popular files tend to be gradually concentrated in a small set of providers. For example, in Gnutella, 50% of all files are served by just 1% of nodes and 98% of all files are shared by the top 20% nodes [69]; in KaZaA, 10% most popular files generate 60% of the download traffic and 70% of the highly popular files will remain popular for at least 10∼15 days [70]. These are strong indications that a small fraction of popular hosts which share most interesting files could be conveniently utilized as the distributors to push security patches to those active downloaders in the system.

We consider a distributed algorithm for bootstrapping key nodes in the P2P systems. Initially, a small set of key nodes are individually decided according to a predefined policy. These key nodes then automatically download and launch the vendor patch so that they become immunized against the surging worm. To describe the algorithm in detail, we first introduce a parameter named *file-offering rate* ϕ_O, which is defined as the number of files a node offers to its requesters in a unit time. This parameter reflects a degree of node popularity in the system. Typically a node calculates its ϕ_O based on its own file-sharing history. For example, node i may derive $\phi_O(i) = D_{out}(i)/T_f$, where $D_{out}(i)$ denotes i's out-degree in its file-access graph within its neighborhood time window T_f. Now we may adopt the following policy to bootstrap the key nodes: *key nodes are selected from a subset of popular nodes with the highest file-offering rates in the system.* Specifically, each candidate node i refers to its recent file-offering rate $\phi_O(i)$ and decides to be a key node only if $\phi_O(i) \geq H_O$ satisfies. Here H_O is a globally defined threshold, which controls the fraction of key nodes. This policy is automatically enforced through the client program. Once a node decides to become a key node, it should automatically fetch the latest security updates (if there are any) from the trusted vendor(s) and immediately launch the protection on the local machine (another option is they register to the vendors so that the vendors may push the latest security patch to them once available). In this way, the key nodes get immunized against the surging worm and are ready to assist those file requesters. We note that as an active holder of more popular files, the user has to sacrifice a little

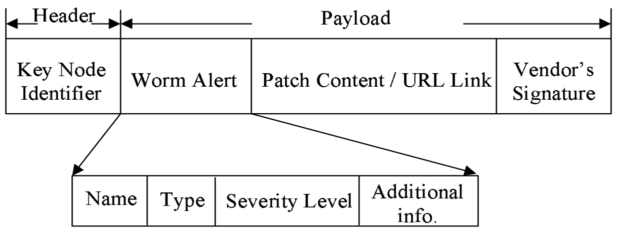


Figure 4.2. Message format of security patch MSG_a

bit freedom (patch activation if needed) and bandwidth (patch transfer) for the security of the entire system. On the other hand, a key node might be malicious or a regular node may claim to be a key node. We will discuss the related security issue shortly.

Disseminating Security Patches Next, we discuss the format of a security patch generated by the key node. This patch is used to notify the receivers of the worm threat and to provide the source of the security update. As illustrated in Figure 4.2, a patch message MSG_a typically contains two parts: a message header which contains the key nodes's identifier, and a message payload which contains (1) the worm alert (name, type, severity level, etc..), (2) the security patch itself (e.g., a Microsoft XP patch in binary delta compression format [71]) or simply a link to the URL of that patch (e.g., the Microsoft Security Bulletin), (3) a vendor signature of (1) and (2). We note that for a specific worm (defined by a specific vendor), the payload of its security patch message is unique. Also, the security patch is *self-verifiable*: either a signature of the well-known vendor is attached to the patch, or the link can be verified on the vendor's website. This mechanism of secure software updates has been adopted by many vendors [72]. For example, since the public key of Microsoft is stored in a user's machine during the installation of Windows OSes, whenever the user receives a software update from Microsoft, she starts an authentication before accepting the update. Similarly, our scheme does not require the receiver to authenticate the patch distributor. Instead, it directly verifies the authenticity of the message content with the vendor or through its web site – these are considered more reliable and trustable.

The next issue is how key nodes may utilize the existing P2P file transfer protocol to internally distribute the security patches to the file requesters. Using Gnutella 0.6 system as an example, search results are delivered over UDP directly to the node who initiated the search If the user decides to download the file from a resulting node, both sides should establish an additional channel or use the existing control channel to perform patch transfer. Typically this requester delivers an HTTP Request to the provider and reads the bit stream of the file content that follows the HTTP Response [73]. We propose the downloader includes its latest patch version in the HTTP Request. Thus, a key node may verify this request and decide whether to deliver an security patch to the downloader just before the file transfer. To indicate the existence of a security patch to the receiver, the key node simply specifies an extra field for the patch content in its HTTP Response. In the case when the provider is behind a firewall, a PUSH process is executed to establish the connection [73] and the remainder of the file download and patch dissemination is identical to the above.

Client Strategy and User Behavior In response to a file search, a client receives a set of replies pointing to different file providers. The main decision that the client needs to make is which one of these node to ask for a copy of the file. This choice clearly has some influence on the defense. We consider the following three major selection strategies:

Random. The client selects a random node, independent of the node's advertised resources. In this mode, when there are α percentage of key nodes in the system, every client has an equal chance of α to download the file from a key node. This also implies that every client will eventually receive the security patch from key nodes.

Best. The client selects the node that advertises the best performance, i.e., the node with the lowest estimated delay (the node's queue length times the file size times the maximum number of simultaneous uploads divided by the access link bandwidth). Unlike the random mode, in this mode every client has a higher chance of downloading files from key nodes (i.e., those popular file holders). However, a small fraction of nodes which are not interested in the popular files may not have a chance to receive the security patch from key nodes.

Redundant. The client performs redundant download from either randomly chosen C nodes or C nodes with the lowest estimated delay. Once the first download finished and the content is verified for correctness, the other downloads are stopped. When the file download from a key node is aborted, the client cannot receive the patch that follows.

Next, we discuss how receivers process and react to the security alert. Although our approach effectively leverages the internal infrastructure to expedite patch propagation and ensures most participating pees receive the update as their file downloads proceed, the immunity level of the system is still in some degree determined by users' responses to the patch update. Upon receiving a security patch MSG_a, the client program first examines the message payload and compares it with the existing version of the patch. Any out-dated or duplicated patch will be discarded. An accepted patch typically notifies the user of an emergent worm threat in the system. Such P2P-oriented warning reminds the users to immediately launch the protection. However, the user's decision still matters even if her incentive of patch acceptance has been boosted. If she accepts the patch, the application authenticates the patch payload either by directly examining the vendor signature or by visiting the trusted vendor site and verifying if the patch link is consistent with the specific worm information. The application applies the new patch in the local machine (sometimes it may first download the content) immediately after a successful authentication. In this case, the host gets immunized/disinfected against the surging worm and consequently all the files in its shared folder become normal. However, when a user declines the offer, either unwilling to follow the link to install the patch or failing to activate the patch coming with the message, her machine remains vulnerable to the worm. We will quantitatively analyze the impacts of user behavior on the system immunity level in our work.

4.5 A Search-based Approach

This section proposes a search-based approach, in which once a key node detects worm infection in the file it has just downloaded from other participating peers, it immediately performs a search and exploits the result to infer a set of suspicious hosts, to which it pushes the security patch to disinfect or immunize them. This is a reactive defense because patch dissemination is triggered by the detection of a worm instance during the file downloads. Given the latest vendor updates, we assume key nodes are able to detect on-going worm attacks based on techniques such as worm signature matching, taint analysis, or anomaly detection [74, 75, 76, 77, 78].

Next we will focus on answering the following questions in our design.

- Which hosts in the system should be chosen as the key nodes so that they detect file anomalies and distribute patches to others in a most efficient and timely manner? Key nodes should also be bootstrapped in a distributed way.
- Once the key node has detected the file anomaly, how does it infer a set of suspicious nodes through exploiting the query responses and how does it deal with network dynamics?
- How does the key node disseminate the security patches to those suspicious nodes? To be scalable, how should the key node limit its bandwidth for patch delivery?
- What is the user's reaction towards the security patch and how does it influence the immunity level of the system.

4.5.1 Scheme Description

Bootstrapping Key Nodes To address the first issue, we consider a distributed algorithm to bootstrap key nodes in the search-based approach. Similar to the algorithm in Section 4.4.1, key nodes automatically install and launch the latest vendor patches so that they become immunized against the surging worm. However, here we adopt a different policy for individually determining key nodes. We first introduce a parameter named *file-downloading rate* ϕ_I, which is defined as the number of files a node downloads from others in a unit time. This parameter reflects the activity level of the downloader. Each node i may derive $\phi_I(i) = D_{in}(i)/T_f$, where $D_{in}(i)$ denotes the number of files i has downloaded within the time window T_f. Now we may adopt the following policy to bootstrap the key nodes: *key nodes are selected among a subset of nodes with the highest file-downloading rates in the system.* Specifically, each candidate node i refers to its recent file-downloading rate $\phi_I(i)$ and decides to be a key node only if $\phi_I(i) \geq H_D$ satisfies. Here H_D is a globally defined threshold which controls the fraction of key nodes. This policy is automatically enforced through the client application.

The above policy chooses those active file requesters as the key nodes because these nodes keeps actively downloading and activating files from various sources, hence their chance of being infected is relatively higher than normal hosts. Keeping these nodes updated with the latest vendor patches not only protects these vulnerable nodes themselves against the worm attack, but also provides these nodes with the ability to detect the infection from the file providers. In the search-based approach, whenever a key node P finishes downloading a file f_p, it immediately examines the status of the file to detect possible attacks. Once an anomaly has been identified in the downloaded file, the key node immediately composes a security patch Msg_a using the format we discussed in Section 4.4.1.

Inferring Suspicious nodes The next issue is to which nodes these security patches should be disseminated. Pushing the patch directly to the provider who has uploaded the abnormal file is effective. However, this is not efficient because the key node has a good reason to suspect that other file-owners may have also been infected. On the other hand, simply flooding the patch or locating the targets by IP address scanning or topology exploration is not scalable. Our solution is to let the key nodes exploit the file search list to locate those *suspicious* file providers and disseminate the patch to these nodes. However, there exist a time gap between the original file search and the worm detection and this time gap could be in hours (determined by the file downloading rate $1/\lambda_d$). During this period, nodes may frequently join and leave the network. A good strategy for the key node is to re-perform a file search once it has detected any anomaly in the downloaded file. From the search result, it may acquires the latest file-sharing information and node statuses based on which it may further identify those suspicious targets.

We use Gnutella 0.6 protocol [73] as an example to show the mechanism of a file search list. When a user initiates a query and delivers a Query message (0x80), his request is routed through a tier of ultrapeers using QRP (Query Routing Protocol). Each peer receiving the request matches the search criteria against its local shared files and sends a QueryHit message (0x81) along the same path that carried the incoming Query message once a match is found. The user then aggregates these QueryHits and acquires a *search list* of peers who currently share the file. Besides the description of the resulting file, each QueryHit message also contains (1) The IP address and port number of the responding host; (2) sufficient information for the user to evaluate the download chance from this peer, such as the bandwidth of its access link (in Kb/second), its queue length, the maximum number of simultaneous uploads, etc. If the key node has determined that one or several copies of the file are abnormal, it is very naturally to suspect that some recent accessors of the file could also been infected.

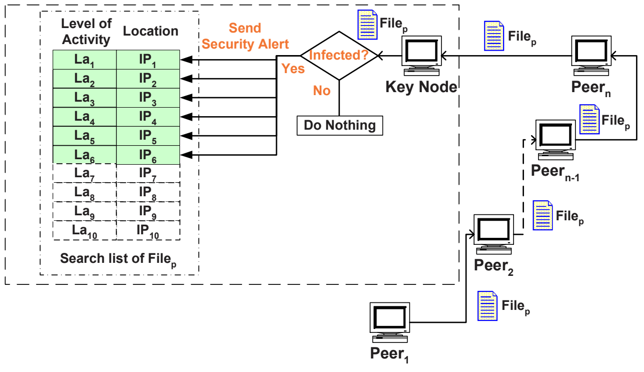


Figure 4.3. An illustration of the search-based approach. In this example, key node P detects an infected file f_p and delivers security patch MSG'_a to $k = 6$ suspicious nodes in the search list S_p.

Delivering Security Patches Next, we show a distributed algorithm for disseminating the security patch. Once a key node P has detected a worm infection in the downloaded file f_p, it immediately re-perform a search on f_p and consequently receives multiple (usually in hundreds) QueryHit responses. It sorts the destination nodes according to their current *activity level* and constructs a *ranked* search list S_p. Here a node i's activity level $L_a(i)$ is computed from its access link bandwidth $Spd(i)$, queue length $QLen(i)$ and number of uploads $N_{up}(i)$ in the QueryHit message, i.e., $L_a(i) = f(Spd(i), QLen(i), N_{up}(i))$, where f is a monotonically increasing function. Based on this ranked search-list S_p. the application in the key node computes an activity lower bound $H_L(P)$ based on the bandwidth $Spd(P)$ and the current number of connections (ongoing P2P traffic) in the local machine. The key node will always choose from S_p top k target nodes whose activity level satisfies $L_a \geq H_L(P)$ and establishes a direct HTTP connection with each of these suspicious nodes to push the security patch to them. Note that this patch transfer is out-of-band (not transferred over the Gnutella overlay). Figure 4.3 illustrates an example of the search-based approach.

In the above process, a security patch message MSG'_a also contains the identifier of the infected file f_p. Upon receiving this patch, each suspicious node, say j, needs to verify the existence of f_p and then displays a warning message in the local machine. Similar as in Section 4.4.1, a user decides to launch the patch update or simply ignore it. If she accepts the update, the application first authenticate the patch content/link to ensure that it is from a trusted vendor. Once the patch has been successfully applied, node j becomes immunized to the worm threat and its shared folder will be immediately scanned and cleansed. We will quantitatively analyze the impact of user behavior on the system immunity level in our work.

4.6 Security and Performance Analysis

We discuss two attacks that may happen in both schemes.

Fake Security Alerts A malicious node, either a key node or a regular node claiming to be a key node, may replace security patches with worms and deliver them to other hosts. This attack will fail because our signature-based mechanism allows a receiver to verify if the patch truly comes from a trusted vendor or the link to the patch is correct. On the other hand, we notice that a lot of false messages may cause a DoS attack to other hosts. Since we do not assume a PKI, P2P nodes may not be able to authenticate each other. Indeed, even a PKI is available, it does not solve this type of insider attacks. A simple solution is that a node blacklists the nodes reporting false alerts based on their IP addresses. To prevent IP spoofing, before a node accepts a security alert, it challenges the source.

Patch Suppression Attack A malicious (or selfish) candidate key node may not propagate security patches. That is, in the download-based approach, it does not offer the security patches to downloaders and in the search-based scheme it does not care about other susceptible nodes. This patch suppression attack will degrade the effectiveness of our schemes. However, it only decreases the actual α. As long as they are not a lot, our schemes will still work. Otherwise, we should increase the value of α.

We analyze the performance of the internal patching schemes and provide the theoretical results in Appendix A.

4.7 Evaluation of Effectiveness

Environmental Setup We evaluated and compared our schemes in a variety

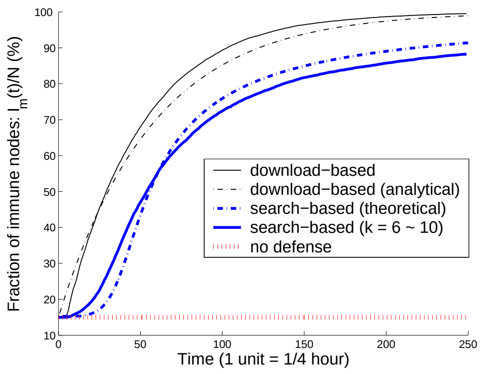
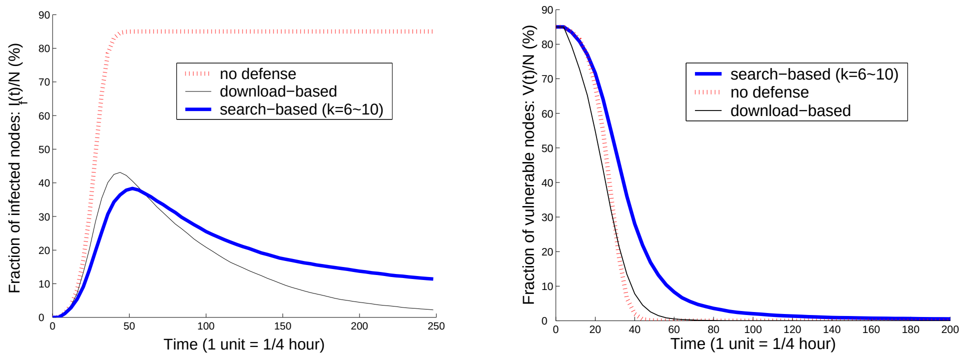


Figure 4.4. Comparing performance of different patching mechanisms in Gnutella 0.6; fraction of immune nodes vs. time; N=20k nodes, M=1k files, λ_d=1 file/hour, λ_a= 1.0, α=10%, β=0.7

of file-sharing systems. For unstructured networks we implemented a Gnutella simulator based on Gnutellasim from limewire.org; for structured networks we used P2PSim ([79]) to construct a basic Chord [4] infrastructure for routing queries and responses. We implemented a protocol similar as NeuroGrid [80] to generate large-scale file-sharing traffic on top of the routing infrastructures. We studied the case when a file-sharing worm Benjamin.a [81] surges in the network and evaluated the effectiveness of our countermeasures.

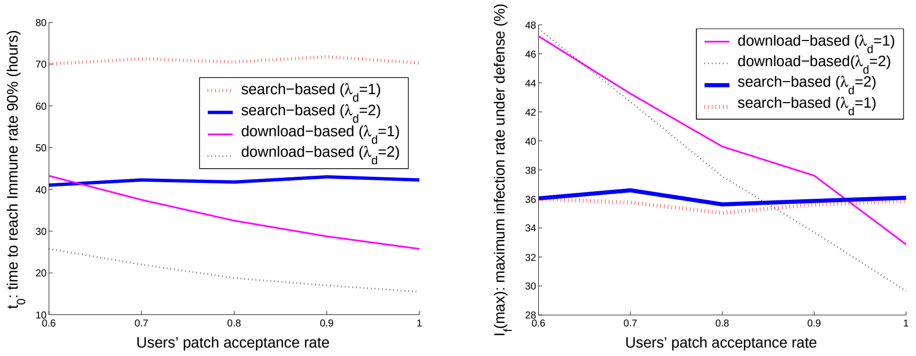
We adopted the following metrics in our evaluations: t_0, time takes to reach an immunity rate $\Psi = 90\%$; $h(t_0)$, the the percentage of abnormal files at time t_0, which also reflects the infection rate $I_f/N(t_0)$ when $\lambda_a \rightarrow 1$; $I_f(max)$, the maximum infection rate which indicates how severe the system has been attacked. For each scheme, we also investigated the system evolution status, the impacts from user behavior and the message overhead. To examine the schemes' tolerance against node dynamics (joins/departures), our implementation followed the observations from Gnutella 0.6, i.e., 45% of the nodes quit the network in less than $4 \sim 5$ hours, and 22% persistent node tend to stay in the network for longer than 24 hours. Each of our experiments takes 100 runs. We report the mean of the measurement results. Unless otherwise indicated, in all our tests, the total population $N = 20,000$ nodes. The number of files (with different contents) varies from $1,000$ to $10,000$ and the average size of shared folders ranges from 5 to 50 files. We set the initial the percentage of abnormal files $h(0) = 1.5\%$, the initial infection $I_f(0)/N = 0$ and the initial immunity rate $i(0) = 15\%$. Among these immune nodes, $\alpha = 5 \sim 10\%$ of the entire population were bootstrapped as key nodes and each of them obtained the latest security updates from vendors.



(a) Fraction of infected nodes vs. time (b) Fraction of vulnerable nodes vs. time

Figure 4.5. Comparing performance of different patching mechanisms in Gnutella 0.6; N=20k nodes, M=1k files, λ_d=1 file/hour, λ_a= 1.0, α=10%, β=0.7

Scheme Effectiveness We compared the time performance and the system evolution status of different approaches, using the same set of parameters (e.g., λ_a, λ_d, α and β). We also used the no-defense case as the base line. Our test results are shown in Figure 4.5. Figure 4.4 illustrates the change of immune population over time. Without any defenses, the system keeps a low immunity rate and has to rely on individuals' patch updates. The download-based approach and the search-based approach both significantly increase the immunized population. The former takes around 35.5 hours to achieve a 90% immunity rate while the latter takes around 62.5 hour due to its reactive nature. The download-based approach largely depends on the activity level of file downloads and the search-based approach is triggered by worm detections. Figure 4.5(a) shows the change of the infected population over time. Without any defenses, the worm spreads in a relatively high speed and infects all the vulnerable hosts within 9.5 hours. Both our schemes effectively help the system reduce the infected population by internally pushing the security patch to disinfect those victims. A further comparison indicates that the search-based approach has a relatively slower disinfection speed; it takes 62.5 hours to reduce the overall infection rate to below 10%. However, it keeps a lower maximum infection rate ($I_f(max)/N = 37\%$). On the contrary, the download-based approach takes 45 hours to reach an infection rate below 5%, but it yields a higher maximum infection rate ($I_f(max)/N = 44\%$) in the system. Figure 4.5(b) illustrates the change of vulnerable population over time. Without any defenses, the vulnerable population quickly drops to zero (within 10 hours) as more and more nodes get infected during file downloads. Our schemes effectively slow down this process by either immunizing the vulnerable hosts or disinfecting the victims. We can see that after around 62.5 hours, there remain few vulnerable nodes and victims in the system and the immunity rate exceeds 90%.



(a) t_0 vs. $(\beta,\ \lambda_d)$ (b) $I_f(max)$ vs. $(\beta,\ \lambda_d)$

Figure 4.6. Impact from user behavior on the defense scheme (Gnutella 0.6)

User Impacts and Overhead We evaluated the impacts of system parameters and user behavior on the defense. Figure 4.6 illustrates the test result in Gnutella 0.6 system. Figure 4.6(a) shows that both schemes take less time (t_0) to achieve an 90% immunity rate as the speed of file download (λ_d) increases. For the download-based approach, a higher download speed results in a faster patching process; for the search-based approach, a higher download speed leads to more worm infections and this in turn speeds up the patching process. The figure also indicates that in the download-based approach, t_0 gets reduced as users become more willing to accept the patch (β increases). However, this is not distinct in the search-based scheme due to its reactive nature. When more users are patched, the defense also gets slowed down. Figure 4.6(b) shows that in the download-based scheme, the severity level of worm attacks ($I_f(max)$) quickly drops as β increases. When $\beta \geq 0.85$, the maximum infection rate in the system is below that of the search-based scheme.

