Early Stage Botnet Detection and Containment via Mathematical Modeling and Prediction of Botnet Propagation Dynamics

Julian Rrushi, Ehsan Mokhtari, and Ali Ghorbani

Faculty of Computer Science
University of New Brunswick
Fredericton, NB, E3B 5A3
Canada

Phone: (506) 453-4566
Fax: (506) 453-3566
Email: fcs@unb.ca
http://www.cs.unb.ca

TR 10-206, December 23, 2010

Abstract

The research that we discuss in this technical report shows that mathematical models of botnet propagation dynamics are a viable means of detecting early stage botnet infections in an enterprise network, and thus an effective tool for containing those botnet infections in a timely fashion. The main idea that underlies this research is to localize weakly connected subgraphs within a graph that models network communications between hosts, consider those subgraphs as representatives of suspected botnets, and thus employ applied statistics to infer the underlying propagation dynamics. The inferred dynamics are materialized into a model graph, which we use within a subgraph isomorphism search process to determine whether or not there is a match between the inferred propagation dynamics and the actual propagation dynamics observed from the weakly connected subgraphs. We conduct modeling based on an intersection of statistics and graph theory such as a match between the two leads to a timely identification of infected hosts. Our mathematical modeling relies on measures of network vulnerability rates, which in this research we estimate via a statistical approach that draws on epidemiological models in biology. That estimation approach is based on random sampling and follows a novel application of statistical learning and inference in a botnet-versus-network setting. We have implemented this overall research in the Matlab and Perl programming languages, and thus have validated its effectiveness in practice in the Emulab network testbed. We have also validated the vulnerability rate estimation approach extensively with respect to realistically simulated botnet propagation dynamics in a GTNetS network simulation platform. In the technical report we describe our overall approach in detail, and thus discuss experiments along with experimental data that are indicative of the effectiveness of our overall approach to detect early stage botnet infections in an enterprise network.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Mathematical models that capture propagation dynamics of botnets, i.e. networks of compromised hosts that are referred to as bots and are controlled by remote attackers who are known as botmasters (Ramachandran *et al.*, 2007), and malware in general have received considerable research attention in the last decade. Nevertheless, the majority of that research focuses on the mathematical models per se and not on possible ways of exploiting those models for network security. Dagon et al. (Dagon *et al.*, 2006) are among the first to propose a practical use of such mathematical models. Dagon et al. employ botnet propagation dynamics predicted via one of such models to comparatively rank botnets, and hence prioritize the responses to those botnets. In this report we propose a novel approach that exploits botnet propagation dynamics to detect bot infections at their very early stage. The ultimate objective behind such early state detection is botnet containment.

The approach that we discuss in this report is applicable to an enterprise network or to a typical network that underlies the backbone routers of an Internet Service Provider (ISP). The tactical design of botnets has been evolving towards higher degrees of sophistication. In (Wang *et al.*, 2010), Wang et al. discuss the design of a hybrid Peer-to-Peer (P2P) botnet that is even more advanced than current real-world botnets. With this result, gaining insight into and predicting the propagation dynamics of highly sophisticated botnets in a timely fashion is crucial to their mitigation. That is a thesis that we prove in this research. More specifically, this research proves that propagation dynamics are a viable means of early stage botnet detection. For that purpose, we focus on one type of bots, namely uniform scanning bots, and thus exploit a mathematical model that captures the propagation dynamics of that type of bot.

There are also many other mathematical models that capture the propagation dynamics of bots that employ other means of target selection. We believe that approaches similar to the one discussed in this report can be devised to work on those bots by exploiting the corresponding mathematical models of propagation dynamics. In terms of implementation we propose this research as a prototype whose effectiveness has been tested in practice. We discuss that effectiveness later on in this report. We wrote bot code and practically analyzed its propagation characteristics and dynamics in the Emulab network testbed (White *et al.*, 2002). We sniffed with tcpdump the network packets that were generated, and thus applied a graph modeling on those packets to create graph structures within what in this report we refer to as a data graph. The data graph is constructed as follows. Each distinct host that participates in network communications as sender or receiver is modeled as a vertex, which is labeled with the Internet Protocol (IP) address of that host.

Each edge directed from one vertex towards another vertex denotes a network communication in which the host represented by the former vertex acts as source, while the host represented by the latter vertex acts as destination. Each one of such directed edges is assigned pairs of two attributes, namely a timestamp that indicates the time at which the network communication took place along with a string that indicates the specific network service that the network communication in question relates to. When conducting a botnet detection and containment task in a real-world network, we apply such graph modeling on all packets as we do not know a priori which packets are generated by bots. Thus, the data graph comprises vertices and edges that represent network communications which may be legitimate or malicious. In various runs in the Emulab network testbed we noticed that bot infections along with bot infection attempts create weakly connected subgraphs within the data graph.

Thus, replacing all of the directed edges in each such subgraph with undirected edges produces a connected undirected subgraph, i.e. for every pair of distinct vertices in that undirected subgraph there exists a path from one to the other. Our finding is to some degree similar to the finding of Ellis et al. (Ellis *et al.*, 2004), according to which worm network behavior is characterized by tree-like propagation and reconnaissance. We consider weakly connected subgraphs within the data graph as suspected botnets, and hence the network services in them as suspected vulnerable services. We then statistically infer the propagation dynamics that underlies each one of such weakly connected subgraphs. An error-tolerant match between the inferred dynamics and the actual dynamics observed on the network enables us to determine in a timely fashion which hosts are infected, and thus take action to filter network accesses to vulnerable network services in other hosts from those vulnerable hosts.