Figure 4.7 illustrates the message overhead of the defense schemes. Using Microsoft XP as an example, the patches during SP2 are in binary delta compression

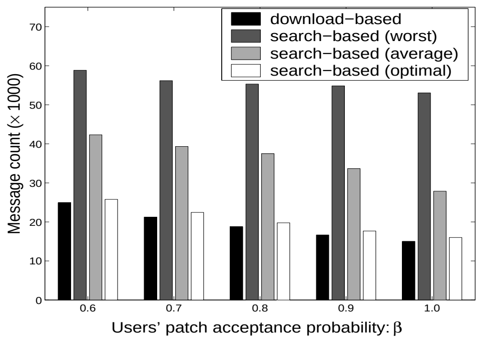


Figure 4.7. Message overhead of the defense scheme (N = 20,000 nodes, $\Psi = 90\%$); the download-based approach has less overhead than the search-based one due to no alert duplicates.

format [71] and the mean patch size is 32.9 KBytes [82]. This patch and its vendor signature (typically around 300 Bytes) constitute the main part of the payload in an alert message. Thus, the average length of a patch message is 33.2 KBytes. The figure shows that when β increases from 0.6 to 1.0, the message count of the download-based approach decreases until finally it reaches around $20,000 \times (90\% - 15\%) = 15,000$. We examined three cases for the search-based scheme: (1) *worst case* in which each key node simply delivers the patch to its targets. Patch messages could be duplicated and the message count is above 50, 000; (2) *average case* in which a key node does not deliver a patch to the same target and the message count is above 28,000; (3) *optimal case* when key nodes collaborate to avoid patch duplicates or each node indicates its current patch version in the QueryHit response. Hence, there are few patch duplicates and the message count approaches 15, 000 when $\beta \rightarrow 1$.

Chapter 5

Designing Device-level Countermeasures against Cell-phone Worms

5.1 Introduction

Mobile communications supporting both voice and data services become ubiquitous and essential in people's daily life. This popularity however also makes cellular networks and mobile devices such as cell phones and PDAs attractive targets to various malicious attacks. The introduction of new functionalities (such as email, file download, Blue-tooth communication, etc.) and the adoption of common operating systems (such as Symbian [15] and Windows CE [16]) in mobile devices make them even more vulnerable to attacks. This situation gets even worse as mobile devices evolve very quickly. Today's cell phone, especially the thriving smart-phones are typically as powerful as a-few-year-old PCs. This indicates the trend that cell phones will soon be facing with the similar worms that are terrorizing today's PCs. According to F-Secure [83], currently there are more than 200 mobile worms (or viruses) in circulation. Examples of the most notorious threats to cell phones include the Skull [34], Cabir [35] and Mabir [84] worms targeting at the Symbian operating systems. The research organization IDC estimates that by 2008 the market for mobile security software will grow yearly by 70%.

In our work, we refer to these worms as *cell-phone worms*. Cell-phone worms are malicious codes that exploit the vulnerabilities in cell-phone software and propagate in the network through popular services such as MMS (Multimedia Messaging Service), Blue-tooth communication, or both. Traditional cell-phone worms (such as Skull [34], Cabir [35] and Mabir [84]) take control of a phone's Blue-tooth interface and continuously scan for other Blue-tooth-enabled devices that enter the scanning range, and finally infect any such devices. Due to the limited radio coverage and the localized wireless connections, worm propagation through Blue-tooth alone is usually too slow to spread throughout the entire network. Some latest cell-phone worms also utilize the network infrastructure to achieve faster and more global propagation. For example, CommWarrior [33], which continues to wreak havoc among Symbian phone users in many countries, scans an infected phone's contact list and randomly sends itself in an MMS message. Because most users would already trust the originator, they usually accept the attachments in the messages and unwittingly get their phones infected. This process is repeated and more cell phones could be quickly infected.

Cell-phone worms are destructive both to the users and to the network infrastructure. A user of an infected phone will be unconsciously charged for numerous messages sent by the worm, and the battery of the cell phone will be drained very quickly. Moreover, the reported damage caused by cell-phone worms so far extends from the loss of data and privacy to damaging the device hardware. For example, the first trojan spy for Symbian phones named Flexispy [85] records information of the victim's phone calls and messages, then sends them to a remote server. In addition, automated cell-phone worms generates huge unauthorized traffic to cause misuse or denial-of-service to the network infrastructure. Therefore, both cell-phone designers and network service providers must employ appropriate countermeasures to get well prepared for the surging worm threats from cell phones.

Cole et al. [86] analyzed computer worm propagation and the impact of mitigation in mobile ad-hoc networks. Mickens et al. [87] introduced a new probabilistic queuing model to investigate malicious worms (e.g., Cabir) which spread through Blue-tooth links among cell phones. Their model treats node mobility as a first-order concern. Abhijit et al. [88] designed an agent-based model for the worm spread. Their work reveals various phone vulnerabilities and the possible severity of hybrid threats. Shackman [89] discussed platform security for Symbian OS. Hurman [90] demonstrated the techniques of exploiting Window CE vulnerabilities. F-Secure [83] and Symantec [91] provided worm-signature–based solutions to detect and eliminate various worms from the smart phones. Mulliner et al. [92] demonstrated a proof-of-concept exploit that crosses service boundaries in Windows CE phones. They used the labeling technique to protect the phone interface against malicious attacks that come through the phone's PDA interface. They also revealed several buffer overflow vulnerabilities (e.g., SMIL in the MMS message) in Windows CE phones and proposed solutions such as boundary checks and MMSC sanitization [93]. Guo et al. [94] suggested some security solutions such as smart-phone hardening, telecom-side and Internet-side defenses. Radmilo et al. [95] demonstrated an unique attack from the Internet that utilizes MMS vulnerabilities to exhaust cell phones' batteries. They identified two vulnerable components in the network and proposed mitigation strategies.

The problem of quarantining cell-phone malware, to our best knowledge, has not been adequately addressed. Currently, the best defense mirrors the strategy against computer viruses with the inception of security patches for cell-phones. However, it is challenging for users to acquire signature files in a timely manner. Some recent work proposes more active ways to defend against the threats. For example, Bose et al. [96] proposed an algorithm to automatically identify compromised phones based on user interactions and suggest a proactive containment framework to quarantine those suspected devices. Cheng et al. [97] designed a collaborative virus detection and alert system named SmartSiren for securing smartphones. These solutions, however, rely on network or some external agents to throttle worm spreads. We notice that countermeasures deployed on phone devices are more effective both in detecting malware and in preventing malicious messages from entering the network in an early stage. On this side, Mulliner et al. [92] adopt a labelling technique to protect a phone against cross-service attacks from the phone's PDA interface. However, they did not provide solutions against automated phone malware [93].

As there is a wide variety of malware threatening cell phones in different aspects, a comprehensive and comparative study on their attacking strategies and propagation behavior is very necessary for designing effective countermeasures against them. A solution is incomplete if it considers only a subset of malware features. Towards these design goals, we make the following contributions in designing a device-level countermeasure against cell-phone worms.

- First, we categorize cell-phone malware based on their infection vectors, malicious behavior and attacking strategies. We analyze two major attacking strategies from software perspective which could be adopted by malware to exploit security vulnerabilities in phone devices. Based on our real implementations on these attacks, we put forward challenging issues on combating existing and upcoming threats to cell phones. Our categorizations and implementations are among the first known complete work in the literature.

- Second, we propose an access control–based mechanism which controls accesses to key system resources through enforcing a customized access control policy in cell-phone environment. More importantly, we are the first to show how to integrate realtime support and the proposed security protection into a commodity version of platform OS (for Linux-based phones) and our solution also applies to smartphones based on Symbian and Mac OS.

- Third, by realizing the limitations of the access control–based mechanism in containing more elaborated cell-phone malware, we further propose a more comprehensive and intelligent protection scheme which identifies and blocks cell-phone malware using artificial intelligence (AI) techniques such as Graphic Turing test (GTT). Through challenging a resource-requesting application, our scheme differentiates whether this application is legitimate (user-initiated) or illegal (malware-initiated). We show how to instantiate and integrate GTT through detailed implementations on Symbian-base phones. To our knowledge, we are the first to introduce AI techniques into cell-phone devices for combating malware attacks.

- Forth, different from previous work which only show proof-of-concept attacks and preventions, we provide detailed implementations and thorough evaluations of our protection schemes on two major types of cell-phone platforms: Symbian and Linux OS. The experimental results including benchmark and performance data demonstrate that both our solutions are effective in identifying and blocking malware in a wide variety of smartphone devices. Also, both of them are lightweight and can be conveniently deployed on existing cell-phone hardware.

5.2 Categorizing Cell-phone Malware

We first do a comparative study on existing and upcoming security threats to cell-phone devices. Our study focuses on three different aspects of cell-phone malware, namely *infection vectors*, *attacking strategies*, and *malicious behavior*.

5.2.1 Infection Vectors

Infection vectors are vehicles that malware exploit to propagate within cellular networks. Existing infection vectors of cell-phone malware include Blue-tooth, MMS/SMS messaging, and Wi-Fi communications. In the future, any communication channel between a cell phone and other devices in a network can be exploited by a malware for spreading malicious programs.

Bluetooth technique helps cell phones set up *localized* wireless connections for message dissemination. Specifically, two Bluetooth–enabled phones that are in close proximity with each other initiates a symmetric key authentication process named *paring* to set up a secure channel, through which they exchange data files. Most traditional cell-phone malware such as Skull [34], Cabir [35] and Mabir [84] exploit this vector to propagate. As Bluetooth has only a short coverage from 10 meters (Class II) to 100 meters (Class I), and phone users have to manually accept such connections (i.e., worm spreads rely on user operations), Bluetooth is not the most effective vehicle in propagating malware among cell phones.

Unlike Bluetooth, text-based Short Message Service (SMS) and its enhanced version – Multimedia Message Service (MMS) are provided by 2G/3G wireless networks. Exploiting MMS/SMS message deliveries, a cell-phone malware can include its malicious code either in a message attachment or in the message content and propagate throughout the network. In this mode, the malware travels through the network infrastructure and reaches remote targets, hence its spread is more system-wide and faster than that of Bluetooth-based malware. Recently emerging malware (e.g., CommWarrior [33]) adopts this effective vehicle in addition to traditional infection vectors such as Bluetooth.

Nowadays, new smartphones (e.g., Nokia 9300) also use 802.11b wireless interface to communicate with each other or visit the Internet. These devices typically incorporate cell phone and PDA functionality. Malware can exploit Wi-Fi communications and infect their PDA and phone modules, and then launch cross-service attacks [92, 98] to their cellular network.

5.2.2 Attacking Strategies

From software perspective, we study two basic forms of attacking strategies adopted by existing malware to compromise a cell-phone device. The key difference between these two strategies is whether a malware creates a new process in order to launch its attack.

Attacking Strategy I: *In this case, a malware always creates a new process within the software platform to execute its malicious code and compromise a cell phone.* Note that this new process typically has a new security context (in terms of access control mechanism) which is different from that of the existing running process (legitimate application), e.g., a different type or domain. As the security context of an application is typically determined based on its authenticity and integrity information, a malicious code cannot provide these to be verified by the platform OS. Most existing malware adopt this strategy because people usually have limited accesses to the implementation details of phone platform OS (e.g., Symbian, Windows Mobile) and the internal application framework (e.g., messaging and phone calls). In most cases, OS such as Symbian are only open to phone manufacturers. People cannot easily change and recompile a kernel as they do in Linux; user-level programs in Symbian phones cannot directly invoke system calls. Instead, they have to invoke related framework APIs (provided by manufacturers) and register themselves within the platform in order to be executed. Most cell-phone malware exploit this feature and launch attacks through legally installed user applications. For example, a Symbian malware [84, 35, 34] includes its malicious code in an SIS installation file [1] and attaches it to an infection vector (e.g., a MMS message). Once a recipient activates this SIS file, the malicious program is installed and executed, incurring serious attacks to the phone, e.g., the program could invoke messaging APIs to deliver numerous malicious messages, or secretly collect user information and send to a malicious server. In addition, a malware in Linux-OS–based phone can also adopt this strategy to launch attacks. A figure in [99] demonstrates our implementation of attack strategy I in Motorola A1200 [100]. In a running messaging application named *qtmail*, when a user opens an malicious email/message attachment, a new sub-process is forked and it invokes an *exec* function to switch to a malicious program which launches the email/messaging attack. This malicious program (named *attack* in the figure) has a new security context.

Attacking Strategy II: *In this case, a malware does not create a new process in the phone. Instead, it hijacks an existing cell-phone application (e.g., messaging process) to execute its malicious code within a legitimate security domain.* These attacks usually happen in cell phones with open platform OS and application frameworks, e.g., Linux-based phones. Automated malware adopt this strategy through exploiting software vulnerabilities such as buffer-overrun to launch attacks. Note that the malicious code to be executed when an application is hijacked can be either pre-injected to the existing framework (e.g., in APIs) or directly loaded into memory in real time (e.g., through messaging). Compared with strategy I, this attacking method is more elaborated because it simply utilizes the security context of the hijacked process to execute the malicious code, hence it is more challenging for a malware detector/blocker. In addition, it is more disastrous to cell phones because it does not require human operations to trigger the attack program. Mulliner et al. [93] revealed software vulnerabilities in MMS messaging and demonstrated a proof-of-concept attack in Windows CE-based phones. As the market share of Linux–based smartphones increases (reaches 25% in global and over 30% in Asia during 2007 [101]), we believe that more automated malware will appear in Linux platforms. A figure in [99] demonstrates our implementation of attack strategy II in Motorola A1200. A worm hijacks a legitimate messaging application named *qtmail* and frequently accesses messaging APIs to generate malicious messages. Identifying such a hijacked process is difficult because the

[1] The format of an user-initiated installation file for distributing Symbian applications.

security context of *qtmail* remains valid.

5.2.3 Malicious Behavior

After compromising a cell phone, a malware executes its malicious code to continue propagation and bring various damages to the device. Typically, a MMS/SMS-based malware scans the phone address book and randomly chooses some members to launch attacks through sending malicious messages to these new targets. A Bluetooth–based malware takes control of a victim phone's Bluetooth interface and continuously scans for other Bluetooth-enabled phones within its range. Once a new target is detected, the worm inter-connects two devices and transfers the malicious message to the target, which gets infected and becomes a new attack source.

Besides the above propagation behavior, a malware can also make various damages to a compromised phone. For example, user privacy information such as friends' names and contacts can be secretly collected by a malware (e.g., a Trojan spy named Flexispy [85]) and delivered to an external malicious server; delivering numerous malicious messages can quickly deplete a phone's battery power; malicious program can also crash the running applications of a phone and erase important data in the device. Our implementation [99] demonstrates the behavior of an external attack server, which keeps receiving private user information and malicious messaging requests (e.g., *sendmail*) from a malware residing in a compromised Motorola A1200 phone.

5.3 Overview of Countermeasures

Based on the comparative study, we focus our work on designing system-level countermeasures to detect and block cell-phone malware. We consider a general *attack model* in which a malware (e.g., Commwarrior [33]) adopts MMS/SMS or Bluetooth as its major infection vector and executes its malicious code on a compromised phone to exploit new targets from the address book and disclose user privacy information to an external server. This attack model is representative to most existing and upcoming cell-phone malware.

In the following discussions, we assume that a malware always launch attacks from application level, i.e., they could compromise software for phone applications such as email and MMS/SMS messaging. However, they cannot break the kernel. We note that there are a few techniques which can be applied to cell-phone platform and help prevent malware's kernel-hacking, for example, using integrity measurement architecture (IMA) [102, 103] for identifying modifications on Linux kernel modules, and using virtualization techniques [104] for isolating kernel-level attacks. These countermeasures are not the focus of this dissertation.

5.3.1 Architecture and Components

To achieve the goal of combating cell-phone malware, our system-level solution should first be *real-time* so that it is able to detect new and polymorphic malware instead of merely known attacks. Besides, it should be *automated* and *self-healing*, such that cell phones are able to autonomously recover from infection. Our defense should also be lightweight in that its computational overhead and battery consumption on phone devices are minimal.

Our countermeasure consists of two schemes. Specifically, we design an access control–based protection scheme to defend against malware which adopt attacking strategy I (i.e., a malware launches attacks through running malicious processes with new security context). Considering such a defense is not capable of identifying more elaborated malware which adopt attacking strategy II (i.e., a malware hijacks an existing phone application to execute malicious code within a legitimate security domain), we adopt artificial intelligence techniques such as Graphic Turing test (GTT) to provide more secure and intelligent authentication against both forms of malware attacks. Figure 6.1 illustrates a high-level architecture of the countermeasure.

Our access control–based protection scheme prevents key system resources in a phone from being accessed by illegal processes. It consists of two components: *defining policy for protecting key system resources* and *authenticating resource access via system calls*, while the latter depends on the former. Our GTT-based authentication scheme employs artificial intelligence techniques in a phone device to differentiate a hijacked process from a normally running process. It also con-

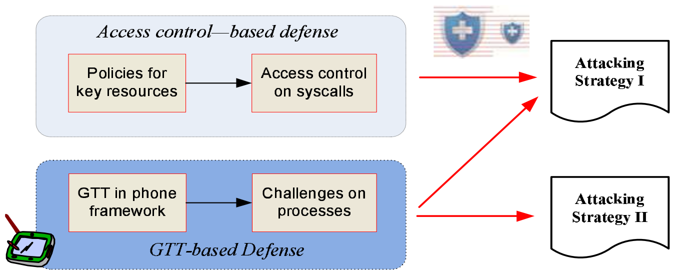


Figure 5.1. Architecture of device-level countermeasure.

sists of two phases: *embedding GTT in phone framework* and *challenging resource-requesting processes*. Both schemes help users identify and block known/unknown malware attacks and recover their phones in an automated and self-healing way.

5.3.2 Access control–based Protection

We first investigate the MMS/SMS messaging service, a major infection vector for malware attacks. In order to deliver a message to the recipient, a normal messaging process in the phone invokes a sequence of key system calls to access important system resources (e.g., file, socket, and modem device) and acquire related system services. For example, in a Linux-based phone (e.g., Motorola A1200 and E680), to search for the email address or the phone number of a recipient, a messaging process named *mmsclient* invokes *open("address_book", O_RDONLY)* to access the contact list in the phone address book; to deliver the SMS/MMS message, *mmsclient* calls *fd=open("/dev/ttyS0", O_RDWR)* to open the modem device [2] (i.e., serial device *ttyS*0) and invokes *write(fd, message, length)* to send a composed message *message* to the modem, which eventually processes the message and transmits it to the air interface; to send an email messaging through Wi-Fi interface, a process named *smtpclient* invokes *socket(AF_INET, SOCK_STREAM)* to create a stream socket through which it communicates with a SMTP server. We notice that these key system calls are important monitoring and authentication points in the kernel, because no matter what a malicious application could be, its final goal is to invoke these system calls as a normal process does and gain accesses to important system resources and launch malicious attacks. Therefore, to examine

[2] Modem device is a separate phone unit, which contains radio frequency (RF) and base-band components, low-level coding-decoding software and wireless protocol stacks.

the difference between a normal application and a malicious process, we aim at examining the security attributes of the on-going process at these checking points.

Traditional cell-phone OS have built in discretionary access control (DAC) mechanisms such as capability list and access control list. For example, Symbian OS 9.0 uses a capability-based framework [105], where 40% of its APIs are protected with capabilities. To use a service, an application either has to pass certain tests and be signed against a certificate (Symbian signing), or is installed by a signed installation package containing the required list of capabilities. However, the fundamental weakness in DAC model is that the ability to grant and use access creates a big security hole, where malicious process can get control of important system resources. For example, in native Linux system, the *owner-group-world* mode specifies the permissions to a file in filesystem. An owner of the file can grant permissions to others. However, as other users always access the file with programs in the system, the behavior of the programs cannot be justified. Thus malware can get the permissions if they are invoked by authorized users. Therefore, we design our access control–based protection scheme based on an embedded kernel level mandatory access control (MAC) mechanism, which labels each program according to its required permissions such that an individual user cannot change the label. There are different MAC policy models in PC platform, e.g., Biba [106], and Clark-Wilson [107].

We adopt a defense model that is similar to Security Enhanced Linux (SELinux) [108] to achieve strong protection in cell-phone devices. Based upon the principle of least privilege [109], SELinux employs Linux Security Modules (LSM) inside Linux kernel to implement strong MAC. This clearly boosts the security level of a traditional phone OS, which itself is only based on DAC. SELinux associates labels (security contexts) of the form *user:role:type* to all subjects (processes) and objects (files/directories, programs, sockets, etc.). Within a security context, the type attribute represents the type of the subject or the object, e.g., *sshd_t* and *syslogd_t*. Instead of directly associating a *user* with a *type*, SELinux associates a *user* with a *role* and a *role* with a set of *type*s. The *role* merely simplifies the management of users. This means that access control in SELinux is primarily enforced via so-called Type-Enforcement [110] policy model.

In our defense model, permissions that a subject can have are defined according to object classes, e.g., *file:*{*read, write, append, lock*}, and *netif:*{*tcp send, tcp recv*}. A typical policy rule is to allow a type of subjects to have some permissions on another type objects. For example, the following rules define two types *qtmail_t* and *qtaddressbook_t* within a cell phone: any subject of *qtmail_t* can have all permissions to any object of *qtaddressbook_t*; any subject of *user_t* only has the permission of *getattr*, e.g., can see the file with *ls* command. We note that without defining other policy rules on *qtaddressbook_t*, it can only be read and written by *qtmail_t* subjects, i.e., we only allow processes with type *qtmail_t* (e.g., a normal email messaging process named *qtmail*) to have full permissions on the resource of address book. We adopt similar policy rules for access control on other key system resources (e.g., modem device and socket) in cell phones. Such a defense model can be flexibly configured to achieve fundamental security goals such as limiting raw access to private user data in cell phones, protecting platform OS and system file integrity, confining process privileges, and separating processes and domains.

```
type qtmail_t;
type qtaddressbook_t;
allow qtmail_t qtaddressbook_t:file *;
allow user_t qtaddressbook_t:file getattr;
```

The above security policy can be enforced through Linux Security Module (LSM), which is a kernel module that has been included in Linux kernel 2.6. With LSM enabled in a cell-phone OS, a set of hooks are placed in Linux kernel such that whenever sensitive system call functions (defined as the monitoring points for cell phones) are invoked by user level processes, the corresponding hooks start the authentication and authorization by checking with the LSM. Within LSM, a security server implements the decision logic based on its TE policy model, where the binary policy is pre-loaded. Thus, through authenticating these system calls, our defense denies unauthorized accesses to key system resources in phone devices and identifies those malicious resource-requesting processes. By defining appropriate policy rules, our scheme can support the Mobile Phone Reference Architecture recently specified by Trusted Computing Group (TCG) [111, 112]. Specifically, a cell-phone is a multi-stakeholder computing environment, where resources from different stakeholders (typically, manufacturer, network provider, service provider, and the user) need to be strongly isolated, and the communication between them have to be tightly controlled. Our access control–based scheme can meet these requirements.

5.3.3 GTT-based Protection

At the system level, we want to design malware containment techniques to prevent unauthorized messages from leaving compromised phones or from entering those benign cell phones. Specifically, a sender has strong motivation not to let her phone be compromised; even if it is compromised, she wishes to block the malware in the earliest stage so that her phone will not become an attacking source in the system. To identify and throttle malware within cell phones devices, we first need to differentiate a malware-initiated messaging process from a normal user-initiated messaging process. This is relatively easy when a malware always creates an unauthorized (or unregistered) process to invoke system calls or messaging framework APIs to execute the malicious program, because such malicious process typically has new security context which can be identified according to predefined security policies. However, in the case when a malware launches attacks through a registered application (e.g., the email messaging process *qtmail* in a Linux-based phone), this becomes a challenging task because the malicious process obtains the same security context as the registered application (i.e., *qtmail*), which is legitimate to access sensitive resources. This case is demonstrated in [99].

To detect this malware behavior, one approach is to use content-based filtering [113]. For example, those messages that are delivered with .SIS attachments and attractive titles are more suspicious than others. However, as we have experienced from combating PC email spam, this involves complex learning engines and does not work well either in resource-constraint cell-phone devices. Another approach is to let a user manually confirm every message that is leaving her phone. Specifically, when sending a message to the air interface, the messaging framework requires the user to enter YES/NO using the keypad (or touch-screen) to confirm her message delivery. This of course helps block some simple attacks. However, more elaborated malware could compromise the application-layer keypad (or touch-screen) driver and inject a false input to circumvent this single-touch protection. Moreover, a malware can easily crash the phone by generating numerous messaging requests to deplete system resources.

We adopt artificial intelligence (AI) techniques to solve this challenging issue. Our idea is to rely on AI techniques to authenticate the messaging behavior of an ongoing process and identify malware attacks. Specifically, our countermeasure involves executing a Graphic Turing test (GTT) before a messaging framework eventually delivers a message to the air interface. GTT has a nice feature that human can always pass a Turing test while an automated malware cannot. In this case, even if an elaborated malware has compromised the application-layer keypad driver, it is still unable to figure out the correct answer of a GTT to fabricate a story that the message is initiated by a human being. We note that this countermeasure identifies cell-phone malware adopting both forms of attacking strategies, because it does not simply verify the security attributes of the on-going process; instead, it differentiates a malicious message originator from a normal user through authenticating their genuine natures (AI techniques are capable of doing this). Moreover, this approach helps a user detect a wide variety of malware, including new malware that are unknown to security vendors (e.g., zero-day or polymorphic ones), so that the user can immediately block the malware within the phone device and apply a security patch to recover it whenever the patch is available.

Although our GTT-based protection scheme requires an user operation during the message delivery, the rate of normal MMS/MMS messaging (on the order of 0$\sim$10.07 messages/hour [114]) is unlikely to be very frequent for every user due to the limited resources of dedicated signalling channels, for example, Standalone Dedicated Control Channels (SDCCHs) in GSM [115], configured for SMS messaging service in Base Stations. Moreover, in the worst case, this operation takes only several seconds to complete the test, which nearly has no impact on the time response when compared to the time a user takes to compose a message. We note that a user is motivated to take such a test because she hopes to prevent her phone from sending numerous malicious messages which will cause privacy leakage, extremely high service charges and quick depletion of battery power[3]. In addition,

[3]Transmission via the radio interface consumes most power in messaging due to its signaling interactions with network. GTT blocks malicious transmission and saves power.

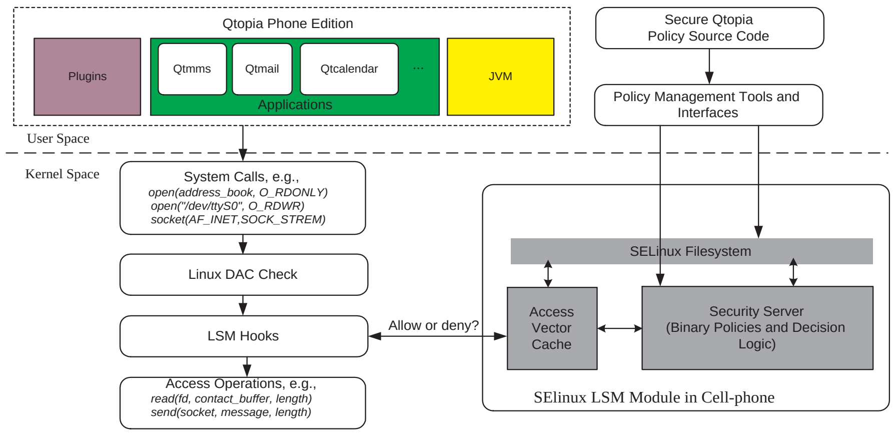

Qtopia Phone Edition
Plugins
Qtmms
Qtmail
Qtcalendar
...
Applications
JVM
Secure Qtopia Policy Source Code
Policy Management Tools and Interfaces
User Space
Kernel Space
System Calls, e.g.,
open(address_book, O_RDONLY)
open("/dev/ttyS0", O_RDWR)
socket(AF_INET,SOCK_STREM)
Linux DAC Check
LSM Hooks
Allow or deny?
Access Operations, e.g.,
read(fd, contact_buffer, length)
send(socket, message, length)
SELinux Filesystem
Access Vector Cache
Security Server (Binary Policies and Decision Logic)
SElinux LSM Module in Cell-phone

Figure 5.2. Cell-phone malware containment with access control on key system resources

a phone user can temporarily turn off the functionality of GTT or switch it to a semi-protection mode, in which the messaging framework randomly executes GTTs to challenge the message sender. This is especially useful when the user decides to deliver a large amount of messages, for example, during holidays or birthdays. Of course, a tradeoff between user convenience and security always exists.

Based on the above design principles, we study practical problems of how to instantiate a GTT and embed it into a wide variety of smart phones which run different platform OS, and where to perform GTT security challenge to message initiators. We also study whether there are alternatives for GTT and how to prevent GTT from being circumvented by a more elaborated malware. We discuss these implementation issues in Section 5.4.

5.4 Implementation

5.4.1 Access control–based Protection

5.4.1.1 Porting SELinux on TI OMAP-5912OSK Board

To our best knowledge, there is no existing Linux phone on the market that includes SELinux [4]. We ported the NSA SELinux [116] into TI OMAP-5912OSK [117],

[4]Motorola A1200 implements some kind of access control called Motoac. However the source code and policy file are not public yet.

which is a generic smartphone development board with ARM9 processor operating at 192 MHz, 32 Mb RAM and 32 Mb Flash. The OMAP-5912OSK originally comes with a MontaVista Linux 2.4 binary (including a kernel patch of realtime support for cell phone), with which we found that porting SELinux is difficult. Our porting includes three main tasks: building a SELinux-enabled Linux kernel, building a SELinux-aware root filesystem, and porting SELinux libraries and tools. The first task is the easiest one as Linux 2.6 has already included NSA SELinux Module and it has incorporated the realtime feature since kernel version 2.6.18. We use the standard Linux 2.6.20 source code and customize it for the OMAP-5912OSK hardware. Thus, we completely replace the original MontaVista Linux 2.4 on the board.

We use *Buildroot* [118] to build a root filesystem. *Buildroot* integrates basic packages for a root filesystem for embedded systems. Most importantly, it seamlessly integrates *Busybox*, which combines tiny versions of many common UNIX utilities into a single small executable, thus reducing memory and disk spaces for embedded environments. We then ported the NSA stable SELinux release [116] onto the board, which includes *libsepol-1.16.6*, *checkpolicy-1.34.3*, *libselinux-1.34.13*, *libsemanage-1.10.5*, and *policycoreutils-1.34.11*. All these are libraries and tools to compile and manage policy, configure SELinux, and provide interfaces to security-aware applications. The porting also includes *libpython-2.4*, which is required by SELinux tools. To test our porting, we run the NSA example policy [116] on the OMAP-5912OSK board.

Some significant efforts of our porting lie in the difference between *uClibc* [118] and *glibc*. The NSA SELinux libraries and tools are built on *glibc*, while our development filesystem on OMAP-5912OSK is built with *uClibc*, which is optimized for embedded devices. As a result, a lot of configurations and patches of SELinux have to be changed according to the different implementations of C libraries. The details are included in our ported package which is available for public downloading.

5.4.1.2 Cell-phone Malware Containment with SELinux

Figure 5.2 shows a complete flow of applying access control on key system resources to contain cell-phone malware. Here we explain in details how this scheme works in Qtopia phone edition 4.2 [119], the major application platform for Linux-based phones such as Motorola A1200, Sony Mylo and many others [120]. Recall that our strategy is to authenticate accesses to key resources such as user address book, modem devices, and Wi-Fi interfaces on a mobile device. Therefore, we focus our control on the system calls that are invoked towards these resources, such as *open("address_book", O_RDONLY)*, *open("/dev/ttyS0", O_RDWR)*, and *write(fd, message, length)*. Fortunately, hooks can be defined by LSM in most places where we want to control system file openings and device accesses. Therefore the main task in this scheme is to (1) define SELinux policy rules to limit the permissions that a program can have, (2) label corresponding programs to confine the access to these resources, and (3) limit domain transitions from any other domains to those that are allowed to access these resources. We explain these steps as follows, respectively.

The fundamental goal of our access control–based scheme is to only allow legitimate accesses to key resources. Therefore, SELinux policies should permit legitimate access requests while deny others. The key problem is to identify what kind of privileges are required for each program or subject, such as to satisfy the least privilege principle. One feature of SELinux policy is that all *allow* rules in a *policy.conf* are positive; that is, a single rule always adds some permissions to a subject type. Therefore to prevent any illegal subject type having unexpected permissions, we define "private" types of our target resource objects such that they are only visible to subject types in the same *domain*. Here by domain we mean all trusted subjects that are allowed to access a target object. For example, in Qtopia platform, Qt applications and plugins (e.g., *qtmail*, *qtmms*) are typically trusted to the address book object. Thus permissions from them to read/write user address book are allowed, while those from others are denied.

To provide private object types, we leverage the recent SELinux Loadable Policy Module, which offers a flexible way to create self-contained policy modules that are linked to an existing policy without re-compiling the whole policy source each time. A policy module can define private types and attributes and then define interfaces so that other modules can use them, thus enabling type encapsulation and controlled communication between the subjects whose types are defined in different modules. After a module is created, it can be loaded to the kernel during runtime using SELinux policy management tools. A significant benefit with this approach is that, as a mobile phone is a multi-stakeholder computing environment, each stakeholder can deploy its own security policy without any interference with other stakeholders. As a result, the integrity and confidentiality of its data and applications deployed on the phone is preserved. In addition, a loadable policy module can be updated during runtime and over-the-air, e.g., with regard to the policy change of a stakeholder.

To correctly enforce SELinux policies and confine the behavior of applications, another key issue is subject and object labelling. As SELinux is label-based access control, assigning appropriate labels to target subjects and objects, and controlling the permissions to change these labels are critical. Particularly, in cell-phone environment we have to solve two problems: (1) we need to label the key resources and trusted subjects with appropriate labels on a cell-phone OS, and (2) besides the policy rules to enable trusted subjects' permissions on target resources, we need to define rules for permission *labelfrom* and *labelto* to subjects and key resources such that only legitimate processes can get these permissions and re-label them. Significantly different from DAC which an object owner determines the permissions to access an object, SELinux controls which domain can have these permissions such as to enable the permissions based on corresponding types.

Domain transition is another mechanism to confine permissions of a process. Particularly, when a process executes another program, the new process first has the same security context as the calling process. If domain transition rules are defined, the new process can be transited to a new type so that some sensitive permissions can be given to it. The principle behind domain transition in SELinux is that, by defining a particular program with which a privileged domain can be entered, some permissions can be tightly controlled since only this program can have these permissions. Thus, policy rules are required to specify what kind of subjects can invoke these privileged programs.

5.4.1.3 Defining Policy for Securing Messaging

Next, we use a concrete example to demonstrate how cell-phone malware can be effectively contained with the above techniques. Specifically, we show how to construct a policy module for securing messaging services in Linux-based phones which adopt Qtopia Phone 4.2 as the application platform. Qtopia provides a number of integrated messaging applications (e.g., SMS, MMS, email client) to phone users. Here we use *qtmail* – the email messaging application as an example. We note that policy rules can be defined for other applications within this module or with other modules. As discussed in Section 5.2, a malware attacks the cell phone either by executing a malicious process from the framework of *qtmail*, or by directly hijacking *qtmail* (we show detailed implementations in Section 5.5.1). Note that both scenarios eventually result in malicious exploits on system resources. In this example, we examine malware exploits on user address book. We show how to define SELinux policy rules and explain from the domain transition perspective why such policy rules help protect the Qtopia messaging application against a malware which adopts attacking strategy I. We first construct the following policy module.

```
policy_module(qtmail, 1.0)
require {
  type base_t;
  type sysadm_t;
}
type qtmail_t;
type qtaddressbook_t;
allow qtmail_t qtaddressbook_t:file *;
allow base_t qtaddressbook_t:file getattr;
allow sysadm_t qtmail_t:{dir file}
  {relabelto relabelfrom};
allow sysadm_t qtaddressbook_t:{dir file}
  {relabelto relabelfrom};
allow type_transition base_t qtmail_t:
  process qtmail_t;
allow qtmail_t qtmail_t:file entrypoint;
allow base_t qtmail_t:process transition;
```

As previously discussed, we define two types: *qtmail_t* for *Qtmail* client application, and *qtaddressbook_t* for user email address book object file. Two allow rules specify that only *qtmail_t* subject can read and write the address book object, while any other subjects (*base_t*) can have the *getattr* permission of it, e.g., by using *ls* command. Any other accesses to the address book object are denied by SELinux security module. The next two allow rules state that only *sysadm_t* subjects can label objects (including program file and data file) to/from *qtmail_t* and *qtaddressbook_t*; that is, only a system administrative program can make a program file to *qtmail_t* such as to read object file of *qtaddressbook_t*. This prevents the possibility that a malicious program can relabel an arbitrary program to *qtmail_t* which then can read the address book file. The last three allow rules define the domain transition that when a *qtmail_t* program file is executed, the new process is transited from *base_t* to *qtmail_t* automatically. This ensures that a *qtmail_t* process only can be created by executing a *qtmail_t* program file.

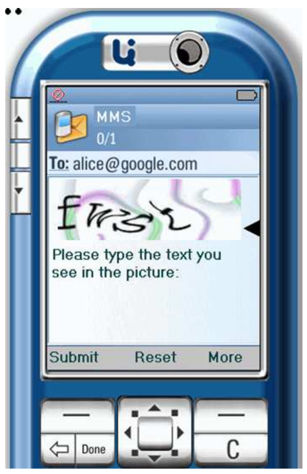
Hence, by enforcing this policy module, we can ensure that only *qtmail_t* process can read and write user address book, a *qtmail_t* process can only be created by executing a *qtmail_t* program file, and only system administrative process can label a program file to type *qtmail_t*. Therefore, without an explicit administrative change, any malicious process launched by a malware cannot be labelled as *qtmail_t* and its access to the user address book object will be denied.

For attack II, as the legitimate *qtmail* process is hijacked, which is already labelled with *qtmail_t* when it is launched, our access control–based scheme cannot block its malicious access to the user address book. Therefore we need a more intelligent scheme as described in next sections.

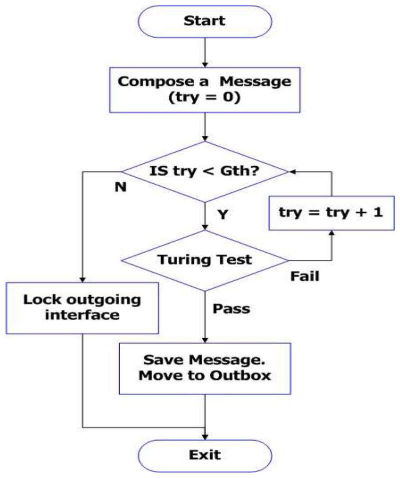
5.4.2 Graphic Turing Test on Cell Phones

AI techniques such as GTT helps people differentiate normal human behavior from automated attacks, hence they are popularly adopted by network servers for filtering automated actions from spammers or worms, e.g., in securing email account registration [121] and defending against DoS attacks [122]. In our case, we design a GTT-based mechanism which helps a cell phone identify a worm-initiated messaging process and take further protection against it. This mechanism is equally effective in combating Bluetooth-based worms. However, here we use MMS-based worms as our example.

Instantiating GTT We choose to incorporate a visual CAPTCHA (Completely Automated Public Turing test to Tell Computers and Human Apart) [123] test into a mandatory point of each message delivery. CAPTCHA is a program that can generate and grade tests that most humans can solve, but automated programs such as worms cannot. Thus, for a normal cell-phone user, she simply needs to pass an easy visual test before sending her newly composed message to the output interface (usually a buffer named Outbox). However, a worm most probably fails this authentication and all its unauthorized out-going messages will be eliminated from the cell phone. One realization of CAPTCHA is GIMPY, which concatenates an arbitrary sequences of letters to form a word and renders a distorted image of the sequence. GIMPY relies on the fact that humans can read the words within the distorted image and existing automated software cannot. A user authenticates herself by entering an ASCII text in the same sequence of letters as what appears in the image. Fig.5.3(a) shows an example of GIMPY test in a cell phone.



(a) GIMPY test in cell phone



(b) Flow of protection

Figure 5.3. GTT-based protection on cell phone

In choosing an appropriate GTT instance, there exists a tradeoff between the instance's complexity (i.e., security) and its convenience to the user. A GIMPY test with less number of characters are more likely to be broken by an intelligent worm. However, normally users do not want to spend much time in answering the challenges. In this dissertation, although for demonstration purpose we use GIMPY, our scheme conveniently supports an upgrade to a more secure instance of GTTs, e.g., Animal-PIX, which asks a user to select between a set of pre-defined animals. Unlike Gimpy, Animal-PIX has never been reported broken even in the PC world, and it helps reduce a user response time through her immediate observations. We note that there is an advantage that strongly favors our applications of CAPTCHA on cell-phones, because *a worm is unlikely to install complex machine-learning tools such as neural networks in a resource-constraint phone device (the latest model has a 600 MHZ CPU and hundreds of Mbytes memory) to build classifiers and recognize the texts and figures, although there had been several successful cases against simple CHAPTCAs [124, 125] in PC systems.* We give further discussions on this issue in Section 5.5. On the other hand, since it is unrealistic to assume a system with perfect CAPTCHA, we will consider in the modeling (Appendix B) a non-zero probability that GTT can be accidently circumvented by a worm's random guesses in this context. Another alternative in CAPTCHA is to apply biometric techniques on cell phones, for example, a user can conveniently uses her finger-prints [126] to confirm each message delivery. However, this assumes that a worm cannot access the user's finger-print. This can be achieved by applying the access control mechanism in platform OS security [127].

Implementing GTT on Symbian Phones The next issue is where to incorporate a GIMPY test. In our Symbian phone experiment, we embed a GIMPY test within the messaging framework (Nokia S60 and Sony-Ericsson UIQ 3 SDKs), whose APIs must be invoked each time before an application eventually accesses the hardware interface (i.e., modem) to transmit a message. A modem is responsible for data coding/decoding and protocol stacks in a cell phone, and any messaging request (e.g., MMS and SMS) and the message content should be finally directed to this device in order to be transmitted to the air interface (see AT commands defined in GSM protocol 0707, 0705 [38]). Specifically, phone vendors such as Nokia and Sony-Ericsson can integrate the test into a Symbian messaging library function named $CMmsClientMtm :: SendL()$. This function is invoked each time when a user application initializes a communication entity named $iMmsMtm$ and composes a new MMS message (Figure 5.4). A closer look into a messaging procedure shows that $iMmsMtm$ first sets the message content (including recipient address, attachment, etc.) and moves the message to a temporary buffer named Outbox (through calling $CMmsClientMtm :: SendL()$). At the same time, $iMmsMtm$ starts a timer by calling $wait \rightarrow start()$ and has the system scheduler deal with the final message delivery to the modem interface. Through augmenting the library function $CMmsClientMtm :: SendL()$ into $CMmsClientMtm :: NewSendL()$

(Figure 5.5), we place the GIMPY test at the point right after the message content has been built and just before it is sent to the buffer of the modem interface. Thus, only authorized messages that pass the authentications are transferred to the modem. Because Gimpy test is embedded in the messaging APIs, a user can conveniently update it through downloading a new function library from a phone vendor's website. For example, she can upgrade the software to execute another form of GTT such as Animal-PIX [123].

```
/* inside user or malware application */
...
iMmsMtm = (CMmsClientMtm *)NewMtmL(KTypeMMS);
iMmsMtm->CreateMessageL(serviceId, aMsgId);
iMmsMtm->SetMessagePriority(priNormal);
iMmsMtm->AddAddressL(aRealAddress);
iMmsMtm->SetSubjectL(aSubject);
iMmsMtm->CreateAttachmentL(aAttId, aFileName) ;
...
iMmsMtm->SetMessageRootL(aAttId);
iMmsMtm->entry()->setVisible(true);
iMmsMtm->SaveMessageL(aMsgId);
iMmsMtm→ NewSendL();        /* invokes GTT */
wait.Start();
CActiveSceduler::Start(op);
...
```

Figure 5.4. A Symbian messaging application

Addressing challenges from Open-source Devices The above solution, however, only considers the case when the worm leverages the existing messaging framework to launch attacks and to propagate. An application-layer worm in a Linux-based smartphone can exploit the openness of the communication software and launch attacks that directly deliver malicious data to the modem interface, without following the existing messaging framework. For example, in the above example, a worm may bypass *mmsclient* and directly invoke *write(fd, message, length)* to transfer its malicious message. Although such attacks have not yet been reported from existing smartphones, the potential threats are approaching as Linux-based phones with open source are taking more and more market shares (reached 15% in 2007). We know that Linux 2.6 has incorporated SELinux into the kernel and enables privileged users to conveniently define access-control policies for

```
/* rewriting lib funcion */
CMMSClientMtm::NewSendL() {
  InsertGimpyPicL(RandomNo);
  EchoL(KGTTMessage);
  ...
  while(++gttry <= NumTry){
    GTTDialog->ExecuteLD(R_GTT_DIALOG);
    iGTTString->Des() = gttinput;
    if(!CompGimpy(RandomNo, iGTTString)){
      EchoL(KCompareSucceed);
      CMMSClientMtm::SendL();  /*orignal lib function*/
      break;}
    ...}
  if(gttry == NumTry) BlockSendMtm(this);
  ...
}
```

Figure 5.5. Integrating GTT into Symbian library function

system resources on a per-process basis. Thus, any illegal process directly accessing phone resources such as the address-book file and the modem from outside the messaging framework does not follow the defined policies and will be immediately identified and blocked. However, as we mentioned earlier, in the case when a legitimate process such as *mmsclient* is hijacked, its security context remains valid and unauthorized accesses to the resources can still be granted according to the security policy.

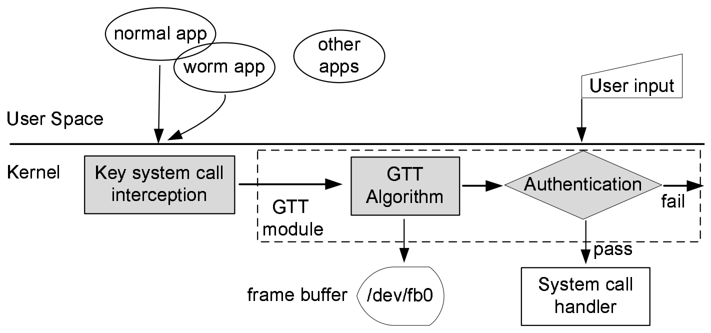


Figure 5.6. Implementing GTT in Linux-based smartphones

We propose a more effective solution – a kernel-module based approach to solve this challenging issue. Fig.5.6 shows the defense architecture which consists of two parts: *key system-call interceptions* and a *GTT kernel module*. Specifically, we monitor and intercept key system calls which may be invoked by both a normal application or a worm process. Once such a system call is caught, a kernel-level GTT module presents a challenge to the call initiator.