The various mathematical models developed so far that capture the propagation dynamics of botnets and self-replicating malware in general, i.e. viruses, worms, etc., depend directly or indirectly on measures of network infection rates and network susceptibility, i.e. vulnerability, rates, which are often referred to as malware virulence. This research shares that characteristic as it relies on measures of network vulnerability rates. In that regard, we have devised a statistical approach to vulnerability rate estimation that draws on epidemiological models in biology. Few other research works have investigated the problem of estimating the vulnerability rate in an enterprise network. Choi et al. (Choi *et al.*, 2010) apply maximum likelihood estimation (MLE) (Fisher, 1922) in an approach that estimates the size of vulnerable host population in a local or enterprize network in relation to a worm.

The estimation approach of Choi et al. exploits a deterministic relation between the number of vulnerable hosts and the mean inter-arrival time between successive host infections. As such, the estimation approach of Choi et al. relies on observation of host infections as those infections take place. In our case we do not know a priori which computer network communications result in host infections, consequently we cannot obtain the mean inter-arrival time between successive host infections. In fact detecting those host infections at an early stage is exactly the ultimate objective of this research. Our statistical approach to vulnerability rate estimation also uses MLE in part. Nevertheless, it has no reliance on knowing host infections in progress a priori, and thus can provide network vulnerability rates that are usable within our overall approach.

Chapter 2

Initial Modeling

We now discuss how we leverage a specific mathematical model to infer the propagation dynamics of a suspected botnet in an enterprise network. Various research in the area of epidemiology developed mathematical models that capture the propagation of infectious diseases among humans (Anderson *et al.*, 1992; Daley and Gani, 1999; Andersson and Britton, 2000). The high degree of similarity in terms of propagation between biological viruses and computer malware such as bots and worms was a clear suggestion of the applicability of those mathematical models to computer malware. The adoption of the mathematical models in question in computer science research led to two main categories of formalisms that represent malware propagation dynamics, namely deterministic models (Staniford *et al.*, 2002; Chen *et al.*, 2003; Dagon *et al.*, 2006; Chen and Ji, 2005; Fan and Xiang, 2010) and stochastic models (Rohloff and Basar, 2005; Sellke *et al.*, 2008). Deterministic models perform well in modeling the dynamics of malware spread when the number of infected hosts is large (Sellke *et al.*, 2008), and thus do not produce workable results when the number of infections is small.

Consequently deterministic models are not applicable to this research, namely because we seek to detect and contain bot infections at their very early stage in an enterprise network. Thus, unlike most of the related research discussed later on in this report, our approach is applicable from the first few bot infections, and as such it relies on characterization of the stochastic evolution of the botnet infection process. Stochastic models provide better accuracy in the case malware propagation is targeted at its early stage (Liljenstam *et al.*, 2003). Epidemic models in general capture mean behavior. A key factor that determines how much accurate the captured behavior is consists of coping with the variability around that mean. The variability in question is very high during the early stage of infections (Sellke *et al.*, 2008). Given that the botnet infection process per se is probabilistic, a stochastic model naturally has better means of coping with such high variability.

In this research we apply a stochastic model developed from research in the field of epidemiology, which is discussed in (Andersson and Britton, 2000; Daley and Gani, 1999). The model was adapted to capture computer worm behavior by

Rohloff and Basar in (Rohloff and Basar, 2005). Let γ denote the number of hosts in the enterprise network, and φ the number of hosts that were originally vulnerable just before the enterprise network was reached by bot infections. Thus, φ denotes the number of hosts that could potentially get infected by bots. Let us denote with β the bot infection parameter, namely the average rate at which a bot initiates infection attempts against hosts in the enterprise network. In this research with j we denote the possible sizes of the infected host population throughout the infectious time. Thus, the values of j lie in $\{1, 2, ..., \varphi\}$. The infectious time represents the duration of the botnet infection process in the enterprise network.

It is the amount of time that elapses between the moment in which the botnet initiates the first infection attempt against the enterprise network and the moment in which $j = \varphi$, namely the moment in which the original vulnerable host population becomes exhausted and the infected host population reaches its maximum possible size. The botnet infection process in the enterprise network is modeled as a Markov jump process. Each state of that process, which we denote with I_j, represents a stage in which the botnet infection has progressed to j infected hosts. Thus, the states of the botnet infection process are $I_1, I_2, ..., I_\varphi$. Let T_j denote the amount of time that the botnet infection process is in state I_j. T_j is a random variable whose mean or expected value $E(T_j)$ according to the conclusions provided in (Rohloff and Basar, 2005; Andersson and Britton, 2000; Daley and Gani, 1999) is given by the following equation:

$$E(T_j) = \frac{\gamma}{\beta(\varphi - j)j} \tag{2.1}$$

The practicality of the stochastic model in the form of Equation 2.1 is limited when applied to enterprise networks of medium or large sizes, and thus the model in that form is not employable in inferring the dynamics of a real-world suspected botnet in those networks. Although not expressed explicitly in Equation 2.1, γ refers to the size of the online host population in the enterprise network. Clearly offline hosts do not count as they do not play any role in the botnet infection process. A straightforward way of discovering γ would have been to send Internet Control Message Protocol (ICMP) echo request packets to all IP addresses allocated for use in the enterprise network, and thus merely count the number of hosts that respond with an ICMP echo reply packet. Nevertheless, ICMP may not be enabled in many hosts, and furthermore it is not rare for packet filter firewalls to drop ICMP messages in transit because of network security reasons.