A normal MMS process and a worm process both need to invoke the system call *write(fd, message, length)* to eventually transfer the message content to the modem, although they have different initiators. To intercept this key system call, we place a hook in *write*() and enforce a simple parameter checking each time when it is invoked. Specifically, we only aim at catching a system call *write*() with the first argument being "/dev/ttyS0" (modem interface) and the second argument begins with "AT+CMGS" (AT command type for messaging request in GSM 0705 [38]). Upon detecting the first messaging request, a GTT module is immediately loaded by the kernel to provide authentication. Note that this protection resides in the kernel memory until it is explicitly removed or when there has been no messaging requests for a sufficient long period of time. This GTT module has two components: the *GTT algorithm* and the *authentication part*. The GTT algorithm is similar as we introduced above except that its rendering function writes the output directly to the *frame-buffer driver* instead of the phone's application view. In Linux Kernel 2.6, the frame-buffer device represents the memory buffer of the video hardware and allows user programs to access the graphics hardware through a well-defined interface [128]. Specifically, the frame buffer can be treated as a memory device (similar as */dev/mem*), and each user program can use *mmap*() to map the frame buffer into its own address space and seek the desired position to read/write the memory content. Considering that in Linux smartphones, application-level GTT could be compromised by worms, directly writing GTT output to the frame-buffer device provides a more secure way of presenting an authentic/unfalsified graphical challenge to the phone user. The authentication part simply verifies the user's GTT response and invokes the original system call (i.e., *write*()) if her response is correct. Note that we do not need to protect the user input (response) at the application level, because even if the worm successfully compromises the user input, it is still unable to figure out the correct answer of a GTT challenge [5]. As a result, such attack will be identified and the original system call will not be invoked.

[5]Nor do we need to worry about a Denial-of-Service attack in which a worm destroys the frame-buffer display of GTT, because an anomaly will be detected once a normal user fails to reply the GTT challenge.

Identifying and Blocking Worms Deploying GTT in a cell phone not only prevents unauthorized messages from leaving the terminal but also helps a user identify on-going attacks in the phone in a *automated* and *real-time* manner. Considering that a human could inadvertently enter wrong answers to a challenge, we devise a worm-blocking mechanism, as shown in Fig.5.3(b). Basically, a terminal defines a threshold G_{th} as the allowed number of GTT failures during a messaging attempt. This threshold is set by the user and stored as a phone parameter. The value of G_{th} should allow human mistakes such as typos while never let a worm succeed in guessing the GTT answer. Also, anyone who fails a GTT will be given a brand new challenge. In this way, a worm can be clearly identified after continuously failing GTTs for G_{th} times. Once the phone has detected the worm's existence, it should temporarily block the outgoing messaging interface. For example, in Symbian OS it blocks the *ClientMtm* entity and hence related AT commands for sending messages to the modem. Note that this protection does not influence the interface for incoming messages. The phone user will be notified of the worm attack and the on-going defense, so that she can disinfect this blocked phone by installing a security patch or sending it for professional maintenance.

5.4.3 Comparing Device-level Countermeasures

We have introduced two system-level countermeasures to throttle cell-phone malware, namely the access-control–based and the GTT-based protection scheme. These two schemes have some common features. First, none of them rely on known signatures and manual updates to contain the malware, hence they essentially provide realtime protection to phone devices. Second, these two schemes both achieve automated malware detection and isolation, based on which automated online security update becomes very practical.

We also note that there are significant differences between the two schemes. First, the former identifies malware according to pre-defined process capabilities on accessing key system resources, while the latter identifies malware by telling the nature of a resource-requesting process, i.e., whether the initiator is human or malware. From this sense, the latter is more comprehensive and intelligent, as we have shown in combating malware adopting both forms of attacking strategies.

However, the GTT-based scheme has false positives due to its nature of using artificial intelligence, while the access control–based scheme does not. We evaluate the GTT selection criteria and show how to reduce false positives in Section 5.5. In real life, choosing which protection scheme heavily depends on some practical factors in the context of cell-phones, for example, the platform OS currently used (i.e., the complexity of defense implementation), and the current hardware configuration of the cell phone (i.e., whether malware can launch some complex tools to do more than random-guessing towards a challenge).

5.5 Evaluations of Effectiveness

We implemented our system-level defenses on real devices, including Symbian-based and Linux-based phones, which represent the majority of existing smartphone products. In this section, we evaluate the feasibility and effectiveness of these defenses.

5.5.1 Experimental Settings

Symbian-based smartphone We conducted our experiments on Symbian-based phones because of their popularity in current smartphone market. Symbian OS is a proprietary operating system designed for mobile devices, with associated libraries, user interface frameworks and common tools, provided by Symbian Ltd. Based on the standard Symbian OS, phone manufacturers such as Nokia and Sony-Ericsson have developed their own versions of Symbain C++ SDKs for application developers, for example, Nokia platform uses Series 60 (S60) SDKs, and Sony-Ericsson platform uses UIQ SDKs.

In our tests, we chose Nokia S60 phones (Nokia 3230, E62) and Sony-Ericsson UIQ phones (Sony-Ericsson P900) as the target devices. Each of these smartphones has a 32-bit RISC CPU (ARM-9 series, 123-220MHZ) and an expandable memory of 10-32 MB. All these devices support Bluetooth and MMS/SMS functionalities. We adopted Metrowerks CodeWarrior V3.1.1 (for Symbian OS v9.0) as the integrated development environment (IDE). To test the compatibility of our GTT-based defense with most smartphone vendors, we implemented the scheme using two major Symbian C++ SDKs, Nokia S60 2rd Edition and Sony-Ericsson UIQ 2.1, respectively. Using the CodeWarrior compiler, object codes of malware and defenses were first built into PC executables and tested in a Windows-based Symbian phone emulator named WINSCW. Finally, these programs were built into target executables for the ARM-9 hardware platform and deployed to real phone devices.

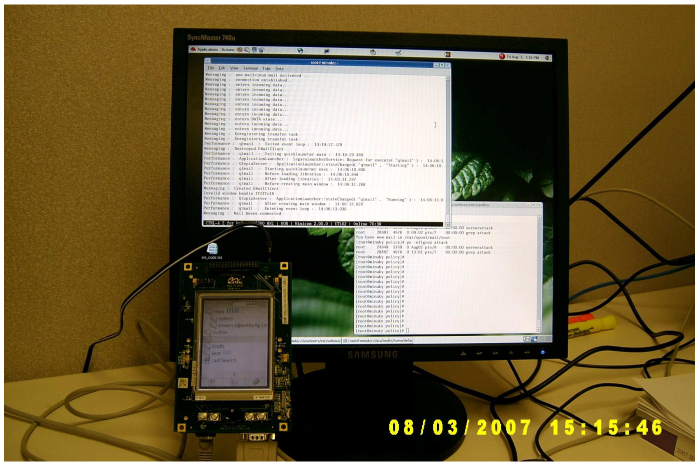


Figure 5.7. OMAP-5912OSK as a generic development platform for both WinCE and Linux-based smartphones; in our experiments, we use Linux kernel 2.6.20.4, Qtopia Phone Edition 4.2

Linux-based Smartphone Linux-based smartphones have had substantial growth in recent years. In our tests, we chose OMAP-5912OSK [117] as the development board for Linux-based phones. OMAP-5912OSK is a generic hardware platform for developing smartphones based on both Windows CE and Linux OS. The OMAP board includes TI processor ARM926TEJ operating at 192MHZ, 32 MByte RAM, Mistral's Q-VGA/touchscreen, ethernet/USB/serial port. Original OMAP board ships with MontaVisa Linux OS for OSK (kernel 2.4). We customized it to kernel version 2.6.20.4 for deploying our access control–based defense. To develop phone applications based on the hardware, we chose Trolltech [119] Qtopia Phone Edition 4.2 as the application platform. Qtopia is currently running in a wide variety of Linux-based phones [120] (e.g., Motorola A1200, OpenMoko), it provides a complete set of C++ SDKs, user-friendly tools and APIs to application developers. Our source code of malware and defenses was first built into PC executables and tested in a Linux-based emulator. Finally, these programs were cross-compiled into target executables for the ARM-9 processor and deployed to the OMAP board. Figure 5.7 shows our smartphone testbed using OMAP-5912OSK board.

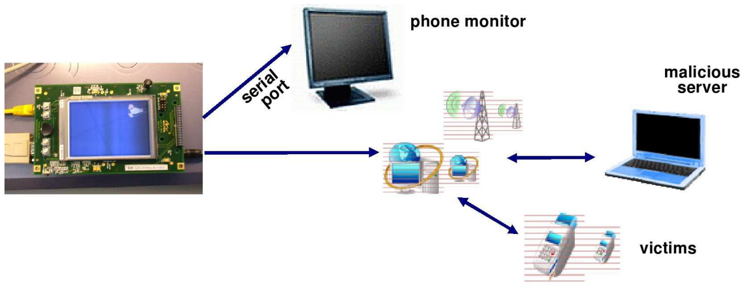
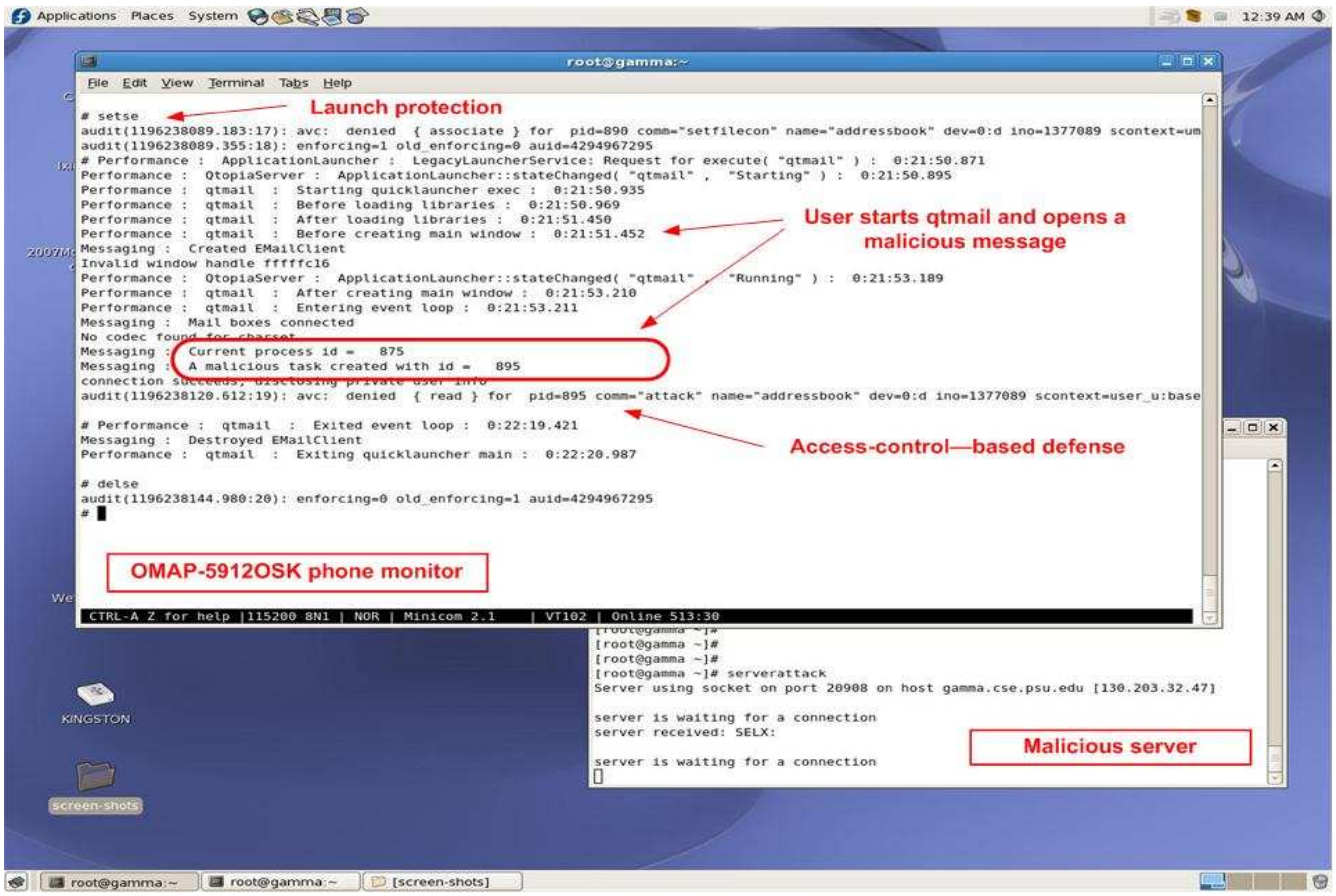


Figure 5.8. Configuration of cell-phone experiments (using OMAP-5912OSK platform as an example).


Applications Places System
12:39 AM
root@gamma:~
File Edit View Terminal Tabs Help
Launch protection
User starts qtmail and opens a malicious message
Access-control—based defense
OMAP-5912OSK phone monitor
Malicious server
KINGSTON
screen-shots
root@gamma:~
root@gamma:~
[screen-shots]

```
# setse
audit(1196238089.183:17): avc:  denied  { associate } for  pid=890 comm="setfilecon" name="addressbook" dev=0:d ino=1377089 scontext=um
audit(1196238089.355:18): enforcing=1 old_enforcing=0 auid=4294967295
# Performance :  ApplicationLauncher :  LegacyLauncherService: Request for execute( "qtmail" ) :  0:21:50.871
Performance :  QtopiaServer :  ApplicationLauncher::stateChanged( "qtmail" ,  "Starting" ) :  0:21:50.895
Performance :  qtmail  :  Starting quicklauncher exec :  0:21:50.935
Performance :  qtmail  :  Before loading libraries :  0:21:50.969
Performance :  qtmail  :  After loading libraries :  0:21:51.450
Performance :  qtmail  :  Before creating main window :  0:21:51.452
Messaging :  Created EMailClient
Invalid window handle fffffc16
Performance :  QtopiaServer :  ApplicationLauncher::stateChanged( "qtmail" ,  "Running" ) :  0:21:53.189
Performance :  qtmail  :  After creating main window :  0:21:53.210
Performance :  qtmail  :  Entering event loop :  0:21:53.211
Messaging :  Mail boxes connected
No codec found for charset
Messaging :  Current process id =   875
Messaging :  A malicious task created with id =   895
connection succeeds, disclosing private user info
audit(1196238120.612:19): avc:  denied  { read } for  pid=895 comm="attack" name="addressbook" dev=0:d ino=1377089 scontext=user_u:base

# Performance :  qtmail  :  Exited event loop :  0:22:19.421
Messaging :  Destroyed EMailClient
Performance :  qtmail  :  Exiting quicklauncher main :  0:22:20.987

# delse
audit(1196238144.980:20): enforcing=0 old_enforcing=1 auid=4294967295
# 
CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.1     | VT102 | Online 513:30
```

```
[root@gamma ~]#
[root@gamma ~]#
[root@gamma ~]#
[root@gamma ~]# serverattack
Server using socket on port 20908 on host gamma.cse.psu.edu [130.203.32.47]

server is waiting for a connection
server received: SELX:

server is waiting for a connection
```

Figure 5.9. Access-control–based scheme identifies and blocks a malware which adopts attacking strategy I; Phone device: OMAP-5912OSK, Platform OS: Linux 2.6 kernel, Phone application framework: Qtopia 4.2

Experiment Configuration Figure 5.8 shows the configuration of our smartphone experiments. An administrator can control an OMAP board and log its events through an external *Minicom* terminal (serial port). Each OMAP board connects to a standard modem device through which it communicates with other smartphones within 2G/3G cellular networks. In addition, each board can access the Internet through its on-board network interface (using either wired or Wi-Fi communications). We also implemented an external malicious server, which establishes connections (SSL) with the on-board malware and receives private user contact information stealthily gathered from the OMAP board. Therefore, besides the malware attacks inside the board, the malicious server itself can exploit disclosed user information and launch automated messaging attacks to vulnerable phones by executing its own messaging service such as *sendmail*.

We implemented two representative malware: Cabir [35] and CommWarrior [33] in both Linux and Symbian-based smartphones. These malware adopt attacking strategy I. For instance, when a Symbian phone user unwittingly opens an attachment in a message titled “Breaking News”, a CommWarrior process is started and it randomly retrieves some recipients from the phone addressbook as its new victim targets and secretly delivers (in background) malicious messages with the similar attractive titles to them. For each of these messages, the malware specifies the MIME type as “application/vnd.symbian.install” and attaches a malicious SIS file which contains a complete installation of CommWarrior. The detailed messaging flow invoked by CommWarrior in Symbian platform has been shown in Figure 5.4. In our tests, we set the malware’s messaging speed as once in every $20 \sim 30$ *seconds*, which is much more frequent than users’ normal messaging rate (on the order of $0 \sim 10$ $messages/hour$). Note that although these malware were only reported from Symbian-OS, in our implementations, we extended the above malware cases to the environment of Linux-based phones which are receiving more and more attentions from malware writers.

In addition, we implemented an automated malware which adopts attacking strategy II in compromising smartphone devices. Using the OMAP-5912OSK device as an example, when a user just clicks on (or highlights) a newly arrived message entitled “Breaking News”, the running process named *qtmail* which belongs to Qtopia’s email messaging framework invokes function *EmailHandler::mailRead(Email**

mail) to process the message and interpret it on the phone screen. However, there is a buffer-overrun vulnerability inside this function (i.e., no boundary check for its temporary storage of the message content). The malware exploits this vulnerability and hijacks the program flow by replacing the return address of this function. Now that the execution of *qtmail* has been redirected to the injected code[6], which starts a malware timer by executing *QTimer::start()* and associates the timer event to an attack function named *attackLoop()*. This *attackLoop()* results in the similar attacks as we mentioned above. Similarly, for MMS messaging, the malware seeks buffer-overrun vulnerability in a function named *MMSHandler::MMSRead()*, which is invoked by the MMS messaging process *qtmms*. We note that in these attack cases, the malware does not include any attachment to the message and wait for user's reaction to install itself on the phone.

5.5.2 Access control–based Defense

We launched both forms of attacks. The Attack I is successfully prevented by the access control–based scheme with customized policy, as Figure 5.9 shows. The *avc: denied* exception message shows that the system call *open("address_book", O_RDONLY)* is rejected due to the security context of the malicious process. As expected, attack type II cannot be prevented by this scheme, we do not show the snapshot due to space limit here.

As SELinux LSM inserts hooks and checks access control policies in many system calls, it introduces overhead to main primitive functions and inter-process communications (IPCs). We studied the performance of our access control–based scheme with microbenchmark to investigate the overhead for various low-level system operations such as process, file, and socket accesses in phone devices. Our test includes two policies: our simplified policy for cell-phone environment, and the original NSA example policy. We compared the results with those measured without SELinux involved (baseline).

Our benchmark tests were performed with the LMbench 3 suites [129]. Table 5.1 shows the measurements in microseconds. For each operation, the SELinux overhead consists of all required permission checks. For example, the null I/O

[6]This malicious code could either exist in the message content or be pre-injected into some library files such as libqtmail.so and libqtmms.so by the malware.

Benchmark	Baseline	Our Policy	%	NSA Policy	%
null I/O	22.5	31.4	39	29.5	31
stat	48.4	66.5	37	67.9	40
open/close	1113	1179	6	1277	14
0KB create	2985.1	3257.3	9	3436.4	15
0KB delete	3174.6	3268.0	3	3546.1	11
fork	4169	4270	2	4281	2
exec	18K	19K	5	19K	5
sh	78K	83K	6	83K	6
pipe	264.2	319.6	20	322.8	22
AF_UNIX	460	529	15	511	11
UDP	574.1	648.3	13	817.9	12
TCP	771.4	858.6	11	1044	35

Table 5.1. Benchmark results in OMAP-5912OSK. All measurements are in microseconds. Smaller is better.

benchmark measures the average time for a one-byte read from */dev/zero* and a one-byte write to */dev/null*. The overhead consists of permission revalidation for both of them. The stat operation overhead includes permission checks for searching a path and obtaining file attributes. The open/close measures the time to open a file for reading and close it after that. The 0KB create and delete measure the time to create and delete a zero-length file. The fork, exec, and sh benchmarks measure the times to create process in different ways. The other tests measure round-trip latency for different IPC mechanisms.

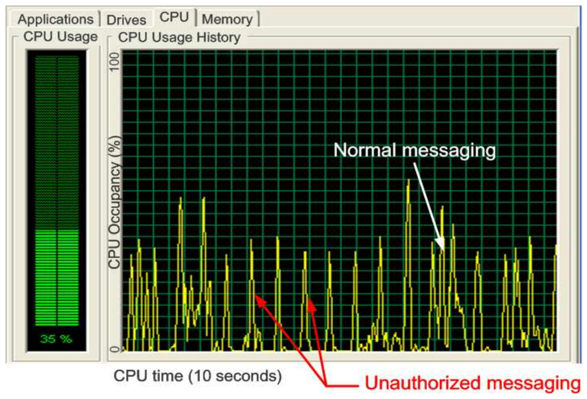
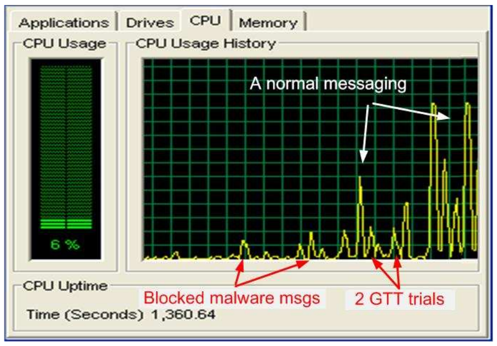


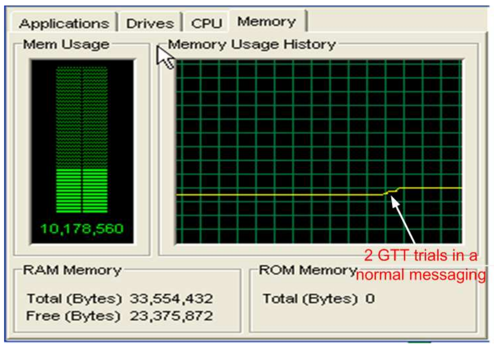
Figure 5.10. Nokia 3230 compromised by malware adopting attacking strategy II; average CPU occupancy exceeds 35%

Both sets of measurements show the same trend that security checks in null I/O and stat operations introduce more overhead percentage than others. However, the total time of security checks is quite small in these operations, i.e., less then 20 microseconds. Typical file related operations such as open/close and create/delete, and process related operations such as fork, exec, and sh, also have very small overhead. The average is around 5%. Another observation with our benchmark results is that, although the NSA example policy introduces more overhead due to its large policy size, in most operations it has the same order of overhead as our simplified policy. The reason is that, although NSA example policy includes many types and rules, in a cell-phone environment, typically there are much fewer processes (and thus types) concurrently running in the system than that in a PC environment. Therefore, the variance of permission check overhead with different policies in the phone context is not significant. As a summary, using Linux kernel 2.6 as the platform OS, our access control-based defense is lightweight when compared with the baseline benchmark.