Host firewalls also may block ICMP messages. For example, in its default settings the Windows 7 firewall blocks all incoming ICMP echo request packets. Knowledge of the overall number of IP addresses allocated statically or dynamically for use in the enterprise network does not imply knowledge of γ. Some computers are turned off at evening or night, or even during the day depending on the work schedules or habits of the corresponding users. Predicting whether or not a computer user will turn of his/her computer along with the time in which a possible computer turn off will take place and the time frame during which the computer will remain turned off is not realistic. The use of laptop computers is quite common in enterprise networks. Laptop computers were designed for mobile use, and thus by definition imply varying presence in the enterprise network. Computer users are often away from their habitual place of connection to the enterprise network due to business trips or holidays, and thus their computers might possibly be disconnected for the duration of those events.

Arguably all of these phenomena along with others that cause random variability in the size of the online host population in the enterprise network are not feasibly predictable. The same discussion holds for φ, which in Equation 2.1 refers to the size of the online vulnerable host population at the beginning of the botnet infection process in the enterprise network. The infeasibility of predicting the phenomena previously mentioned prohibits us from having a workable estimate of φ. Dagon et al. (Dagon *et al.*, 2006) approximate φ as a periodical function of time with a period of 24 hours, which the authors refer to as a diurnal shaping function, in relation to the computer networks in an entire time zone. Dagon et al. infer diurnal shaping functions for specific past botnets empirically from botnet data gathered via a sinkhole over a period of 6 months. Those diurnal shaping functions are reusable in the case the mitigation efforts target a botnet that exploits the same specific vulnerabilities as those botnets.

The approximation of φ to a periodical function of time is suitable for use in a deterministic model such as the time zone-based botnet propagation model developed by Dagon et al. in (Dagon *et al.*, 2006). That is because such model is not concerned with the variability around mean behavior, as we wrote previously in this chapter. Taking into account that the variability in question is not negligible in our case, in this research we consider φ as a random function of time, i.e. the variation of the size of the online vulnerable host population over time is random. Dagon et al. infer φ from a post-mortem study, while we would need to have φ at hand shortly after the beginning of the botnet infection process in the enterprise network. Furthermore, we aim at employing a defensive approach that is applicable to any botnet, regardless of whether or not that botnet exploits the same vulnerabilities as past botnets that were subject to post-mortem analysis.

A brute force approach to discovering φ would require port scanning of each individual online host in the enterprise network to determine whether or not that host runs network services that we suspect are vulnerable to the suspected botnet. Given that an enterprise network of medium or large size comprises hundreds or thousands of hosts, respectively, a brute force approach is too costly in terms of time from a botnet containment perspective. In this research we have found a means of neutralizing the requirement of knowing γ and φ for being able to employ in practice the stochastic model represented by Equation 2.1. We can observe in Equation 2.1 that it indirectly incorporates a measure of the vulnerability rate that corresponds to state I_j of the botnet infection process. With vulnerability rate we mean the proportion of online hosts in the enterprise network that run network services exploitable by the various exploits implemented in the bot codebase.

In this report we denote that vulnerability rate with δ_j. In the denominator,

$(\varphi - j)$ is the number of online vulnerable hosts that could get infected by bots but are not yet infected by bots at a stage in which the botnet infection process is in state I_j. According to the definition of vulnerability rate previously mentioned we have that $\delta_j = \frac{\varphi - j}{\gamma}$, consequently we can conclude that $\frac{\gamma}{\varphi - j}$ in Equation 2.1 is exactly δ_j^{-1}. With this observation, the stochastic model takes the following form:

$$E(T_j) = \frac{1}{\beta \delta_j j} \tag{2.2}$$

In the following chapter we show how we estimate δ_j without any prior knowledge of φ and γ. Later on in this report we show how we use vulnerability rate estimates to derive the possible values of j, and thus acquire the ability to employ in practice the stochastic model represented by Equations 2.1 and 2.2.

Chapter 3

Vulnerability Rate Estimation

3.1 Basic Approach

Our approach to vulnerability rate estimation draws on the epidemiological models discussed in (Bhattacharyya *et al.*, 1979; Chiang and Reeves, 1962; Thompson, 1962; Walter *et al.*, 1980). In this chapter, let us denote the vulnerability rate with the generic term δ as our estimation approach is not dependent on the states of the botnet infection process. We use a discrete uniform distribution of the IP address space allocated to the enterprise network to randomly generate a sample of IP addresses of hosts from that network. We then organize the IP addresses in the sample into pools. The assignment of the IP addresses in the sample to specific pools is performed in a random fashion. That random assignment creates pools with various sizes within the sample. With size of a pool we mean the number of IP addresses in the sample that have been assigned to that specific pool. We consider an IP address as vulnerable if we find that the corresponding host runs network services that are presumably vulnerable to the suspected botnet.

We consider a pool as positive if that pool contains one or more vulnerable IP addresses, and negative otherwise. For each pool we need to determine whether or not it is a positive pool. We do so by checking the IP addresses in a pool in a random order. Once an IP address in a pool under examination is found to be vulnerable, the inspection of that pool is considered complete and that pool is marked as positive. Otherwise, the inspection proceeds with checks on the other IP addresses in the pool in question. If at the end of the inspection process no IP addresses are found to be vulnerable, the pool under examination is marked as negative. Let us consider a data vector $x = (x_1, x_2, ..., x_m)$, in which x_i is a random variable that denotes the number of positive pools of size i $\forall i \in \{1, 2, ..., m\}$. In the remaining of this chapter, when we refer to i either directly or indirectly we mean $\forall i \in \{1, 2, ..., m\}$.

Let us denote with f and f_i the probability density functions of the data vector x and x_i, respectively. Testing whether or not a pool of size i is positive is a Bernoulli trial as the probability that a pool of size i is positive, which in this report we denote with ε_i, remains constant in all inspections of pools of size i. Thus, the inspections or tests of all pools of size i form a Bernoulli process. We can notice that in such Bernoulli process the random variable x_i follows a binomial model. That is due to fact that the Bernoulli trials on pools of size i are independent of one another, and ε_i remains constant from Bernoulli trial to Bernoulli trial as we wrote previously. In this report with model we mean a parametric family of probability density functions that are indexed by specific parameters. Let $n = (n_1, n_2, ..., n_m)$ be a data vector whose elements n_i are counts of pools of size i in the sample.

We have that $x_i \sim \text{Binom}(n_i, \varepsilon_i)$, and hence f_i is a binomial distribution that can be expressed as follows:

$$f_i(x_i \mid n_i, \varepsilon_i) \tag{3.1}$$

We can observe from Equation 3.1 that the family of binomial distributions followed by x_i is indexed by parameters n_i and ε_i. Note that we obtain parameter n_i from the data in our sample. We now estimate the probability that a pool of size i is positive, namely ε_i. Given that the vulnerability rate of the overall enterprise network is δ, the probability that an IP address in a pool of size i is vulnerable is also δ. Consequently the probability that an IP address in a pool of size i is invulnerable is $1 - \delta$. The probability that all the i IP addresses in a pool of size i are invulnerable is $(1 - \delta)^i$. The probability that not all the i IP addresses in a pool of size i are invulnerable, which corresponds to the probability that a pool of size i is positive, is given by the following equation:

$$\varepsilon_i = 1 - (1 - \delta)^i \tag{3.2}$$

Equation 3.2 also shows that the probability that a pool of size i is positive remains constant over all the Bernoulli trials on the n_i pools of size i, which is a statement that we made earlier in this chapter. That is because clearly both δ and i remain invariable throughout the Bernoulli trials in question, and therefore produce in Equation 3.2 a ε_i that also remains invariable throughout those Bernoulli trials. Given that the random variable x_i follows a binomial model, we can apply the formula for the binomial distribution to estimate the probability of x_i pools of size i in the sample being positive out of n_i such pools as shown below:

$$f(x_i \mid n_i, \varepsilon_i) = \begin{pmatrix} n_i \\ x_i \end{pmatrix} \varepsilon_i^{x_i} (1 - \varepsilon_i)^{n_i - x_i} \tag{3.3}$$

Let us briefly go through the arguments for the validity of Equation 3.3. Let Ω be a set that comprises all possible x_i-size subsets of the set of n_i pools of size i. Thus, each element of Ω is a subset that comprises x_i pools of size i. The probability that x_i out of n_i pools of size i are positive is a *logical AND* between: (a) the probability that any element of Ω comprises pools of size i that are all positive; and (b) the probability that the remaining $n_i - x_i$ pools of size i are negative. Clearly the probability that all pools of size i in an element of Ω are positive is $\varepsilon_i^{x_i}$. Thus, the contribution that an element of Ω makes to the probability that any element of Ω comprises pools of size i that are all positive is $\varepsilon_i^{x_i}$. Given that the total number of contributors is equal to the cardinality of Ω, then the total contribution, i.e. the probability that any element of Ω comprises pools of size i that are all positive, is equal to $\varepsilon_i^{x_i}$ times the cardinality of Ω.

The binomial coefficient in Equation 3.3 calculates the cardinality of Ω. The probability that a pool of size i is negative amounts to $1 - \varepsilon_i$. Consequently the probability that the remaining $n_i - x_i$ pools of size i are negative is $(1-\varepsilon_i)^{n_i - x_i}$. The specification of such probability concludes Equation 3.3. By plugging Equation 3.2 into Equation 3.3 we get the following equation:

$$f(x_i \mid n_i, \varepsilon_i) = \begin{pmatrix} n_i \\ x_i \end{pmatrix} (1-(1-\delta)^i)^{x_i}(1-\delta)^{i(n_i-x_i)} \tag{3.4}$$