Time (Seconds) 1,360.64

(a) GTT blocks malware and reduces average CPU occupancy to 6%



Total (Bytes) 33,554,432
Free (Bytes) 23,375,872
Total (Bytes) 0

(b) GTT's memory usage is as low as 3.13%

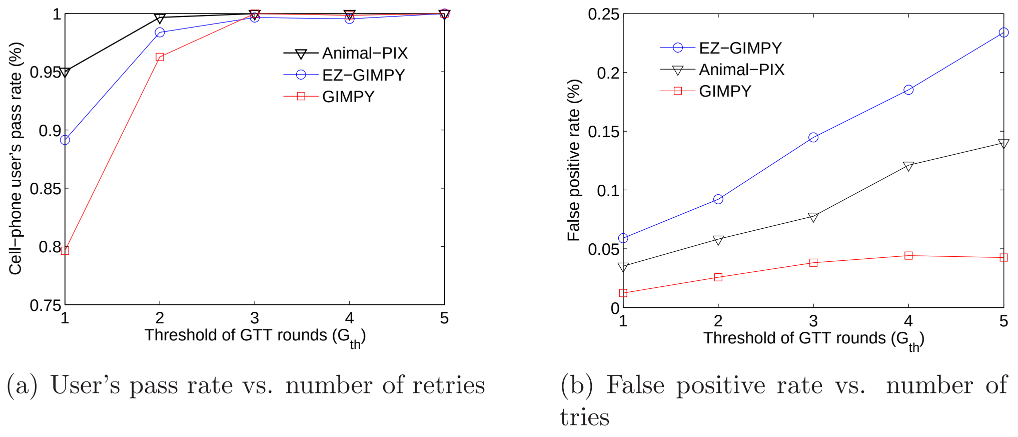
Figure 5.11. Effectiveness of GTT protection in Symbian phones (Nokia 3230)

5.5.3 GTT-based Defense

To evaluate the effectiveness of the GTT-based protection scheme and test its overhead (e.g., CPU occupancy and memory usage) in real phone environments, we launched both forms of attacks on Symbian-based and Linux-based smartphones. The experiment result in Figure 5.10 demonstrates the system resource usages of a Nokia 3230 phone which is compromised by a malware adopting attacking strategy II to cause the same damages as CommWarrior [33] does. We can see that the malware automatically generates a malicious message in every 20 seconds (according to the timer it starts). This results in 35% CPU occupancy by average. We note that there is difference in CPU usage between a normal messaging and a malware-initiated messaging, because the latter is executed in the background hence does not involve Graphic User Interactions (GUIs) during its messaging process.

Figure 5.11 shows the effectiveness and the overhead of the GTT-based scheme. We chose EZ-Gimpy [130], a CAPTCHA which is currently used by Yahoo! to screen out bots. In this instance of CAPTCHA, a message initiator is challenged with a single word (5 characters) image, which has been distorted, and a cluttered and textured background has been added to confuse OCR (optical character recognition) software. As aforementioned, we embedded GTT into the Symbian library function $CMMSClientMtm :: SendL()$, which is currently used in the messaging framework of both Nokia S60 phones and Sony-Ericsson UIQ phones. We chose the threshold $G_{th} = 3$, which means that a message sender can fail up to three different GTT challenges before the out-going messaging interface gets blocked. Figure 5.11 demonstrates a typical case in which our GTT-based scheme successfully identifies and blocks the malware and prevents misuses of system resources. We can see that our protection scheme reduces the average CPU occupancy to 6% or below. In addition, two GTT trials in the test (here the user inadvertently fails the first EZ-GIMY but passes the second) incur low overhead on CPU and memory usage. Specifically, the maximum CPU occupancy is 5% and it only happens within less than one second, the memory space required for an EZ-GIMPY is in order of hundreds of KBytes (as shown in Figure 5.11(b)). Another important thing is that each EZ-Gimpy takes a user 1.5∼2.0 seconds (note that the time unit is one second in Figure 5.11) to complete, hence it has a good time response compared with the time that a user spends in composing a message.

To test different instances of CAPTCHA and evaluate their feasibility and performance in cell-phone environment, we conducted three major forms of tests in Symbian platform. These tests includes (1) EZ-GIMPY [130], in which phone users are presented with an image of a single word. This image has distorted, cluttered, and textured background. (2) GIMPY [?], which is a more difficult variant of a word-based CAPTCHA. For example, 5 words are presented to a user in distortion and clutter similar to EZ-GIMPY. The words are also overlapped, providing a CAPTCHA test that can be challenging for human in some cases. The user is required to name 3 of 5 words in the image in order to pass the test. (3) Animal-PIX [123]: an instance of graphic-based CAPTCHA, in which a phone user is presented with a distorted image of one out of 12 different animals (e.g., bear, cow, pig, etc.). The user is asked to select from the set of predefined animals.



(a) User's pass rate vs. number of retries

(b) False positive rate vs. number of retries

Figure 5.12. Conducting different forms of CAPTCHA tests in cell-phone devices

In our experiments, we adjusted the GTT threshold G_{th} (the allowed number of GTT trials during each messaging). 30 cell-phone users were given the above three types of CAPTCHAs and we recorded the pass rate (%), which is the percentage of users who have eventually passed a GTT (with given G_{th} opportunities). We also conducted the same set of tests on an automated malware, which typically uses a random-guess strategy to answer the challenge. Note that although some simple CAPTCHAs have been broken by automated tools using machine learning technologies [124, 125], these tools require high performance machines to conduct complex tasks such as training and learning. This is extremely difficult for a malware which hides in a resource-constraint cell phone. However, we assume a malware has some degree of intelligence in our tests. For example, in EZ-GIMPY, the malware is able to refer to a dictionary to associate characters in the single word and improve its random guess. Each of our tests with users and malware took 20 runs and we took the mean value. We set the confidence interval to 95%. Figure 5.12 shows the results of our GTT experiments on cell phones.

Figure 5.12(a) demonstrates that cell-phone users have different success rates on different forms of GTTs. Besides the difficulties of the GTTs themselves, convenience of user input from keypad (or touch-screens) can also influence the pass rates. Specifically, graphic-based Animal-PIX is relatively convenient for a phone user to choose one among twelve animals (using navigation keys or touchscreen); EZ-GIMPY requires only a single word input from the user, hence it is easy to achieve using the keypad; GIMPY test requires multiple word inputs from the user, hence it is relatively inconvenient. More convenient input always helps to improve the pass rate. The figure also indicates that when the GTT threshold G_{th} increases, a cell-phone user has a higher chance to pass the GTT. However, this also brings higher false positives to the GTTs, as shown in Figure 5.12(b). Figure 5.12(b) shows that an automated malware has certain chance to succeed in three GTTs. EZ-GIMPY is the easiest for malware because it requires only a single word and the malware can improve its guess with the help of a dictionary; Animal-PIX could be broken when the malware happens to choose the correct one out of twelve predefined animal images; GIMPY is the hardest for the malware because it requires multiple words and relatively complex inputs. Because we need to control the false positives, we conclude that choosing $G_{th} = 2 \sim 3$ is a good strategy for executing GTTs in cell-phone devices. We also believe Animal-PIX is a preferable instance of GTT for securing cell phones, although its overhead on memory usage could be a little bit higher than the other two instances.

Chapter 6

Designing Systematic Countermeasures against Cell-phone Worms

6.1 Introduction

Cell phones are increasingly becoming attractive targets of various worms, which not only cause the leakage of user privacy, extra service charges and depletion of battery power, but also introduce malicious traffic into the network infrastructure. Examples of the most notorious threats to cell phones include the Skull [34], Cabir [35] and Mabir [84] worms targeting at Symbian operating systems. The research organization IDC estimates that by 2008 the market for mobile security software will grow yearly by 70%. Therefore, *both cell-phone designers and network service providers* must employ appropriate countermeasures against such threats.

The problem of quarantining cell-phone worms in a systematic manner, to our best knowledge, has not been adequately addressed. Currently, the best defense mirrors the strategy against computer viruses with the inception of security patches for cell-phones. However, it is challenging for users to acquire worm signature files in a timely manner. Some recent work proposes more active ways to defend against the worms. At the terminal-device level, Mulliner et al. [92] adopted a labeling technique to protect the phone against cross-service attacks from the phone's PDA interface; from the network level, Bose et al. [96] proposed an algorithm to automatically identify compromised phones based on user interactions and suggested a proactive worm containment framework to quarantine those suspected devices. Cheng et al. [97] designed a collaborative virus detection and alert system named SmartSiren for securing smartphones. These solutions, however, are still not complete because they do not leverage collaborations between terminal devices and networks to throttle worm spreads in a systematic way. Moreover, some solutions require deploying external proxies to monitor cell-phone groups [97], which bring extra overhead and alter network architecture.

Contributions: As cell-phone worms threaten both user devices and networks, a solution is partial if it only considers either of them. We aim to design effective yet efficient countermeasures for protecting both sides. We make the following contributions in this dissertation.

- First, after carefully examining the design challenges, we propose a systematic approach, which consists of comprehensive defense mechanisms at both terminal-device level and network level. The design goal of our terminal-level defense is to prevent a compromised phone from stealthily sending MMS messages to others as well as to prevent a user from accepting messages that cannot be authenticated. For the devices that are lack of terminal-side protection, we resort to the network to identify compromised phones based on their misbehavior, and further push the proper security patches to these phones to disinfect them.

- Second, to realize our design goals, we adopt AI techniques such as *Graphic Turing test* (GTT) to throttle automatic worm transmissions and cryptographic techniques such as *identity-based signature* (IBS) to authenticate received messages. Adopting the GTT technique (Chapter 5), our approach is unique in that it is capable of combating elaborated cell-phone worms which cannot be detected by existing access control mechanisms (e.g., Symbian platform security [127] and SELinux). Moreover, our techniques can be implemented in a wide variety of smartphone devices. Based on our cell-phone solutions, we also devise a *push-based secure patching* scheme to rescue cell phones that are identified as compromised.

- Third, we adopt discrete time and recursive analysis to model the epidemic behavior of cell phone worms that exploit both MMS and Bluetooth for spreading. Generic worm models such as [131] are not accurate enough because they assume homogenous network topologies, and existing work either only experimentally demonstrates such system-wide threat [88] or only analyzes mobile worms which exploits Bluetooth communications [87]. Therefore, these results do not directly apply to a complete modeling which mathematically depicts worm propagation for cellular systems.

- Forth, we evaluate the effectiveness of the proposed defenses through real experiments, extensive simulations and theoretical analysis. The results clearly demonstrate that our systematic solution effectively helps mobile communication systems retain a low worm infection rate (less than 3% within 30 hours) even under severe worm attacks.

6.2 System Model

Network Model We consider a network model which supports both MMS and Bluetooth, because these two popular services consist of the major infection vehicles of cell-phone worms. Similar as PC email systems, each MMS-enabled phone keeps a contact list in its address book. Each record in this contact list contains a phone number or an e-mail address of a valid MMS user. An MMS network can be modeled as a directed graph $G(V, E)$, in which each node $n \in V$ represents an active cell phone (i.e., it has been attached to the data network) and each edge $e_{ab} \in E$ from node a to b indicates that a has b in its contact list. We define *out-degree* of a node in the directed graph G as the size of its contact list. On the other hand, Bluetooth communications add variations to this network graph G. A Bluetooth–enabled cell phone i moves randomly and it scans within a short range any Bluetooth–enabled neighbors. Once a neighbor j has been explored, i may have a connection with j (sometimes a manual pairing is needed on their first meet). This connection breaks as they move out of range. During this process, node i essentially varies its edges in the network graph. This variation is determined by multiple factors such as node speed, node density and radio coverage.

Attack Model and Security States An MMS-based worm starts by attacking some initial targets (namely hit-list) in the network. Each infected phone scans its contact list and randomly picks up some members to deliver malicious messages. A susceptible receiver will most likely trust the incoming message (due to its attractive title or familiar source). She activates the attached file and unwittingly gets her phone infected. This process repeats in the network. A Bluetooth–based worm takes control of a victim phone's Bluetooth interface and continuously scans for other Bluetooth-enabled phones within its range. Once a new target is detected, the worm inter-connects two devices and transfers the malicious message to the target, which gets infected and becomes a new attack source.

An active phone has two security states: *susceptible* and *infected*. A susceptible phone is not protected against security threats and it gets infected when exposed to a specific worm, such as CommWarrior [33]. This infected phone returns to susceptible state when its user launches some protection (e.g., applying a patch against CommWarrior from Symantec) on the device. Note that it remains susceptible to other type of worms.

Security Assumptions To our knowledge, all existing attack cases reported from Symbian, WinCE, and Linux-based smartphones, are generated by application-level worms [83, 91]. Using Symbian as an example, a worm uses SIS files [1] as the infection media. Once a recipient activates the message attachment (i.e., the malicious SIS file), a worm application is started such that it compromises the messaging framework (by invoking Symbian APIs) and begins the propagation cycle. Similar processes happen in WinCE and Linux-based devices. The common fact here is that phone OSes are usually more reliable and secure than those popular phone applications in front of worm exploits. Implementation details of Symbian and WinCE OS are not revealed to 3rd-party developers, and security mechanisms such as Symbian Platform Security [127] and SELinux have added enough difficulties to OS-level malware designers. In our work, we focus on addressing different forms of application-level attacks on cell phones. In addition, we assume that the platform OS is robust enough so that a phone will not easily get crashed by application-level attacks.

[1]SIS is the format of an user-initiated installation file to distribute Symbian applications.

6.3 System Overview

Our goal is to effectively combat cell-phone worms. To achieve this, our defense should be *real-time* so that it is able to detect new and polymorphic worms instead of merely known attacks. Besides, our defense should be *automated* and *self-healing*, such that cell phones are able to autonomously recover from infection. Clearly, this requires collaborations between terminal and network. Our defense should also be lightweight in that its computational overhead and battery consumption on terminal devices are affordable.

6.3.1 Architecture and Components

We propose a systematic countermeasure against cell-phone worms. This approach consists of a terminal-level defense and a network-level defense. Fig.6.1 illustrates the high-level architecture of the defense framework.

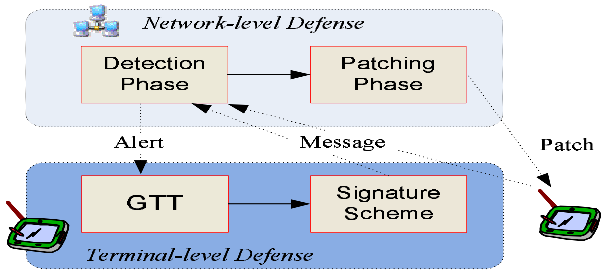


Figure 6.1. Architecture of proposed systematic countermeasure.

Our terminal-level defense has two components: *sender-side worm detection and blocking* and *send-and-receiver-side (two-side) message authentication*. In sender-side protection, a new technique based on Graphic-Turing-test (GTT) detects and blocks worms within phone devices, thus preventing unauthorized message deliveries; in two-side protection, a signature scheme helps both terminal sides verify the authenticity of messages. Our terminal-level defense achieves real-time protection because GTT identifies both known and any new forms of worm attacks using artificial intelligence (AI) techniques.

Leveraging terminal-level protection, our network-level defense consists of a detection phase and a patching phase. In the detection phase, the network monitors terminals' messaging behavior and detects anomalies using different strategies; in the patching phase, the network initiates an automated and secure patching process on those suspected terminal devices. Through these two phases, infected phones recover in an automated and self-healing way.

6.3.2 Terminal-level Defense

On the terminal side, we want to design techniques to prevent unauthorized messages from leaving compromised phones or from entering benign phones. A sender has strong motivation not to let her phone be compromised. Even if it is compromised, she needs to detect the worm in an earliest stage so that her phone will not become an attacking source in the system. For a recipient, she does not want to accept any unauthorized messages because of the potential attacks and the extra service charges (service providers such as Verizon and Cingular also charge message recipients).

As we mentioned in Chapter 5, we can adopt AI techniques to provide an effective enhancement of the existing platform security (access-control–based mechanisms) at the terminal level. Our device-level defense addresses both forms of attacks. It involves executing a GTT before the messaging framework delivers a message to the air interface. GTT has a nice feature that human can pass the test while automated worms cannot. In this case, even if an elaborated worm has compromised the application-layer keypad driver, it is still unable to figure out the correct answer of a GTT to prove that the message is initiated by a human being.

Although GTT helps block malicious messages within a sender's device, the recipient cannot figure out whether an incoming message has been authorized (i.e., passed GTT) or it is a malicious one sent by a non-protected terminal. We propose that right after completing a GTT, the cell phone generates a lightweight digital signature and attaches it to the out-going message, proving that the message has been authorized by the sender. The recipient simply verifies the signature and decides whether and when to retrieve the incoming message.

6.3.3 Network-level Defense

On the network side, the best strategy is to block malicious messages before they enter the rest part of the network infrastructure. Otherwise, not only those message recipients could be threatened, but also network resources (e.g., radio channels, communication bandwidth) could be greatly wasted. To achieve this goal, a network-level defense should first be able to monitor terminals' messaging behavior and detect any anomalies which reflect on-going attacks on the phones. Once the network identifies compromised phones, it should immediately take countermeasures to disinfect the attack sources, such that their malicious messages will be prevented from entering the network system.

We cannot merely rely on terminal-level defense to block worm messages because some phone devices may not support GTT in the near future. These susceptible devices could be the major sources of attacks that eventually enter the network infrastructure. Existing solutions such as [97] suggest deploying external proxies for monitoring cell phone groups and alerting the threatened devices. However, this brings extra overhead and essentially changes network architecture. More importantly, their approach only quarantines those compromised terminals without further actions such as launching user-side protection or disinfecting them.

Based on the incremental deployment status of GTT on phone devices, we design three strategies for detecting compromised phones from network level. First, when GTT has just been introduced into terminal devices, we propose that the network identifies compromised phones through monitoring users' messaging behavior (e.g., messaging rate). Second, when GTT has been widely deployed in the system, we propose that the network instead of the recipient verifies the message sender's signature. Any worm message that fails the verification is considered as malicious. Third, in an intermediate stage when GTT has been partially deployed, we propose a hybrid strategy, which combines the above two and treats terminal devices differently depending on whether they support GTT protection.

In addition to these online-detection techniques, we further propose a *push-based–patching* mechanism, in which the network disinfects origins of malicious messages through automatically pushing software patches to those compromised terminals. Unlike a *pull-based* solution where users themselves have to manually connect to security vendors and request patches for their phone devices, a push-based approach is automated and it actively eliminates malicious origins from the system in the earliest stage.

6.4 Implementation

6.4.1 Authentication for Both Sides

Based on the device-level protection such as GTT, We further design a signature scheme for sender-side signing and receiver-side verifying authorized messages.

Choosing Signature Algorithm Traditional signature techniques such as RSA and DSS use digital certificates to bind a user's public key with her identity. In cell phone platform, however, we adopt identity-based signature (IBS) [39] as our basic algorithm, in which a user's identifier instead of the digital certificate is used as the public key for signature verification. IBS scheme has an advantage that both parties may verify signatures without prior distribution of keys between individual participants (as required by traditional Public-key schemes [132]). This is extremely useful for a cellular system where pre-distribution of authentication keys is expensive or infeasible. Using the traditional RSA-based approach, a recipient needs to have the public key of the sender; it is neither convenient nor scalable for a sender to include her certificate (typically several-hundred-bytes long) in every out-going message. On the other hand, according to 3GPP TS 23.140 [38], a recipient is notified of the sender's identifier in the initial stage of MMS/SMS messaging. This identifier is in the form of E.164 telephone number (MSISDN) and can be conveniently utilized for verifying the signature of the message. Note that due to the user authentication mechanism in wireless systems, this identifier cannot be faked during messaging (3GPP TS 33.102 [38]). Our experiments in Section 5.5 shows that IBS scheme provides a good time response based on security requirement and existing phone hardware.

Implementing Signature Scheme on Cell Phones Basically, our signature scheme consists of two stages, as illustrated in Figure 6.2. In the initial stage (step $1 \sim 5$), a trusted third party PKG (Private Key Generator) H creates a set of public parameters $params$ for all users and distributes a secret key $SK(s)$ to each registered user. Note that k is a random parameter and $params$ includes H's public key, known crypto-functions and groups of numbers. Q_s is a string generated together with key $SK(s)$ but it is kept in the PKG.

One important issue here is which party can act as a PKG and how to pre-distribute keys. A natural choice is a network component responsible for security data management. For example, in a 3G UMTS system, users' security data are stored/retrieved in/from the Home Location Register/Authentication Center (HLR/AuC). As there are several service providers (SPs), we propose to pre-load their public parameters to cell phones when users open their accounts or let users retrieve these parameters from HLR/AuC when they activate GTT in their phones for the first time. Thus, a user does not have to obtain public parameters from the network on the fly for every message, ensuring service availability and reducing latency. In a very rare case when a service provider decides to update in its public parameters, it broadcasts the changes through the system messages to all registered cell phones. Similarly, a user may choose to pre-store her private key in the local phone or securely retrieve it from HLR/AuC.

The second stage (step $6 \sim 8$) is involved in a messaging process, where a sender S signs the message content using its secret key $SK(s)$, and the message recipient R uses the sender's identifier (phone number) s and the public parameter $params$ to derive string Q_s and then verifies the message signature σ. Here we propose the sender concatenates a single-bit GTT indicator G_{ind} to the message body M. This indicator is set to 1 once GTT has been enabled on the sender's phone and it should be included when the sender generates the message signature σ. Through this indicator the recipient learns whether the message is under GTT protection, and it decides whether or when to retrieve the message from its MMS server. Specifically, for a recipient who performs the verification, any GTT-protected message that passes its verification will be fully trusted and retrieved from the server in a high priority; any GTT-protected message which fails the authentication (either the signature or the indicator G_{ind} is faked by the worm) should be discarded from the server; any unprotected message is usually given a lower receiving priority and phone users are hence encouraged to enable the terminal-level defense.

6.4.2 Countermeasures from the Network Side

Our network-level defense consists of two phases: identifying victim phones and rescuing these phones from the network side.

Identifying Compromised Phones We propose three following strategies for identifying compromised phones at the network level.