As n_i lies in n, and as from Equation 3.2 we can derive that ε_i is related to δ by a fixed scaling constant, let us express the binomial distribution f_i in the following form, which is equivalent to Equation 3.1:

$$f_i(x_i \mid n, \delta) \tag{3.5}$$

For any l and k such that $l, k \in \{1, 2, ..., m\}$ and $l \neq k$, x_l and x_k are statistically independent of one another. For that reason, f can be expressed as a multiplication of all f_i defined over $\forall i \in \{1, 2, ..., m\}$ as shown in Equation 3.5. Thus, the distribution of the data vector x is formulated as follows:

$$f(x = (x_1, x_2, ..., x_m) \mid n, \delta) = \prod_{i=1}^{m} f_i(x_i \mid n, \delta) \tag{3.6}$$

Solving for the binomial coefficient in Equation 3.4, and thereafter plugging Equation 3.4 into Equation 3.6 gives us a more detailed formulation of the distribution of the data vector x:

$$f(x \mid n, \delta) = \\ \prod_{i=1}^{m} \left(\frac{n_i!}{x_i!(n_i - x_i)!} (1-(1-\delta)^i)^{x_i}(1-\delta)^{i(n_i-x_i)} \right) \tag{3.7}$$

The model represented by Equation 3.7 comprises a family of probability density functions that are indexed by δ, given that data vectors x and n lie in the data in our sample. Different values of parameter δ produce different probability density functions for data vector x. We are interested in that specific probability density function for data vector x that is most likely to have produced the data in our sample. For that purpose, we analyze the statistical relation between Equation 3.7 and the data in our sample through the lens of MLE (Fisher, 1922). Let the likelihood function $L(\delta \mid n, x)$ denote the likelihood of parameter δ given the data vectors n and x, which we can observe from the pools of IP addresses. The likelihood function $L(\delta \mid n, x)$ is a function of parameter δ, therefore a specific value of parameter

δ fed to that likelihood function results in a specific likelihood or unnormalized probability of that specific value of δ itself.

Thus, different values of parameter δ have different likelihoods. Finding the specific probability density function for data vector x in the model represented by Equation 3.7 that is more likely to be at the origin of the data in our sample, is equivalent to finding a specific value of parameter δ that maximizes the likelihood function $L(\delta \mid n, x)$, i.e. finding the specific value of parameter δ that has the highest likelihood given data vectors n and x. From probability theory we derive that the likelihood function $L(\delta \mid n, x)$ can be defined as in the equation below:

$$L(\delta \mid n, x) = f(x \mid n, \delta) \tag{3.8}$$

By plugging Equation 3.7 into Equation 3.8 we reach the following more detailed formulation of our likelihood function:

$$L(\delta \mid n, x) = \prod_{i=1}^{m} \left(\frac{n_i!}{x_i!(n_i - x_i)!} (1 - (1 - \delta)^i)^{x_i} (1 - \delta)^{i(n_i - x_i)} \right) \tag{3.9}$$

Maximizing the likelihood function $L(\delta \mid n, x)$ is equivalent to maximizing the log-likelihood function $ln\ L(\delta \mid n, x)$ as those two functions are monotonically related to one another. We work with the log-likelihood function rather than the likelihood function mainly because of computational convenience. By applying the natural logarithm to both sides of Equation 3.9 we obtain the log-likelihood function as shown below:

$$ln\ L(\delta \mid n, x) = \sum_{i=1}^{m} ln \left(\frac{n_i!}{x_i!(n_i - x_i)!} (1 - (1 - \delta)^i)^{x_i} (1 - \delta)^{i(n_i - x_i)} \right) \tag{3.10}$$

By applying in Equation 3.10 the transformation rule that regards the logarithm of products, we reach the following form of the log-likelihood function:

$$ln\ L(\delta \mid n, x) = \sum_{i=1}^{m} (ln(\frac{n_i!}{x_i!(n_i - x_i)!}) + ln((1 - (1 - \delta)^i)^{x_i}) + ln((1 - \delta)^{i(n_i - x_i)})) \tag{3.11}$$

In Equation 3.11 we apply the transformation rule that regards the logarithm of powers, and thus obtain the following refinement of the log-likelihood function:

$$\begin{aligned} ln\ L(\delta \mid n, x) = \sum_{i=1}^{m} (ln(n_i!) - ln(x_i!) - ln((n_i - x_i)!) + \\ + x_i\ ln(1 - (1 - \delta)^i) + i(n_i - x_i)\ ln(1 - \delta)) \quad (3.12) \end{aligned}$$

We can omit the first three terms of Equation 3.12, namely $ln(n_i!) - ln(x_i!) - ln((n_i - x_i)!)$, as they do not depend on parameter δ, and therefore have no effects on the specific value of parameter δ that maximizes the log-likelihood function $ln\ L(\delta \mid n, x)$. According to the MLE principle, an existing value of parameter δ that maximizes the log-likelihood function meets the requirement that the first derivative of the log-likelihood function should be equal to zero, as shown in the likelihood equation given below:

$$\frac{d\ ln\ L(\delta \mid n, x)}{d\delta} = 0 \quad (3.13)$$

This likelihood equation is an ordinary differential equation as the parameter vector that maximizes the log-likelihood function contains only one element, namely parameter δ. By plugging the pertinent part of Equation 3.12 into Equation 3.13 we get a formulation of the likelihood equation as the following ordinary differential equation:

$$\frac{d\ \sum_{i=1}^{m}(x_i\ ln(1 - (1 - \delta)^i) + i(n_i - x_i)\ ln(1 - \delta))}{d\delta} = 0 \quad (3.14)$$

The estimate of parameter δ is obtained by solving the ordinary differential equation 3.14. The requirement represented by the likelihood equation in 3.13 and hence in 3.14 is due to the fact that the maximum or minimum of the log-likelihood function $ln\ L(\delta \mid n, x)$ by definition imply that its first derivative converges to zero at the values of parameter δ that maximize or minimize the log-likelihood function in question. This means that the estimate of δ that we have found may be a minimum of the log-likelihood function $ln\ L(\delta \mid n, x)$, i.e. it minimizes that log-likelihood function instead of maximizing it. Always according to the MLE principle, we can validate that our estimate of parameter δ maximizes the log-likelihood function $ln\ L(\delta \mid n, x)$ by checking that its second derivative evaluated at our estimate of parameter δ is negative.