Notation:

- H is HLR/AuC, which is used as a PKG
- S is the sender, which originates a message
- R is the recipient, which receivers a message
- $*$ denotes all users in the network
- MK is the master key of the HLR/AuC
- s is the phone number of user S
- G_{ind} is a GTT indicator
- MD is message digest of message M, using MD5 hash function
- $A \rightarrow B : M$ denotes that user A delivers message M to user$B$

A Complete IBS Flow:

1. $H : (MK, params) = Setup(k)$
2. $H \rightarrow * : params$ // *params is known to all users*
3. $S \rightarrow H : s = ID(S)$ // *S asks HLR for its private key*
4. $H : (Q_s, SK(s)) = Keygen(s, MK, params)$
5. $H \rightarrow S : SK(s)$ // *HLR replies S with private key*
6. $S : (\sigma) = Sign(G_{ind}|MD, params, SK(s))$
7. $S \rightarrow R : G_{ind}|M|\sigma$ // *S sends ind., msg, signature to R*
8. $R : Verify(G_{ind}|M, \sigma, s, params)$

Figure 6.2. IBS scheme for both sender and receiver side authentication

Signature-based. A home MMSC adopts the similar method as described in Section 6.4.1 to verify a sender's message signature. Any malicious message that fails the verification should be discarded from the network and the sender is considered compromised. Because the network always authenticates a user before granting her messaging service request ($MM1\_submit.REQ$), a home MMSC acquires the sender's real identity (phone number) and then adopts the signature scheme (see Section 6.4.1) to verify the message before it reaches the rest part of the network. Clearly, this approach requires that GTT be widely deployed on terminal devices. Note that when some messages are left unsigned, a home MMSC is unable to differentiate malicious messages from those legitimate ones.

Threshold-based. A home MMSC monitors terminals' messaging behavior and detects compromised terminals once it has found anomalies. For example, the network side may adopt *messaging rate* as a metric to measure terminal's behavior. A normal user usually delivers MMS/SMS messages at a rate of $0 \sim 10.07$ messages/hour [114]. However, a cell phone compromised by CommWarrior [33] delivers at a rate over $120 \sim 150$ messages/hour (see our experiment in Section 5.5). To measure user i's messaging rate, a home MMSC simply counts within a recent time window ΔT the number of messaging requests ($MM1\_submit.REQ$) it receives from user i (we denote this number as M_i). If this messaging rate exceeds a predefined threshold β_{th}, i.e., we have $\frac{M_i}{\Delta T} - \beta_{th} > \delta$, where δ is an error tolerance, sender i will be considered *suspicious* within this time window. However, we know that a user's messaging rate could be high on some special occasions (e.g., New Year holiday and birthday), which may cause false positives. We adopt a *challenge-based* mechanism to reduce this error. When the home MMSC suspects a terminal's messaging behavior, it immediately sends a CHALLENGE message to the user, urging her either to launch GTT protection on her device or otherwise to manually confirm the normal variation of the messaging rate (e.g., messaging during holidays). Another issue is how to determine this β_{th}. We first provide a general threshold (e.g., $25 \leq \beta_{th} \leq 100$ messages/hour) according to the overall messaging rate. This parameter is then fine-tuned by the network, which divides phone users into different clusters according to their daily or weekly messaging records. Users within the same cluster share a common threshold β_{th}, which is derived from a subset of cluster members who have launched the terminal-level defense. Users' messaging profiles can be stored in HLR so that MMSCs can conveniently retrieve them. The threshold-based method is effective at an early stage when GTT has just been introduced into the network. It requires more network resources to learn individuals' messaging behavior.

Hybrid. This approach is a combination of the above two methods. Specifically, for GTT-enabled phones, a home MMSC adopts the signature-based approach; while for non-GTT phones, MMSC adopts the threshold-based approach to detect compromised terminals. As more and more users enable GTT in their devices, the signature-based approach will gradually become the majority. Moreover, the threshold-based approach encourages users to launch terminal-level defense whenever it has identified anomalies on their devices.

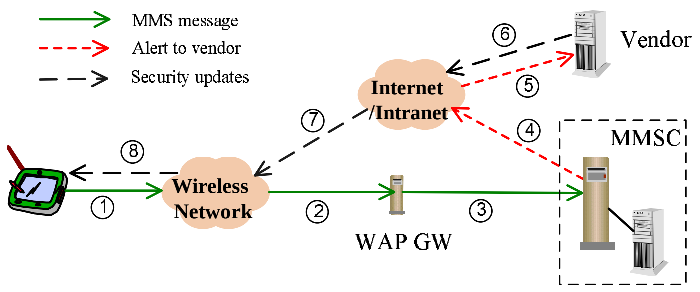


Figure 6.3. A flow of the push-based–patching mechanism. Step 1 $\sim$ 3 a worm message reaches a home MMSC; step 4 $\sim$ 5 the MMSC detects the attack and alerts a security vendor; step 6 $\sim$ 8 the security vendor sends patch updates to an infected phone.

Pushing Patches to Compromised Phones Once a cell phone is suspected being compromised by a cell-phone worm, its MMSC immediately notifies a security vendor (e.g., F-Secure and Symantec) through a direct link or an Internet routing infrastructure. Note that the user's phone number is also included in the notification, so that the vendor knows which cell phone has been suspected and it may deliver the latest security patch(s) (worm-signature–based) to disinfect or immunize the cell phone. Specifically, a security patch contains worm information (e.g., worm type and severity level) and the vendor's signature. This patch should be delivered to the phone via a HTTPS data connection or incrementally using Secure SMS Messages [133]. Upon receiving the patch, the user first authenticates the patch origin. If it is from a trusted vendor, the user decides whether to install and activate the patch. A security update usually involves charges. However, recipients have incentive to install them simply because delivering numerous worm messages to others costs them much more.

Fig.6.3 illustrates a flow of this push-based–patching scheme. We note that this *automated* and *self-healing* mechanism of security update is based on our terminal-level anomaly detection (see Section 6.4.2). Besides, individual users sometimes voluntarily connect to the security vendors and request the latest patches for their phone devices. This belongs to a less effective *Pull-based* service, because phones are not patched in a timely manner. We give analysis on the effectiveness of the push-based–patching mechanism in Appendix 6.5 and evaluate them through simulations in Section 6.6.1.

6.5 Performance Analysis

In this section, we theoretically analyze the effectiveness of our systematic solution, namely the terminal-level and network-level defenses. We start with a NULL scheme case where no defense has been deployed in the system.

Table 6.1. Notation for analysis

Note	Explanation
r	Coverage radius of Bluetooth radio signal
v	Average moving speed of mobile phones
ρ	Distribution density of phones (uniform distribution)
λ_b	Percentage of Bluetooth–enabled cell phones
β	Probability that a victim delivers worm message to a neighbor (also represents the victim's *worm messaging rate*)
α	Probability that a node accepts a worm message
t	Time stamp
$p_{i,t}$	Probability that node i is infected at time t
$\zeta_{i,t}$	Probability that node i does not receive worm messages from its neighbors at time t
η_t	Infected population at time t
δ	Vendor's patching probability on an infected phone (also represents the vendor's *patching rate*)
N	Total cell-phone population in the system

6.5.1 Analysis of the NULL Scheme

We derive a mathematical model for characterizing worm spreads in cellular networks. Kephart and White [131] designed an epidemic model which adopts homogeneous network graphs to depict user communications and worm propagation. Their model assumes nodes have similar connectivity level. However, our network model in Section 6.2 clearly deviates from such homogeneity. In our analysis, we consider phone mobility and the resulting variations in their connectivity. To simplify the problem, we use discrete time to conduct recursive analysis [134, **?**]. Table 6.1 lists the notation we use.

We first consider the case when no defense has been deployed in the system. During a unit time, an infected node j delivers a worm message to its neighbor i with probability β. We denote the probability that node i gets infected at time t as $p_{i,t}$ and derive $\zeta_{i,t}$, the probability that a node i will not receive any worm messages from its neighbors at time t as

$$\zeta_{i,t} = \prod_{j \in Nr_{i,t}} (p_{j,t-1}(1-\beta) + (1-p_{j,t-1})) = \prod_{j \in Nr_{i,t}} (1 - \beta \cdot p_{j,t-1}), \tag{6.1}$$

where $Nr_{i,t}$ denotes all neighbors that deliver messages to i during $\{t-1, t\}$. We have $Nr_{i,t} = Nr_i \cup Nr^*_{i,t}$, where Nr_i is the set of nodes who include i in their address books and $Nr^*_{i,t}$ denotes those who have Bluetooth communication with node i during $\{t-1, t\}$. Note that Nr_i represents a relatively stable set. We derive the average size of $Nr^*_{i,t}$

$$|Nr^*_{i,t}| = ((\pi r^2 + 2rv) \cdot \rho - 1) \cdot \lambda_b^2 \ll |Nr_i|, \;\; when\; \lambda_b \ll 1.$$

This equation computes node i's average Bluetooth connections in unit time. It indicates that when $\lambda_b \ll 1$ (i.e., the percentage of Bluetooth–enabled phones is low or most phones reject connections when they meet), we can use Nr_i to approximate $Nr_{i,t}$ in Eq.6.1.

Next, we derive $p_{i,t}$, the probability that node i is infected at t. Note that there are two causes for the infection: either i has been infected at time $t-1$, or i is susceptible at $t-1$ but accepts a worm message from neighbors. Therefore, we have

$$\begin{aligned} p_{i,t} &= p_{i,t-1} + (1 - p_{i,t-1})(1 - \zeta_{i,t}) \cdot \alpha \\ &\approx 1 - \zeta_{i,t} + p_{i,t-1} \cdot \zeta_{i,t}, \quad if\ \alpha \to 1, \end{aligned} \tag{6.2}$$

where $i = 1 \ldots N$. Given the network topology and related parameters $(\alpha,\ \beta)$, we solve Eq.6.2 and obtain the evolution of the infected population $\eta_t = \sum_{i=1}^{N} p_{i,t}$.

6.5.2 Analysis of the Systematic Countermeasure

We first derive a generic worm containment model for cellular networks. Using this model, we evaluate our systematic approach by analyzing the contributions of its components.

Generic Containment Model We know that node i remains *healthy* (susceptible) at time t due to one of the following reasons: (1) i was healthy at $t-1$ and it did not receive worm messages during $\{t-1, t\}$; (2) i was infected at $t-1$, but it eventually became healthy at t. Meanwhile, it did not receive worm messages; (3) i was infected at $t-1$, it received worm messages but eventually became healthy at time t. We note that in case (2) and (3), node i installed a patch from the security vendor at a certain time-point between $t-1$ and t. Thus, we compute the probability that node i remains healthy at time t

$$1 - p_{i,t} = (1 - p_{i,t-1})\zeta_{i,t} + \delta p_{i,t-1}\zeta_{i,t} + (1 - \frac{\alpha}{2})\delta p_{i,t-1}(1 - \zeta_{i,t}), \tag{6.3}$$

where $i = 1 \ldots N$, δ denotes the patching rate, and $\zeta_{i,t}$ is the probability that node i will not receive worm messages from neighbors during $\{t-1, t\}$ (Eq. 6.1). Note that here we assume in case (3) the patching happened at time $t - \frac{\Delta t}{2}$, where Δt is the length of unit time. This equation gives a generic worm containment model for cellular networks. Given network topology $G(V, E)$, worm messaging rate β and patching rate δ, we can derive the evolution of infected population, which is $\eta_t = \sum_{i=1}^{N} p_{i,t}$. Next, we derive an epidemic threshold for this worm containment model.

Theorem 1. (Containment Condition) *In our worm containment model, if worm spread eventually dies out, then it is necessarily true that* $\frac{2\beta}{(3-\alpha)\delta} < \tau = \frac{1}{\lambda_{1,A}}$, *where* $\lambda_{1,A}$ *is the largest eigenvalue of the adjacency matrix* $\mathbf{A}$ *of network graph* $G(V, E)$.

This theorem shows the basic requirement for a good defense. It can be proved by substituting Eq.6.3 to Theorem 1 in [134]. We give the proof details in Appendix B, it Also, we derive that when worm spread is diminishing due to the countermeasure, infection probability $p_{i,t}$ ($i = 1 \ldots N$) decays exponentially over time (Corollary 1 in [134]). This also indicates that the infected population decreases exponentially over time. Therefore, we may further derive the defense time needed for achieving worm extinction in the system.

Theorem 2. (Containment Time) *If worm spread dies out in a cellular network, the time it takes to reach extinction satisfies* $T_e \leq \log_{\lambda_{1,S}} \frac{N_e}{N_0}$*, where* $\lambda_{1,S} = 1 - (\frac{3-\alpha}{2})\delta + \beta\lambda_{1,A}$*,* N_0 *and* N_e *denote the initial and the final infected population, respectively.*

The proof is given in Appendix B. This theorem indicates that lower values of infection parameters (α, β) and a higher patching rate δ accelerate the worm containment process.

Effectiveness of the Systematic Solution In our terminal-level defense, GTT blocks worm messages originating from compromised phones and IBS helps discard unauthorized messages arriving at recipients. Essentially, the former reduces worm messaging rate β while the latter reduces worm acceptance rate α. Let λ_g represent the fraction of GTT-enabled phones, we may derive

$$\beta = \beta_0(1 - \lambda_g) + \beta_0\lambda_g \cdot r_f < \beta_0, \tag{6.4}$$

where β_0 is the worm delivery rate without protection and $r_f \ll 1$ is the failure rate of a GTT test. Also, we have

$$\alpha = \alpha_0(1 - \lambda_g) + \alpha_0\lambda_g \cdot (1 - \lambda_g) < \alpha_0, \tag{6.5}$$

where α_0 is the worm acceptance rate without protection. From Eq.6.1 and 6.2, we know that this increases $\zeta_{i,t}$ (the probability i does not receive worm messages) and reduces $p_{i,t}$ (the probability i gets infected), respectively. Also, we can see from Theorem 1 and 2 that GTT helps reduce the left-hand side of the condition and the extinction time T_e.

Our network-level defense identifies compromised phones and employs a push-based–patching mechanism to disinfect these victims. Our goal is to find a good detection/patching rate, which not only nicely satisfies the epidemic threshold for worm containment but also reduces the extinction time. Let R_{mms} denote the fraction of MMS-based worms, C_{det} represent network's capability of detecting an infected phone and α_p denote a user's acceptance probability towards a security patch, we have

$$\delta = \overline{R_{mms}} \cdot 0 + R_{mms} \cdot C_{det} \cdot \alpha_p = R_{mms} \cdot C_{det} \cdot \alpha_p. \tag{6.6}$$

This equation shows that disinfections are influenced by the network's detection capability and users' willingness to install the patch. We introduced three strategies to improve the former factor in Section 6.4.2.

6.5.3 Summary

The terminal-level defense aims at eliminating worm messages from the system, instead of actively disinfecting victims. Without a good patching mechanism (i.e., a good δ), the containment condition $\frac{2\beta}{(3-\alpha)\delta} < \tau = \frac{1}{\lambda_{1,A}}$ can never be satisfied. On the other hand, our network-level defense achieves a higher detection rate (with lower false positives) as more and more users launch GTT on their terminals. *In this sense, the components in our systematic approach are complementary to each other and our best strategy is to combine these components together and deploy them simultaneously in the system.* Specifically, our systematic solution aims at achieving the lowest set of infection parameters on communication links and at the same time the highest patching rate on infected phones to obtain the best containment result, i.e., we have $Minimize(\alpha, \beta) \wedge Maximize(\delta)$. We validate our results in the next section.

6.6 Evaluation of Effectiveness

As an initial study, we implemented the user-level defense on real cell-phone devices and evaluate its feasibility and effectiveness.

Smart-phone Environment We chose to conduct the experiments on Symbian-OS–based phones because of its popularity. We adopted the Metrowerks Code-Warrior V3.1.1 for Symbian OS v7.0 as the integrated development environment (IDE). To test the compatibility with major smart-phone manufacturers, we implemented the schemes using two different Symbian C++ SDKs, namely the Nokia Series 60 2rd Edition (S60) and the Sony-Ericsson UIQ 2.1 (UIQ).

In our experiments, we chose S60 phones (Nokia 3230 and E62)as the target devices. Typically, each smart-phone we used has a 32-bit RISC CPU based on ARM-9 series (123-220MHZ) and an expandable memory of 10-32 MB. All the devices support both Blue-tooth and MMS (GPRS-based) communications. Using the CodeWarrior compiler, object codes are first built into the executable and tested in a PC-based WINSCW emulator. Eventually, successful programs are built for the ARM targets and deployed to the devices.

Table 6.2. Time delay of IBS and RSA (in milliseconds)

Device	**CPU**	**Scheme**	**128 bit**	**512 bit**	**1024 bit**
Nokia 3230	123 MHz	IBS	391	593	1,203
Nokia E62	235 MHz	IBS	242	387	736
S-E P900	156 MHz	IBS	315	528	1,009
Nokia 3230	123 MHz	RSA	–	261	414

Implementing IBS on Smartphones To achieve IBS in real smart-phone environments, we modified the MIRACL cryptographic library from Shamus Software [135] and ported it to Symbian OS v9.1. This library includes Boneh and Franklin's bilinear-paring–based IBE scheme [40]. In our protected MMS messaging process, every message that passes GTT is signed with the sender's private key before its delivery and is verified by the receiver using the sender's ID. During the experiment, we examined and compared the time performance of IBS on different smart-phone platforms (3 typical Symbian phones). We set the user ID (MSISDN) to 13 digits and varied the key size from 128 to 1024 bits during the test. A longer key provides a higher level of security to the scheme. Our test takes 20 runs and we report the mean of measurement data. Fig.6.2 shows the time performance (signing/verification) of IBS and RSA on different phone devices. The result clearly demonstrates a trade-off between security and time. Considering the security level (The IBS based on bilinear maps on non-supersingular curves [41] adopts a short key size of 171 bits to achieve 1024-bit security in RSA) and the time response (a processing delay less than 300 ms is acceptable), we conclude the 128-bit key size is a good choice for the current hardware configuration of Symbian phones, which typically have a 235 MHZ CPU or below.

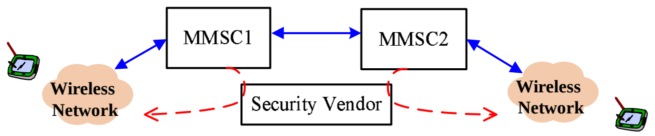


Figure 6.4. Configuration of the network simulator (similar as in Fig.6.3); Two MMSCs may belong to different service providers.

6.6.1 Evaluation of the Systematic Approach

Environmental Setting To study worm propagation and evaluate the proposed systematic defense, we designed a network simulator in which two MMSCs provide messaging service to 20,000 smartphone users (each for 10,000 users). Both MMSCs are securely connected to a security vendor (e.g., F-secure). As we mentioned in Section 6.2, cell phones have a number of contacts in their address books and their messaging relationship forms a logical social network. This social network resembles an email network whose topology is typically *heavy-tailed distributed* [136, 137]. We used the Barabasi Graph Generator [138] to generate power-law graphs and build messaging networks with different degree distribution [2]. To generate normal messaging traffic, we let each MMS user randomly delivers 5 message to its contacts within each hour [114].

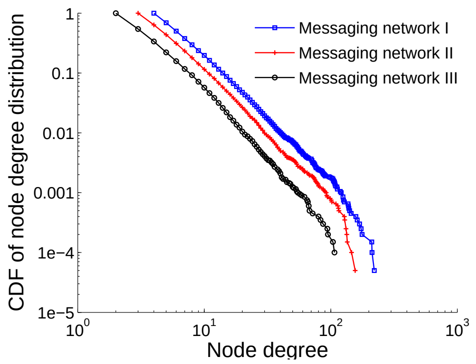
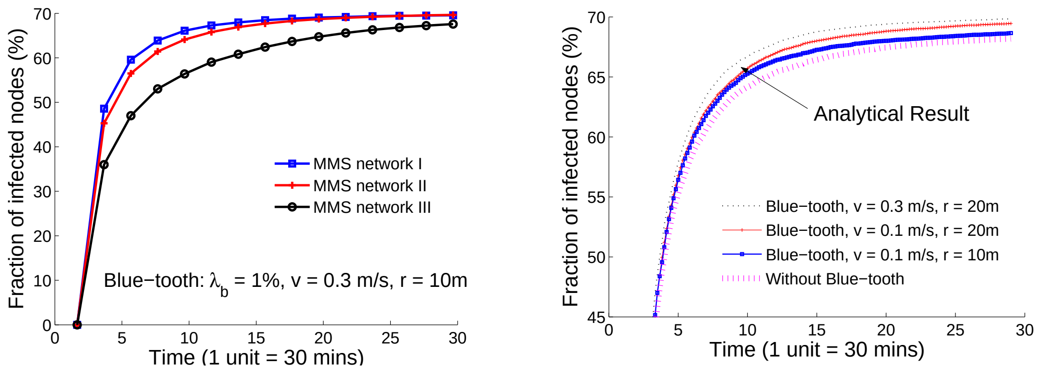


Figure 6.5. Degree distributions of three different messaging network. Network I is the most connected system.

For Bluetooth communications, we assume each node has 10 ∼ 20-meter radio range and its movement follows the random way-point model [139], in which each

[2]Although for test purpose we used power-law graphs for logical MMS networks, our result is independent of the power-law assumption in the network graph

node chooses a random destination, travels there and pauses for a constant time, and then picks another random destination. We adopted CMU's SETDEST tool in NS-2.29 to generate scenario files. The average node speed is $0.1 \sim 1.0$ m/s, and the pause time is $60 \sim 240$ seconds. Our tests took place in a 25,000-meter–wide square area. Each test takes 30 runs and we report the mean values.



(a) Worm spreads are fast in various cellular networks (through MMS Messaging and Bluetooth)

(b) Bluetooth contributes slightly to worm spreads (MMS network I, Bluetooth:λ_b=1%)

Figure 6.6. Cell-phone worm propagation in cellular networks. N=20,000 nodes, $\beta = 0.033$, $\alpha = 70\%$.

Worm Propagation Fig.6.5 shows degree distributions of three messaging networks. Note that here network I has the highest average degree while network III has the lowest. Fig.6.6 demonstrates worm propagation in the systems without defense. Fig.6.6(a) shows that when every user accepts a worm message (with attractive titles) in a probability $\alpha_0 = 70\%$, the worm spreading at a messaging rate $\beta_0 \approx 0.033$ (2 msg/min in Fig.5.10) quickly infects 70% of the entire population. Our result also indicates that worms spread faster in more connected systems.

Fig.6.6(b) illustrates Bluetooth connections have little impact on worm spreads. Our result clearly indicates that a wider communication range r and a higher moving speed v to some extent accelerate worm propagation in the system. This result is consistent with our propagation model in Section 6.5.1. Besides, the average node density ρ is another important factor. In our test, we chose $\rho = \frac{2\times10^4}{25\times25} = 32\ smartphones/km^2$.

Effectiveness of Proposed Defense Fig.6.7 shows the effectiveness of our terminal-level defense, namely the GTT protection on phone devices. We examine if this protection alone is effective enough for containing worm spreads. We studied a real

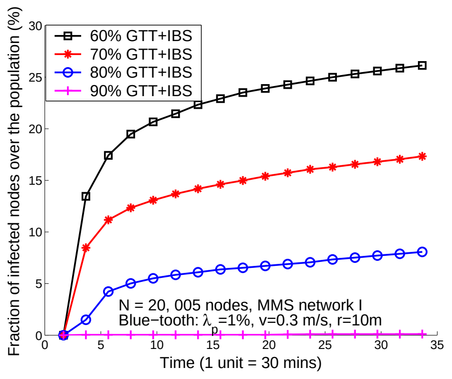
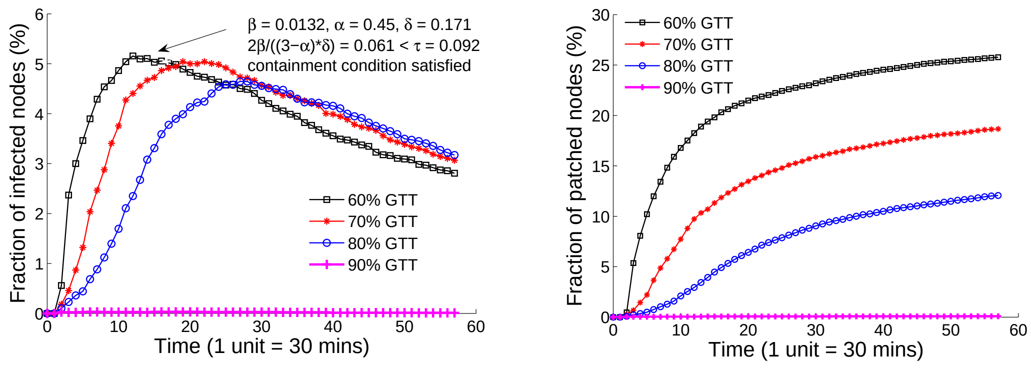


Figure 6.7. GTT helps slow down worm spreads. N=20,000 nodes, MMS network I, Bluetooth: $\lambda_b = 1\%, v = 0.3m/s, r = 10m$)

case when GTT is incrementally adopted by phone users. Fig.6.7 demonstrates that when 60% $\sim$ 90% phone users have launched the protection, worm propagation can be significantly slowed down (compared with Fig.6.6(a)). As more and more users deploy GTT on their devices, the system gets better protected against worms. Note that in our test we assumed a $r_f \leq 5\%$ error rate for GTT. The result also indicates that in real applications, gradually deploying terminal-level defense on phone devices can not completely throttle the worm spread in the system.



(a) Fraction of infected phones vs. time

(b) Fraction of patched phones vs. time

Figure 6.8. A systematic defense blocks worm attacks and disinfects victims. N=20,000 nodes, MMS network I, detection threshold β_{th} = 25 msg/hour, Bluetooth: λ_b = 1%, v = 0.3 m/s, r = 10m

We then studied our systematic defense which combines terminal-level and network-level defenses. Specifically, we set different percentages of phone devices as GTT–enabled and at the same time launched the network-level device monitoring and patching scheme. Meanwhile, we adjusted the worm detection threshold β_{th} and examined its influence on the patched population and the messaging traffic. Fig.6.8 demonstrates that our systematic countermeasure effectively throttles worm spreads in the system. From Fig.6.8(a), we can see that initially when there are not many victims identified by the network, the worm spread reaches a certain level (6%, much lower than in Fig.6.7). However, along with the worm detection and the automated patching process, compromised phones gradually get disinfected and the infected population starts to decrease. This result validates our worm containment condition in Theorem 1. Also, we can see that when more users adopt GTT (α is higher), worm detection reacts slower as there are less compromised phones, and this leads to a slower worm extinction (i.e., containment time becomes longer). This result is consistent with Theorem 2. Fig.6.8(b) shows the patched population versus time. It also suggests that a lower percentage of GTT-enabled phones in the system gets defense compensation of faster worm detection and patching from the network. In this sense, components in our systematic approach are complementary to each other.

Chapter 7

Summary and Open Issues

7.1 Summary

Based on the recent worm outbreaks in P2P systems and the emergence of various worm attacks in cellular networks, we predict that worms/malware are becoming the most dominating and devastating security threats to these hot networks in people's daily life. Current defenses in these systems rely too much on users' self recovery, for example, each user applies the security patch to disinfect her individual machine. This requires the user's precaution, awareness and skills; rather, we prefer designing automated, real-time and systematic countermeasures which leverage the existing terminal devices and network infrastructure, and internal communication mechanisms to actively and quickly launch protection against the worm attacks. These approaches of worm defense should be lightweight and easy to deploy. Also, they should keep one-step ahead and be well prepared in combating a wide variety of new worm threats in the near future. Specifically, for P2P networks, we have proposed a partition-based scheme and a CDS-based scheme to contain ultra-fast topological worm propagation; we have also proposed a download-based approach and a search-based approach for containing file-sharing worms in P2P environments. For cell-phone devices, we have proposed two device-level defenses, namely an access-control–based defense and a GTT-based defense, for blocking cell-phone worm attacks within terminals. We have also designed a systematic countermeasure which consists of both terminal-level and network-level defenses for combating cell-phone worms. Our solution is featured by an identity-based signature scheme at both sender and receiver sides, and a detection-based automatic patching scheme at the network side. More importantly, we showed through detailed reasoning and extensive evaluations that our solutions are are both applicable and effective for containing worm spreads in these hot networks. They can be easily implemented or embedded into existing client programs (e.g., terminal software) or network elements to launch protection in a timely manner. Below, we suggest some interesting future research that could be derived from our existing work.

7.2 Some Interesting Open Issues

Some interesting open issues from our research goes in the following two major directions.

- For file-sharing worm containment in P2P systems, we will consider node diversity in detecting worms and delivering security patches (the free-rider problem [140]), and user diversity which causes deviations from their expected file-sharing behavior. We will further study the cases of best and redundancy strategies in modeling and evaluating the download-based scheme. For the search-based scheme, there could also exist multiple client strategies. Current strategy will lead to the same set of file provides receiving many offerings, but other providers will receive nothing. A more reasonable mode would be randomly selecting k providers. In addition to the above two defense schemes, we will also propose a *reverse path forwarding scheme*. In this scheme, whenever a key node has downloaded a file from a source node, it immediately scans the file. If a worm infection is detected, the key node sends a security patch (with alert) back to the source. The receiver is encouraged to install the patch and then cleanses it folders. Also, it traces back to the neighboring infected points and forwards the security patch to these direct neighbors. In such a hop-by-hop and reverse path forwarding mode, all suspicious nodes on the infection paths (of a specific file) can be disinfected. We will further analyze and evaluate this new scheme in both structured and unstructured P2P file sharing systems.

- For cellphone worm containment in mobile networks, we will consider impacts from the diversity of smart-phone OSes and interworking between different service providers on our two-level defense schemes. In addition, we will consider potentially more devastating cellphone worms which automatically exploit software vulnerabilities and launch OS-level attacks in cellphone devices without relying on the phone user's activation of a received file. Specifically, malicious worms may cause buffer overflow attacks in phone devices and try to bypass the user-level defense (e.g., GTT), which typically has been embedded into the standard MMS messaging architecture in mobile operating systems (e.g., Symbian OS). To defend against these automated and system-level attacks, we will focus our study on two major directions. One is to analyze the content of the received MMS message (payload) and see if the attached file has contained any malicious code; the other is to monitor the process behavior and compare it with known normal process behavior. In the second approach, we need to investigate process states and their transitions in order to differentiate an abnormal process from a normal one. This can be achieved by constructing graphs for a series of system calls in the application (e.g., the messaging process) and examining the similarity between the call graphs. We propose adopting machine learning techniques such as Hidden Markov Model (HMM) to learn the state transition probabilities in the call graph and use it for detecting anomalies caused by process misbehavior.

Appendix A

Performance Analysis of Internal Patching against File-sharing Worms

We derive a new fluid model for worm propagation and analyze the security and performance of the download-based approach. We refer to notation in Table A.1.

Table A.1. Notation for worm propagation model

Note.	Explanation
N	the total number of hosts in the network
$V(t)$	the number of vulnerable hosts
$I_f(t)$	the number of infected hosts
$I_m(t)$	the number of immune hosts
$F(t)$	the total number of files
$h(t)$	the proportion of abnormal files in the system
$s(t)$	the average size of a shared folder
λ_d	the average rate of file download (files/hour)
λ_a	the probability a user activates the downloaded file
α	the percentage of key nodes in the system
β	the probability at which a user accepts a patch

A.1 Deriving the percentage of abnormal files

Proof. We refer to Table A.1 for notation. Let $A(t)$ denote the number of abnormal files at time t, we have $h(t) = A(t)/F(t)$. The change rate of the total number of

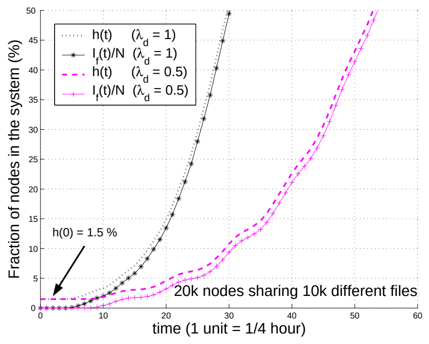


Figure A.1. Fraction of infected nodes vs. time

files $F(t)$ is

$$\frac{dF(t)}{dt} = \lambda_d N \quad i.e., \;\; F(t) = F_0 + \lambda_d N \cdot t,$$

where F_0 is the initial number of files in the system. A newly added abnormal file could be caused either by a vulnerable host activating another abnormal file in the same folder, or by a node directly downloading an abnormal copy from others. Therefore, we have the change rate of the number of abnormal files

$$\frac{dA(t)}{dt} = \lambda_d \cdot \lambda_a \cdot V(t) \cdot h(t) \cdot (s(t) - 1) + N\lambda_d h(t). \tag{A.1}$$

Considering $s(t) - 1 \approx F(t)/N$ and $A(t) = F(t) \cdot h(t)$, we solve equation A.1 and finally get

$$h(t) = h(0) \cdot e^{\frac{\lambda_d \cdot \lambda_a}{N} \int_0^t V(\tau) d\tau}, \tag{A.2}$$

where $h(0)$ is the initial percentage of abnormal files.

We may compute an approximation for $h(t)$. Assuming all the nodes have a similar size of $s(t)$ for their shared folders and the user parameter λ_a approaches 1, which means every client usually activates (opens) the file he has just downloaded, all the abnormal files should gradually be kept by those infected hosts in the system. Hence, we may derive $h(t) \approx \frac{I_f(t)}{N}$. $\square$

A.2 A Fluid Model for Worm Propagation

We first consider the case when no defense has been deployed in the system. Each node is either in vulnerable or immune state, i.e., relation $N = V(t) + I_f(t)$ always satisfies. We show the evolution status of the system under the worm threat. The vulnerable population decreases as some nodes unfortunately download abnormal files, activate these files and get infected. We have

$$\frac{dV(t)}{dt} = -\lambda_d \lambda_a \cdot V(t) \cdot h(t). \tag{A.3}$$

Here $1/\lambda_d$ is the average time a node takes to download a file, and $h(t)$ reflects the percentage of abnormal files at time t. Solving the above differential equation, we get

$$V(t) = N - I_f(t) = V(0) \cdot e^{-\lambda_d \lambda_a \int_0^t h(\tau) d\tau}, \tag{A.4}$$

where $V(0)$ denotes the initial number of vulnerable hosts. This equation indicates that the vulnerable population in the system decreases exponentially as there are more file downloads and activations; the increase of the proportion of abnormal files accelerates the worm spread. We further derive the file state.

Lemma 1. *In a P2P file-sharing system, the percentage of abnormal files can be computed as* $h(t) = h(0) \cdot e^{\frac{\lambda_d \cdot \lambda_a}{N} \int_0^t V(\tau) d\tau}$, *where* $h(0)$ *is the initial abnormal rate. An approximation of* $h(t)$ *can be computed as* $h(t) \approx \frac{I_f(t)}{N}$, *assuming* $\lambda_a \to 1$.

This lemma is proved in Appendix A.1. It shows that user behavior has significant impact on the percentage of abnormal files: more file downloads and activations lead to more infections. However, as the amount of files increases and the vulnerable population decreases, worm infection is gradually slowed down.

A.3 Analysis of the Download-based Defense

Time Performance Next, we examine the download-based defense. For simplicity, we assume users always adopt the random strategy to choose file providers (see 4.4.1) and all infected hosts have a patch acceptance probability β. We define *immunity rate* $i(t)$ as the fraction of immune nodes $i(t) = I_m(t)/N$, and let the initial immune population be $I_m(0)$. Note that these nodes, including the key nodes, either have applied the patch or does not expose the software vulnerability to the worm.

To study how long it takes to achieve a certain level of immunity rate, we formalize the problem as finding a lower bound t_0 for time t, so that we have $i(t) \geq \Psi$ when $t \geq t_0$, where $\frac{Im(0)}{N} \leq \Psi \leq 1$ is a predefined threshold.

Lemma 2. *In a file-sharing system which adopts the download-based defense, the number of immune nodes is $I_m(t) = N + (I_m(0) - N) \cdot e^{-\lambda_d \alpha \beta t}$ and the system takes at least $t_0 = \frac{1}{\alpha\beta\lambda_d} \ln \frac{N - I_m(0)}{N(1-\Psi)}$ hours to achieve an immunity rate Ψ.*

Proof. From the state diagram in Fig 6.2, we know that $N = V(t) + I_f(t) + I_m(t)$ always holds. Each time when a node downloads a file from a key node, it also receives a patch and the user decides whether or not to accept it. Note that only infected and vulnerable $(I_f(t) + V(t))$ nodes are immunized/disinfected in this process. We derive the change of immunity rate.

$$\frac{dI_m(t)}{dt} = (I_f(t) + V(t))\lambda_d \alpha \beta = (N - I_m(t)) \cdot \lambda_d \alpha \beta. \tag{A.5}$$

Here α also denotes the probability that each client selects a key node as the provider. Solving this differential equation for $I_m(t)$, we get the number of immune nodes in the system

$$I_m(t) = N + (I_m(0) - N) \cdot e^{-\lambda_d \alpha \beta \cdot t}. \tag{A.6}$$

From the given condition $i(t) = I_m(t)/N \geq \Psi$, we may further derive $t \geq t_0 = \frac{1}{\alpha\beta\lambda_d} \ln \frac{N - I_m(0)}{N(1-\Psi)}$. ☐

Fig A.2 illustrates the change of immunity rate $i(t)$ when the percentage of key nodes (α) varies from 5% to 15%. Clearly, as there are more patch distributors, the system takes less time to reach a certain level of immunity rate (in our case 90%). For example, when $\alpha = 5\%$, it takes 60 hours for 90% of nodes to receive the patch, whereas it takes 20 hours when $\alpha = 15\%$. This figure also shows that in the random selection mode, each downloader (including those not interested in the popular files) will eventually receive the patch from a key node.

System Evolution Status We also examine the evolution status of the system which adopts the download-based defense. During the worm containment, a vulnerable host either (1) becomes infected when it downloads and activates an abnormal file from a non-key node, or (2) gets immunized when it downloads a file

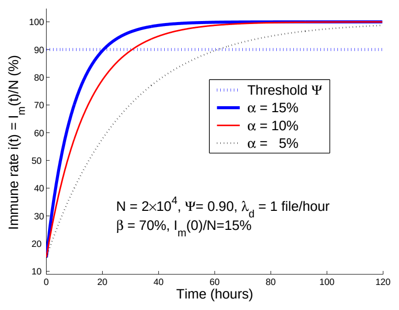


Figure A.2. Immunity rate as a function of time an and key nodes

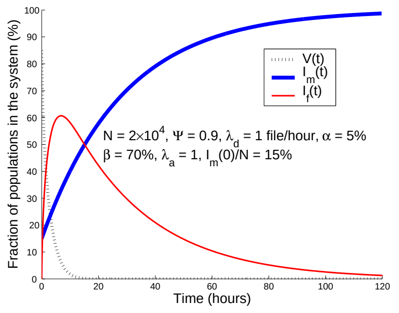


Figure A.3. System evolution status under the download-based defense

from a key node and accepts the patch. Hence, we set up the following equation for the change of the vulnerable population.

$$\frac{dV(t)}{dt} = -\lambda_d\lambda_a(1-\alpha)\cdot V(t)\cdot h(t) - \lambda_d\alpha\beta\cdot V(t). \qquad \text{(A.7)}$$

Note that here $\lambda_d\lambda_a(1-\alpha)\cdot V(t)\cdot h(t)$ computes the reduction caused by (1) $(1-\alpha$ denotes the probability of downloading from a non-key nodes) and $\lambda_d\alpha\beta\cdot V(t)$ computes the reduction caused by (2). We use the approximation $h(t) \approx \frac{I_f(t)}{N}$ and the solution in Equ.A.6 to solve this differential equation for $V(t)$. We also compute $I_f(t)$ using the relation $N - I_m(t) - V(t)$. Fig A.3 illustrates the evolution status of a file-sharing system. Initially, the percentage of infected nodes increases as the worm surges. However, when more and more file downloaders receive the patch, worm infections are gradually cleansed from the network and the infected population starts to decrease. Eventually, immune nodes become the major population. The figures also indicate that the immune time t_0 is determined by several factors: the fraction of the key nodes (α), the file downloading rate (λ_d), patch acceptance rate (β) and the initial immunity rate. Our analytical result has been validated in Section 4.7 (Fig 4.5).

A.4 Analysis of the Search-based Defense

Time Performance We analyze the effectiveness of the search-based defense. Let k denote the average number of suspicious targets to which a key node distributes the security patch, we first derive how long the system takes to achieve an immunity rate Ψ. According to the state diagram in Fig 6.2, $N = V(t) + I_f(t) + I_m(t)$ always holds. In the search-based scheme, the increased immune population comes from either vulnerable nodes or infected nodes. Hence, the rate at which the immune population increases can be computed as

$$\begin{aligned} \frac{dI_m(t)}{dt} &= N\lambda_d \cdot \alpha h(t)(\frac{N - I_m(t)}{N}) \cdot k\beta \\ &= a(N - I_m(t)) \cdot h(t), \end{aligned} \tag{A.8}$$

Where $a = \alpha\beta\lambda_d k$. Note that $\alpha h(t)(\frac{N-I_m(t)}{N})$ computes the probability that a key node downloads an abnormal file from those non-immune hosts.

Also, the decrease of the vulnerable population could be caused either by (1) worm infections or (2) by host immunizations (or disinfections). Therefore, the rate at which vulnerable nodes become either infected or immunized is

$$\begin{aligned} \frac{dV(t)}{dt} &= -\lambda_a\lambda_d V(t)h(t) - N\lambda_d \cdot \alpha h(t)(\frac{V(t)}{N}) \cdot k\beta \\ &= -(a+b)V(t)h(t). \end{aligned} \tag{A.9}$$

where $b = \lambda_a\lambda_d$. Here $\lambda_a\lambda_d V(t)h(t)$ computes the reduction caused by (1); $N\lambda_d \cdot \alpha h(t)(\frac{V(t)}{N}) \cdot k\beta$ computes the reduction caused by (2), in which $\alpha h(t)(\frac{V(t)}{N})$ denotes the probability a key node downloads an abnormal file from a vulnerable host (not infected yet). In this case, the latter receives the patch and could be immunized.

Finally, we know that the infected population (1) increases when some vulnerable nodes get infected, and (2) decreases when some victim nodes have been disinfected. Hence we derive the following differential equation.

$$\begin{aligned} \frac{dI_f(t)}{dt} &= \lambda_a\lambda_d V(t)h(t) - N\lambda_d \cdot \alpha h(t)(\frac{I_f(t)}{N}) \cdot k\beta \\ &= bV(t)h(t) - aI_f(t)h(t). \end{aligned} \tag{A.10}$$

Note that here $\lambda_a\lambda_d V(t)h(t)$ computes the increase of infected nodes caused by (1); $N\lambda_d \cdot \alpha h(t)(\frac{I_f(t)}{N}) \cdot k\beta$ computes the reduction of infected nodes caused by (2), where $\alpha h(t)(\frac{I_f(t)}{N})$ denotes the probability a key node downloads an abnormal file from an infected host. In this case, the latter receives the patch and could be disinfected. To solve these differential equations, we derive the immune population $I_m(t)$. We divide Equ.A.8 by Equ.A.9 and get

$$V(t) = V_0^{-\frac{b}{a}} \cdot (N - I_m(t))^{\frac{a+b}{a}}, \tag{A.11}$$

where $V_0 = V(0)$ denotes the initial vulnerable population in the system. We then apply the approximation $h(t) \approx \frac{I_f(t)}{N} = \frac{N - I_m(t) - V(t)}{N}$ and substitute Equ.A.11 into Equ.A.8. Thus, we have

$$\frac{du}{dt} = -\frac{a \cdot V_0}{N} \cdot u^2 (1 - u^{\frac{b}{a}}) \tag{A.12}$$

where $u = (N - I_m(t))/V_0$. We further solve this equation for $I_m(t)$ and illustrate the change of $I_m(t)$ in Fig A.4. This figure indicates that under an average size $k = 30$, the search-based approach only needs to deploys 5% nodes as key nodes and help the system achieve a 90% immunity rate within 60 hours.

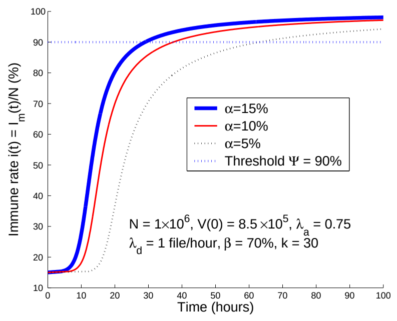


Figure A.4. Immune population vs. time and key nodes

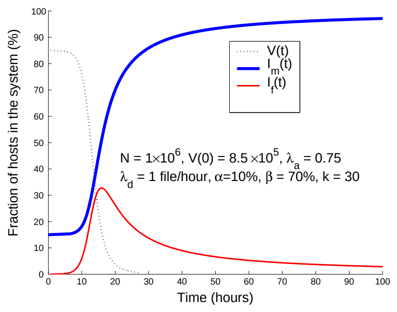


Figure A.5. System evolution status in search-based scheme

System Evolution Status Adopting the similar method as above to solve Equ. A.8, A.9, A.10, we derive the following

$$\frac{dV(t)}{dt} = -\frac{a+b}{N}(c \cdot V^{1+\frac{a}{a+b}}(t) - V^2(t)), \tag{A.13}$$

where $c = V_0^{\frac{b}{a+b}}$. Hence we may compute $V(t)$. Using $I_f(t) = N - V(t) - I_m(t)$, we may further derive the infected population $I_f(t)$. We illustrate the system evolution status in Fig A.5. The figure shows that the infected population initially increases due to the surging worm. However, this triggers the defense and many suspicious nodes receive the patch. Eventually worm infections are eliminated and immune nodes become the major population. The above analytical result has been validated in Section 4.7 (Fig 4.5).