Thus, for our estimate of δ to be a valid measure of vulnerability rate, its value along with data vectors n and x observed from the data in our sample should satisfy the equation below:

$$\frac{d^2\ \sum_{i=1}^{m}(x_i\ ln(1 - (1 - \delta)^i) + i(n_i - x_i)\ ln(1 - \delta))}{d\delta^2} < 0 \quad (3.15)$$

If that is not the case, we search the parameter space further to identify a suitable estimate of δ that indeed maximizes the log-likelihood function $ln\ L(\delta \mid n, x)$.

3.2 A Logit Model for Finding Optimal Estimation Parameters

During practical experiments with the proposed estimation approach we observed that for each pair of sample size and network size values, the level of precision of the approach exhibits dependence on the number of pools. Most numbers of pools produced workable estimates of vulnerability rates with varying estimation errors. Nevertheless, there were numbers of pools that resulted in estimation errors which were so high that the corresponding vulnerability rate estimates were quite unusable in models that predict botnet propagation dynamics. We summarize in Figure 3.1 various estimations as produced by experiments in which the dependency in question emerged most evidently.

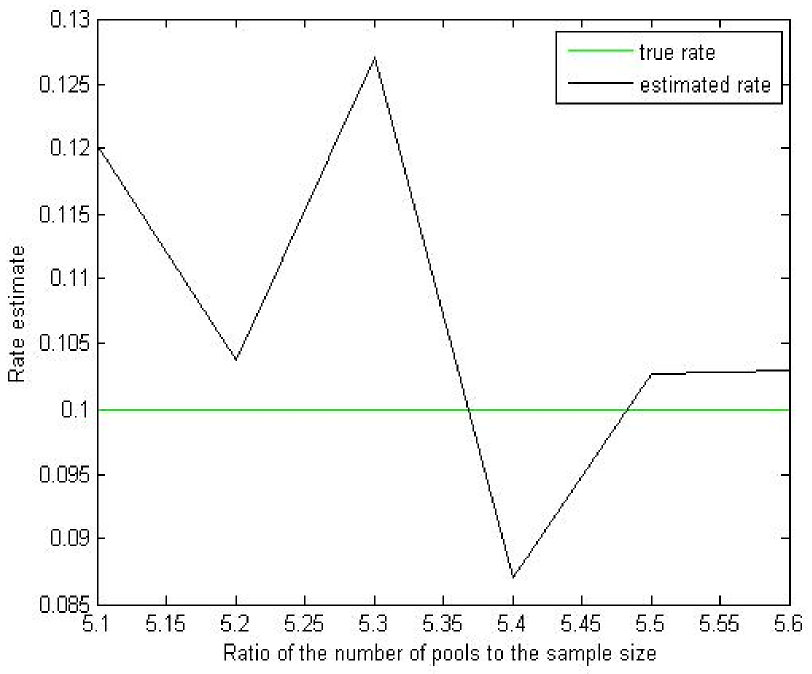


Figure 3.1: Plot of true vulnerability rate versus estimated vulnerability rate for various ratios of the number of pools to the sample size with no engagement of the logit model.

In those experiments we employed a ratio of the sample size to the network size of 1:1000. Figure 3.1 depicts a graph that compares the true rate with a series of estimated rates for various ratios of the number of pools to the sample size. We can notice in Figure 3.1 that values succeeding 1:5.48 are optimal ratios of the number of pools to the sample size, while values such as 1:5.4 or 1:5.3 certainly are not. Let z_1 and z_2 denote the sample size and the network size, respectively, and let τ

denote the number of pools. Clearly $\tau = \sum_{i=0}^{m} n_i$, in which n_i and m are defined as in the previous discussion, namely n_i denotes the number of pools of size i and m is the largest pool size observed after the random assignment of the nodes in the data sample to pools.

Let r and q denote a minimum value and a maximum value for τ, respectively. In the experiments that we discuss later on in this report we tried ratios of the number of pools to the sample size from 1:1 to 1:10, which means that we worked with numbers of pools from $r = \frac{z_1}{10}$ to $q = z_1$. Note that there is no statistical significance behind our selection of those specific numbers of pools. Thus, defenders can employ a different range of numbers of pools according to the sample size that their botnet membership detectors and/or passive vulnerability scanners can process in a timely fashion in their network of reference. We seek to identify those numbers of pools among any $\{r, r+1, r+2, ..., q\}$ which are optimal, i.e. produce workable estimates of vulnerability rates for any pair of sample size and network size values (z_1, z_2).

We address such research problem by developing a logit model described herein that calculates the probability distribution of τ with respect to the capability to produce workable estimates of vulnerability rates. After we obtain that probability distribution, we mark the values of τ with the highest probabilities as optimal numbers of pools for the given $(z_1,\ z_2)$. The remaining of our discussion in this section is conducted within the context of applied logistic regression (Hosmer and Lemeshow, 2000; Kleinbaum *et al.*, 2007). We model τ as a dependent variable, and z_1 and z_2 as exposure variables. We can notice that the possible values of τ are ordered. If we consider each possible value of τ as an outcome category, then the outcome categories or our logit model are ordered and therefore the type of logistic regression that applies to our research problem is the ordinal logistic regression.

Given that we are dealing with two exposure variables, our logit model has two coefficient terms, which in this report we refer to as β_1 and β_2. We associate those two coefficient terms with z_1 and z_2, respectively. There are $q - r$ comparisons between the possible values of τ, consequently our logit model has $q - r$ intercept terms, namely $\alpha_1, \alpha_2, ...\alpha_{q-r}$. The intercept term α_1 corresponds to $r + 1$, the intercept term α_2 corresponds to $r + 2$, and so on. No intercept term α_0 corresponds to r. Let us have $h \in \{0, 1, ..., q - r\}$. The probability that any value of τ equal to or greater than $r + h$ is optimal is given by the following equation:

$$P(\tau \geq r + h \mid z_1, z_2) = \frac{1}{1 + e^{-(\alpha_h + \beta_1 z_1 + \beta_2 z_2)}} \tag{3.16}$$

Similarly, the probability that any value of τ equal to or greater than $r + h + 1$ is optimal is provided by the equation below:

$$P(\tau \geq r + h + 1 \mid z_1, z_2) = \frac{1}{1 + e^{-(\alpha_{h+1} + \beta_1 z_1 + \beta_2 z_2)}} \tag{3.17}$$

Thus, the probability that any value of τ equal to $r + h$ is optimal is:

$$P(\tau = r+h \mid z_1, z_2) \;=\; P(\tau \geq r+h \mid z_1, z_2) \;-\; P(\tau \geq r+h+1 \mid z_1, z_2) \quad (3.18)$$

By plugging Equations 3.16 and 3.17 into Equation 3.18 we obtain the probability of any value $r+h$ of τ being optimal, namely:

$$P(\tau = r+h \mid z_1, z_2) \;=\; \frac{1}{1+e^{-(\alpha_h+\beta_1 z_1+\beta_2 z_2)}} - \frac{1}{1+e^{-(\alpha_{h+1}+\beta_1 z_1+\beta_2 z_2)}} \quad (3.19)$$

The following special cases apply to our logit model. If h is equal to zero, Equation 3.16 is not applicable as there is no intercept term α_0 that corresponds to r. Intuitively, we know that $P(\tau \geq r \mid z_1, z_2) = 1$, consequently Equation 3.19 takes the following form:

$$P(\tau = r \mid z_1, z_2) = 1 - \frac{1}{1+e^{-(\alpha_1+\beta_1 z_1+\beta_2 z_2)}} \quad (3.20)$$

If h is equal to $q-r$, then Equation 3.17 is not applicable as there is no intercept term α_{q-r+1} that corresponds to the non-possible value $q+1$. We know intuitively that $P(\tau \geq q+1 \mid z_1, z_2) = 0$, therefore in this other case Equation 3.19 takes the following form:

$$P(\tau = q \mid z_1, z_2) \;=\; \frac{1}{1+e^{-(\alpha_{q-r}+\beta_1 z_1+\beta_2 z_2)}} \quad (3.21)$$

The input of our logit model comprises z_1 and z_2 only. Thus, for being able to use our logit model, we need estimates of the intercept terms and coefficient terms embedded in that model. We obtain those estimates by conducting statistical learning over data which comprise numbers of pools that are known to be optimal for various sample sizes and network sizes. We obtained those learning data from the experiments that we discuss later on in this technical report. In those experiments the true network vulnerability rates were known to us, therefore we could tell which numbers of pools appeared to be optimal for specific sample sizes and network sizes. The learning data that we constructed from the experiments in question took the form of those presented in Table 3.1. Defenders can easily construct similar data by replaying the experiments in question with numbers of pools, sample sizes, and network sizes of their choice.

We seek to identify those specific intercept terms and coefficient terms that make the learning data more likely. If u denotes the number of learning data records, let us number those records from 1 to u. For $v \in \{1, 2, ..., u\}$, let w_{vh} be a flag variable defined as follows:

$$w_{vh} = \begin{cases} 1 & \text{If in the } v\text{-th row, } \tau = r+h \\ 0 & \text{If in the } v\text{-th row, } \tau \neq r+h \end{cases} \quad (3.22)$$

Table 3.1: Excerpt from the statistical learning data

Optimal Number of Pools	Sample Size	Network Size
72	108	108418
47	142	113920
95	143	100388
121	181	109148
23	231	115824

The likelihood of the learning data is given by the following joint probability:

$$L_{data} = \prod_{v=1}^{u} \prod_{h=0}^{q-r} \left[P\left(\tau = r + h \mid z_1, z_2\right) \right]^{y_{vh}} \tag{3.23}$$