Appendix B

Performance Analysis of Systematic Countermeasures against Cell-phone Worms

B.1 Epidemic Threshold of Cell-phone Worms

Proof. We have node i's healthy probability from Eq.6.3

$$1 - p_{i,t} = (1 - p_{i,t-1})\zeta_{i,t} + \delta p_{i,t-1}\zeta_{i,t} + (1 - \frac{\alpha}{2})\delta p_{i,t-1}(1 - \zeta_{i,t}).$$

We rearrange the above items and get

$$1 - p_{i,t} \approx (1 - \frac{\alpha}{2})\delta p_{i,t-1} + (1 - p_{i,t-1} + \frac{1}{2}\delta p_{i,t-1}) \cdot \zeta_{i,t},$$

where $\zeta_{i,t} = \prod_{j \in Nr_{i,t}} (1 - \beta \cdot p_{j,t-1})$. Following the approximation $(1-a)(1-b) \approx 1 - a - b$ (when $a \cdot b << 1$), we derive the healthy probability

$$1 - p_{i,t} \approx 1 + (\frac{3-\alpha}{2}\delta - 1)p_{i,t-1} - \beta \sum_{j} p_{j,t-1}.$$

Thus, we have $p_{i,t} \approx (1-\delta^{'})p_{i,t-1} + \beta \sum_j p_{j,t-1}$,where $\delta^{'} = \frac{3-\alpha}{2}\delta$. Converting this equation to the matrix form

$$\mathbf{P_t} \approx ((\mathbf{1}-\delta^{'})\mathbf{I} + \beta\mathbf{A})\mathbf{P_{t-1}} = \mathbf{S}\mathbf{P_{t-1}} = \mathbf{S^t}\mathbf{P_0}, \tag{B.1}$$

where $\mathbf{S} = (\mathbf{1}-\delta^{'})\mathbf{I}+\beta\mathbf{A}$, and $\mathbf{A}$ is the adjacency matrix of network graph $G(V,E)$. Decomposing the matrix, we get

$$\mathbf{P_t} \approx \sum_{\mathbf{i}} \lambda^{\mathbf{t}}_{\mathbf{i,S}}\ \mathbf{v_{i,S}}\ \mathbf{tr}(\mathbf{v_{i,S}})\ \mathbf{P_0}, \tag{B.2}$$

where $\lambda_{i,S}$ is the i'th eigen value of $\mathbf{S}$, $\mathbf{v_{i,S}}$ is the i'th eigen vectior of $\mathbf{S}$, and $\mathbf{tr}(\mathbf{v_{i,S}})$ is the transpose of $\mathbf{v_{i,S}}$. According to [134], we have

$$\lambda_{i,S} = 1 - \delta^{'} + \beta\lambda_{i,A}, \quad \forall i \tag{B.3}$$

The goal of our defense is to let worm spread declines until to the extinction, hence vector $\mathbf{P_t}$ should approach zero for t, which is large enough. This happens when $\forall i, \lambda^t_{i,S}$ goes to 0, and we should have the largest eigen value $\lambda_{1,S} < 1$, i.e., $1 - \frac{3-\alpha}{2}\delta + \beta\lambda_{1,A} < 1$. Hence, we get $\frac{2\beta}{(3-\alpha)\delta} < \tau = \frac{1}{\lambda_{1,A}}$ □

B.2 Time to Reach Worm Extinction

Proof. Suppose in $t = 0$, we have N_0 infected smartphones in the system. We want to derive the time T_e, at when the infected population decreases to N_e, due to our network-level countermeasure (the push-based–patching scheme). According to Eq.B.2, $\mathbf{P_t} \approx \sum_{\mathbf{i}} \lambda^{\mathbf{t}}_{\mathbf{i,S}}\ \mathbf{v_{i,S}}\ \mathbf{tr}(\mathbf{v_{i,S}})\ \mathbf{P_0} = \lambda^{\mathbf{t}}_{\mathbf{1,S}} * \mathbf{C}$, where $\mathbf{C}$ is a constant vector. Therefore, we derive the infected population as

$$n_t = \sum_{i=1}^{N} p_{i,t} = \lambda^t_{1,S} * \sum_i C_i, \tag{B.4}$$

where C_i is the i'th element in vector $\mathbf{C}$. This means the number of infected nodes decays exponentially over time. We have $N_e = \lambda^{T_e}_{1,S} \cdot N_0$, i.e., $T_e = \log_{\lambda_{1,S}} \frac{N_e}{N_0}$, where $\lambda_{1,S} = 1 - \frac{3-\alpha}{2}\delta + \beta\lambda_{1,A}$. □

Bibliography

[1] Zhou, L., L. Zhang, F. McSherry, N. Immorlica, M. Costa, and S. Chien (2005) "A First Look at Peer-to-Peer Worms: Threats and Defenses," .

[2] http://en.wikipedia.org/wiki/Gnutella.

[3] http://en.wikipedia.org/wiki/KaZaA.

[4] Stoica, I., R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan (2002) "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications," in *IEEE/ACM Transactions on Networking*.

[5] http://en.wikipedia.org/wiki/Skype.

[6] http://www.bittorrent.com.

[7] Rowstron, A. and P. Druschel (2001) "Pastry: Scalable, distributed object location and routing for large-scale p2p systems," in *Proc. of International Conference on Distributed System Platforms*.

[8] Zhao, B., J. Kubiatowicz, and A. Joseph (2001) "Tapestry: An infrastructure for fault-resilient wide-area location and routing," in *Technical Report UCB/CSD-01-1141, U.C.Berkeley*.

[9] Ratnasamy, S., P. Francis, M. Handley, R. Karp, and S. Shenker (2001) "A Scalable Content-Addressable Network," in *SIGCOMM'01*.

[10] http://en.wikipedia.org/wiki/Frequency_division_multiple_access.

[11] http://en.wikipedia.org/wiki/CDMA.

[12] http://en.wikipedia.org/wiki/GSM.

[13] http://en.wikipedia.org/wiki/General_Packet_Radio_Service.

[14] Zeng, A. and et al. (1999) "Recent advances in cellular wireless communications," in *IEEE Communications*.

[15] (2006) "http://www.symbian.com/developer/index.html," Symbian Ltd.

[16] (2006) "Windows Mobile-based Smartphones," http://www.microsoft.com/windowsmobile, Microsoft Corp.

[17] Staniford, S., D. Moore, V. Paxson, and N. Weaver (2004) "The Top Speed of Flash Worms," *Proc. of ACM Workshop on Rapid Malcode (WORM'04)*.

[18] Moore, D., C. Shannon, and K. Claffy (2002) "Code Read: A Case Study on the Spread and Victims of an Internet Worm," *Proc. of ACM SIGCOMM Internet Measurement Workshop*.

[19] Pang, R., V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson (2004) "Characteristics of Internet Background Radiation," *Proc. of ACM IMC*.

[20] Singh, S., C. Estan, G. Varghese, and S. Savage (2003) "The Earlybird System for Real-time Detection of Unknown Worms," *Tech. Rep., CS2003-0761*.

[21] Kim, H. and B. Karp (2004) "Autograph: Toward automated, distributed worm signature detection," *Proc. of the 13th Usenix Security Symposium*.

[22] Newsome, J., B. Karp, and D. Song (2005) "Polygraph: Automatic Signature Generation for Ploymorphic Worms," *IEEE Security and Privacy Symposium*.

[23] J.Jung, V. Paxson, A. Berger, and H. Balakrishnan (2004) "Fast Portscan Detection Using Sequential Hypothesis Testing," *Proc. of IEEE Symposium on Security and Privacy*.

[24] Wang, H., C. Guo, D. Simon, and A. Zugenmaier (2004) "Shield: Vulnerability-driven Network Filters for Preventing Known Vulnerability Exploits," *Proc. of the ACM SIGCOMM Conference*.

[25] Costa, M., J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, and P. Barham (2005) "Vigilante: End-to-end Containment of Internet Worms," *SOSP'05*.

[26] Liang, Z. and R. Sekar (2005) "Fast and Automated Generation of Attacks Signatures: A Basis for Building Self-protecting Servers," *Proc. of 12th ACM Conference on Computer and Communication Security*.

[27] Wang, K. and S. Stolfo (2004) "Anomalous Payload-based Network Intrusion Detection," *RAID'04*.

[28] Wang, K., G. Cretu, and S. Stolfo (2005) "Anomalous Payload-based Worm Detection and Signature Generation," *RAID'05*.

[29] Locasto, M., K. Wang, a. Keromytis, and S. Stolfo (2005) "Flips: Hybrid Adaptive Intrusion Prevention," *RAID'05*.

[30] http://securityresponse.symantec.com.

[31] "United States Computer Emergency Reading Team," http://www.us-cert.gov/nav/t01.

[32] Costa, M., J. Crowcroft, M. Castro, and A. Rowstron (2004) "Can We Contain Internet Worms?" .

[33] Lactaotao, M. (2005) "Security information: virus encyclopedia: symbos_comwar.a: technical details, trend micro incorporated," .

[34] Chien, E. (2004) "Security Response: Symbos.skull, symantec corporation," .

[35] Ferrie, P., P. Szor, R. Stanev, and R. Mouritzen (2004) "Security Response: Symbos.cabir, symantec corporation," .

[36] Stutzbach, D., R. Rejaie, and S. Sen (2005) "Characterizing Unstructured Overlay Topologies in Modern P2P File-sharing Systems," in *Internet Measurement Conference*.

[37] http://en.wikipedia.org/wiki/Comparison_of_file_sharing_applications.

[38] (March, 2006) "3GPP TS 23.140 V6.12.0," in *http://www.3gpp.org/ftp/Specs/*.

[39] Shamir, A. (1984) "Identity-base cryptosystems and signature schemes," in *Proc. of Crypto'84*, Springer-Verlag.

[40] Boneh, D. and M. Franklin (2001) "Identity-Based Encryption from the Weil Pairing," in *Proc. of Crypto'01*, Springer-Verlag.

[41] Boneh, D., B. Lynn, and H. Shacham (2001) "Short Signatures from the Weil Pairing," in *ASIACRYPT'01*.

[42] Cha, J. and J. Cheon (2003) "An Identity-Based Signature from Diffie-Hellman Groups, Public Key Cryptography," in *Proc. of PKC'03*.

[43] Weaver, N., V. Paxson, S. Staniford, and R. Cunninggham (2004) "A taxonomy of computer worms," in *USENIX Security'04*.

[44] http://www.symantec.com/avcenter/security/Content/7680.html.

[45] "KaZaA and Fasttrack P2P network client buffer overflow vulnerability," http://secunia.com/advisories/8868/.

[46] M.Vojnovic and A. Ganesh (2005) "On the Effectiveness of Automatic Patching," in *WORM'05*.

[47] Cai, K., Y. Kwang, S. Song, and Y. Chen (2005) "Collaborative Internet Worm Containment," in *IEEE Security and Privacy'05*.

[48] Altekar, G., I. Bagrak, P. Burstein, and A.Schultz (2005) "OPUS: Online Patches and Updates for Security," in *USENIX Security'05*.

[49] Ripeanu, M., I. Foster, and A. Iamnitchi (2002) "Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design," in *IEEE Internet Computing Journal*.

[50] Karypis, G. and V. Kumar (1998) "A fast and high quality multilevel scheme for partitioning irregular graphs," in *SIAM Journal on Scientific Computing*.

[51] Anthony, B. and G. Blelloch http://www.cs.cmu.edu/afs/cs/project/pscico-guyb/realworld.

[52] Kernighan, B. and S. Lin (1970) "An efficient heuristic procedure for partitioning graghs," in *Bell Systems Technical Journal*.

[53] Chawathe, Y., S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker (2003) "Making Gnutella-like P2P systems scalable," in *ACM SIGCOMM'03*.

[54] Guha, S. and S. Khuller (Apr. 1998) "Approximation algorithm for connected dominating sets," in *Algorithmica*.

[55] Garey, M. and D. Johnson (1979) "Computers and Intractability: A Guide to the Theory of NP-Completeness," Freeman, New York.

[56] Kutten, S. and D. Peleg (1995) "Fast distributed construction of k-dominating sets and applications," in *PODC'95*.

[57] Chen, Y. and A. Liestman (2002) “Approximating minimum size weakly-connected dominating sets for clustering mobile ad hoc networks,” in *ACM MOBIHOC’02*.

[58] Zegura, E., K. Calvert, and S. Bhattacharjee (March 1996) “How to model an internetwork,” in *IEEE INFOCOMM’96*.

[59] Good, N. and A. Krekelberg “Usability and privacy: a study of KaZaA P2P file-sharing, 2002,” in *http://www.hpl.hp.com/shl/papers/kazaa/index.html*.

[60] “http://www.facetime.com/securitylabs/imp2pthreats.aspx,” .

[61] “www.viruslist.com/en/virusesdescribed?chapter=153311928,” .

[62] Zetter, K. (2004) “KaZza Delivers More Than Tunes,” in *The Wired Magazine*.

[63] Shin, S., J. Jung, and H. Balakrishnan “Malware Prevalence in the KaZaA File-Sharing Network,” in *ACM Internet Measurement Conference’06*.

[64] Dumitriu, D., E. Knightly, A. Kuzmanovic, I. Stoica, and W. Zwaenepoel (2005) “Denial-of-Service resilience in peer-to-peer file sharing systems,” in *Sigmetrics’05*.

[65] kumar, R., D. Yao, A. Bagchi, K. Ross, and D. Rubenstein (2006) “Fluid modeling of pollution proliferation in P2P networks,” in *Sigmetrics’06*.

[66] Thommes, R. and M. Coates (2006) “Epidemiological modeling of peer-to-peer viruses and pollution,” in *Infocom’06*.

[67] Walsh, K. and E. Sirer (2005) “Thwarting P2P pollution Using object reputation,” in *Cornell technical report TR2005-1980*.

[68] Vojnovic, M. and A. Ganesh (2005) “On the race of worms, alerts and patches,” in *ACM Workshop on WORM*.

[69] Hughes, D., G. Coulson, and J. Walkerdine (2005) “Free Riding on Gnutella Revisited: the Bell Tolls,” in *Proc. of IEEE Distributed Systems Online*.

[70] Leibowitz, N., M. Ripeanu, and A. Wierzbicki (2003) “Deconstructing the KaZaA Network,” in *Proc. of IEEE IWAPP’03*.

[71] "Using binary delta compression technology to update windows operating systems," Microsoft online White Paper.

[72] Bellissimo, A., J. Burgess, and K. Fu (2006) "Secure Software Updates: Disappointments and New Challenges," in *USENIX Hot Topics in Security Workshop (Hot-Sec'06)*.

[73] "The Gnutella Protocol Specification," http://www.the-gdf.org.

[74] Brumley, D., J. Newsome, D. Song, H. Wang, and S. Jha (2006) "Towards Automatic Generation of Vulnerability-based Signatures," in *Proceedings of the 2006 IEEE Symposium on Security and Privacy*.

[75] Barrantes, E., D. Ackley, T. Palmer, D. Stefanovic, and D. Zov (2003) "Randomized instruction set emulation to disrupt binary code injection attacks," in *ACM CCS'03*.

[76] Kc, G., A. Keromytis, and V. Prevelakis (Oct. 2003) "Countering code-injection attacks with instruction-set randomization," in *ACM CCS'03*.

[77] Newstone, J. and D. Song (2005) "Dynamic taint analysis: Automatic detection and generation of software exploit attacks," in *Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS'05)*.

[78] Mulz, D., F. Valeur, C. Kruegel, and G. Vigna (2006) "Anomalous system call detection," in *ACM Transactions on Information and System Security, 2006*.

[79] "P2PSim: a simulator for peer-to-peer protocols," http://pdos.csail.mit.edu/p2psim.

[80] Joseph, S. (2002) "NeuroGrid: Semantically Routing Queries in Peer–to–Peer Networks," in *International Workshop on Peer-to-Peer Computing*.

[81] http://www.viruslist.com/en/viruslist.html?id=49790.

[82] Gkantsidis, C., T. Karagiannis, P. Rodriguez, and M. Vojnovic (2006) "Planet Scale Software Updates," in *Proc. of SIGCOMM'06*.

[83] http://www.f-secure.com/wireless/threats.

[84] Chien, E. (2005) "Security Response: SymbOS.Mabir, Symantec Corporation," .

[85] http://www.f-secure.com/v-descs/flexispy_a.shtml.

[86] Cole, R., N. Phamdo, M. Rajab, and A. Terzis (2005) "Requirements on Worm Mitigation Technologies in MANETS," in *PADS'05*.

[87] Mickens, J. and B. Noble (2005) "Modeling Epidemic Spreading in Mobile Networks," in *ACM WiSe'05*.

[88] Bose, A. and K. Shin (2006) "On Mobile Virus Exploiting Messaging and Bluetooth Services," in *SecureComm'06*.

[89] Shackman, M. (2005) "Symbian OS v9 – Platform Security," Symbian Developer Network.

[90] Hurman, T. (2006) "Exploring Windows CE Shellcode," Pentest Security Assurance.

[91] http://securityresponse.symantec.com/.

[92] Mulliner, C., G. Vigna, D. Dagon, and W. Lee (2006) "Using Labeling to Prevent Cross-service attacks against smart phones," in *DIMVA'06*.

[93] Mulliner, C. and G. Vigna (2006) "Vulnerability Analysis of MMS User Agents," in *Proc. of ACSAC'06*.

[94] Guo, C., H. Wang, and W. Zhu (Nov. 2004) "Smartphone Attacks and Defenses," in *HotNets-III, UCSD*.

[95] Racic, R., D. Ma, and H. Chen (2006) "Exploiting MMS Vulnerabilities to Stealthily Exhause Mobile Phone's Battery," in *SecureComm'06*.

[96] Bose, A. and K. Shin (2006) "Proactive Security for Mobile Messaging Networks," in *Proc. of WiSe'06*.

[97] Chen, J., S. Wongand, H. Yang, and S. Lu (2007) "SmartSiren: Virus Detection and Alert for Smartphones," in *Proc. of MobiSys'07*.

[98] Miller, C., J. Honoroff, and J. Mason "Security Evaluation of Apple's iPhone," http://www.securityevaluators.com/iphone.

[99] http://www.cse.psu.edu/˜ lxie/snapshots/mserver.html.

[100] http://direct.motorola.com/hellomoto/us/motoming.

[101] http://www.linuxdevices.com/news/NS9419753617.html.

[102] Sailer, R., X. Zhao, T. Jaeger, and L. Doom (2004) "Design and Implementation of a TCG-based Integrity measurement architecture," in *Proc. of Usenix Security Symposium'04*.

[103] Jaeger, T., R. Sailer, and U. Shankar (2006) "PRIMA: Policy-Reduced Integrity Measurement Architecture," in *Proc. of SACMAT'06*.

[104] http://www.virtuallogix.com/.

[105] (2006) "Symbian OS: Overview to Security," in *http://www.forum.nokia.com/info*, Nokia Corperation.

[106] Biba, K. J. (1977) *Integrity Consideratio for Secure Compuer System*, *Tech. rep.*, Mitre Corp. Report TR-3153, Bedford, Mass.

[107] Clark, D. and D. Wilson (1987) "A Comparison of Commercial and Military Computer Security Policies," in *Proc. of IEEE Symposium on Security and Privacy*.

[108] Loscocco, P. and S. Smalley (2001) "Integrating Flexible Support for Security Policies into the Linux Operating System," in *Proceedings of USENIX Annual Technical Conference*, pp. 29 – 42.

[109] Saltzer, J. H. and M. D. Schroeder (1975) "The protection of information in computer systems," *Proceedings of the IEEE*, **63**(9), pp. 1278–1308.

[110] Boebert, W. E. and R. Y. Kain (1985) "A Practical Alternative to Hierarchical Integrity Policies," in *Proceedings of the Eighth National Computer Security Conference*.

[111] "TCG Mobile Reference Architecture Specification Version 1.0," https://www.trustedcomputinggroup.org/specs/mobilephone.

[112] Zhang, X., O. Aciicmez, and J. Seifert (2007) "A Trusted Mobile Phone Reference Architecture via Secure Kernel," in *ACM workshop on Scalable trusted computing*.

[113] et al., J. H. (2006) "Content Based SMS Spam Filtering," in *DocEng'06*.

[114] Enck, W., P. Traynor, P. McDaniel, and T. L. Porta (2005) "Exploiting open functionality in sms-capable celluar networks," in *CCS'05*.

[115] http://www.3gpp.org/ftp/Specs/archive.

[116] http://www.nsa.gov/selinux.

[117] http://www.elinux.org/OSK.

[118] http://buildroot.uclibc.org.

[119] http://trolltech.com/products/qtopia.

[120] http://trolltech.com/products/qtopia/qtopiainuse/qtopiadevices.

[121] http://www.pcmag.com/article2/0,4149,1306805,00.asp.

[122] Morein, W., A. Stavrou, D. Cook, A. Keromytis, V. Misra, and D. Rubenstein (2003) "Using graphic turing tests to counter automated ddos attack against web servers," in *ACM CCS'03*.

[123] Ahn, L., M. Blum, N. Hopper, and J. Langford (2003) "CAPTCHA: Using Hard AI Problems for Security," in *EUROCRYPT'03*.

[124] Chellapilla, K. and P. Simard (2004) "Using Machine Learning to Break Visual Human Interaction Proofs (HIPs)," in *Proc. of Neural Information Processing Systems (NIPS'04)*.

[125] Mori, G. and J. Malik (2003) "Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA," in *CVPR 2003*.

[126] Singh, G., C. Estan, and S. Savage (2004) "Automated worm fingerprinting," in *Proc. of OSDI'04*.

[127] http://forum.nokia.com/main/platforms/s60/security.html.

[128] http://en.wikipedia.org/wiki/Linux_framefuffer.

[129] http://sourceforge.net/projects/lmbench/.

[130] http://www.captcha.net/news/ai.html.

[131] Kephard, J. and S. White (May, 1991) "Directed-graph epidemiological models of computer virus," in *Proc. of 1991 Computer Society Symposium on Research in Security and Privacy*.

[132] Constantinos, S., F. Grecas, and I. Venieris (2003) "Introduction of the asymmetric cryptography in GSM, GPRS, UMTS, and its public key infrastructure integration," in *Mobile Network and Applications*.

[133] http://www.f-secure.com/small_businesses/products/fsms.html.

[134] Wang, Y., D. Chakrabarti, C. Wang, and C. Faloutsos (2003) "Epidemic Spreading in Real Networks: An Eigenvalue Viewpoint," in *SRDS*.

[135] http://indigo.ie/˜ mscott/.

[136] Newman, M., S. Forrest, and J. Balthrop (2002) "Email networks and the spread of computer viruses," in *Physical Review*.

[137] Zhou, C., D. Towsley, and W. Gong (2004) "Email Worm Modeling and Defense," in *ICCCN'04*.

[138] Barabasi, A. and R. Albert (Oct., 1999) "Emergence of Scaling in Random Networks," in *Science, pages 509-512*.

[139] Broch, J., D. Maltz, D. Johnson, Y. Hu, and J. Jetcheva (1998) "A Performance Comparison of Multi-hop Wireless Ad-hoc Network Routing Protocols," in *Proc. of MobiCom'98*.

[140] Biddle, P., P.England, M.Peinado, and B. Willman (2002) "The Darknet and the Future of Content Distribution," in *ACM Workshop on Digital Rights Management*.

Vita

Liang Xie

Liang Xie was born on the 4th of August 1971. He received his Bachelor's degree in Electrical Engineering from Soochow University, China in 1993. Before he joined the department of Computer Science and Engineering in Pennsylvania State University in fall 2004, he had been working in telecommunication companies as a software engineer and team manager for around 10 years. His former employers were Huawei Technologies and China Telecom, and his area covered communication protocols and system architectures in both wired and wireless networks.