In Equation 3.23, L_{data} is a function that returns the likelihood of the learning data. By plugging Equation 3.19 into Equation 3.23 we obtain a more elaborate definition of the likelihood of the learning data:

$$L_{data} \quad = \quad \prod_{v=1}^{u} \prod_{h=0}^{q-r} \left[\frac{1}{1 + e^{-(\alpha_h + \beta_1 z_1 + \beta_2 z_2)}} - \frac{1}{1 + e^{-(\alpha_{h+1} + \beta_1 z_1 + \beta_2 z_2)}} \right]^{w_{vh}} \tag{3.24}$$

The values of z_1 and z_2 in Equation 3.24 are available from the learning data records. After performing the multiplications of the probabilities contributed by each individual learning data record in Equation 3.24, we obtain L_{data} as a function of the intercept terms and coefficient terms, namely $L_{data}(\alpha_1, \alpha_2, ..., \alpha_{q-r}, \beta_1, \beta_2)$. The estimates of the intercept terms and coefficient terms that we are looking for consist of those specific values of the terms in question that maximize L_{data}, which we now identify via the MLE principle. Let us organize the intercept terms and coefficient terms in L_{data} as a parameter vector $\theta = \left(\theta_1, \theta_2, ..., \theta_{q-r+2}\right)$. $\theta_1, \theta_2, ..., \theta_{q-r}$ correspond to intercept terms $\alpha_1, \alpha_2, ..., \alpha_{q-r}$, while $\theta_{q-r+1}, \theta_{q-r+2}$ correspond to coefficient terms β_1, β_2. For $i \in \{1, 2, ..., q-r+2\}$, the estimates that we are seeking are the solutions of a system of partial differential equations of the following form:

$$\frac{\partial ln\left[L_{data}\left(\theta\right)\right]}{\partial \theta_i} = 0 \tag{3.25}$$

Placing those estimates in Equations 3.19, 3.20, and 3.21, makes our logit model ready to use for finding optimal numbers of pools. Armed with an estimate of δ, we are now in the conditions of exploiting the stochastic model represented by Equations 2.1 and 2.2.

Chapter 4

Model Graph Generation

The dynamics of a suspected botnet that we infer statistically are materialized in a directed attributed graph, which in this research we refer to as a model graph. The vertices of the model graph represent hosts in the enterprise network that are infected by bots. Each vertex is labeled with the infected population size reached when the host represented by that vertex is infected by a bot, namely j, for $j \in \{1, 2, ..., \varphi\}$. Thus, a vertex with label j represents the host whose infection causes the botnet infection process to enter state I_j. In this research we refer to those vertex labels as infected host numbers or bot numbers. It is a mapping between infected host numbers and IP addresses observed in concrete network communications in the enterprise network that leads us to identification of infected hosts. We discuss that mapping process later on in this discussion. In the model graph, each directed edge from an origin vertex to a destination vertex indicates infection of the host that is represented by the destination vertex.

That host infection is initiated by a bot running on the host represented by the origin vertex. Each edge has an attribute that indicates the mean relative time in minutes at which the corresponding host infection takes place in the enterprise network. The time is considered relative to the infection of the first host in the enterprise network. Thus, the mean infection time of infected host 1 is 0.0 minutes. An example of a model graph is shown on the left part of Figure 4.1. That example model graph was generated in relation to bots that initiate infection attempts at an average rate of 2 per minute in a network of 10 hosts, 5 of which are vulnerable to those bots. Model graph generation is mostly translated into estimating the number of offsprings of each possible vertex, determining which vertex is parent of what other vertex or vertices, and estimating the mean relative time at which bot infections as modeled by concrete directed edges take place.

Recall from previous chapters that a suspected botnet is represented by a weakly connected subgraph within the data graph, and that the network services suspected to be vulnerable are those indicated on the edges of that weakly connected subgraph. Let us denote with v the size of the infected host population at the moment in which our approach begins processing a weakly connected subgraph formed within the data graph. Firstly, we estimate the vulnerability rate that applied to the enterprise network before that network was reached by bot infections. Let that estimate be denoted by δ_0. At that point in time all possibly infected hosts currently represented in the weakly connected subgraph were still vulnerable and thus uninfected. For that reason, we ignore the weakly connected subgraph when calculating that estimate. More precisely, when determining whether an IP address in a pool within a random sample is vulnerable, we port scan the host with that IP to check if it runs any of the network services that are suspected to be vulnerable.

If that is the case, we conclude that the IP address in question is vulnerable regardless of the fact that the corresponding host might be represented in the weakly connected subgraph, and thus presently might be infected. Besides the consideration just discussed, estimation of δ_0 proceeds as described previously. Secondly, we leverage the data in the sample used to estimate δ_0 to determine the portion of δ_0 that is due to 1 vulnerable IP address. In other words, we estimate the contribution that a single vulnerable IP address makes to δ_0. In the sample in question, we select randomly an IP address that was found to be vulnerable during the estimation of δ_0. We then mark that IP address as infected, and thus invulnerable, and thereafter repeat the estimation procedure from that point on, i.e. randomly organize the IP addresses in the sample into pools, determine which pools are positive, etc.

In the above discussion we are making the assumption that an infected host does not get reinfected as most contemporary botnets patch the exploited vulnerabilities or make those vulnerabilities unexploitable in order to prevent other competing botnets from taking over hosts already under their control. With the intervention previously discussed we obtain the equivalent of the estimation process that produced δ_0 as conducted when the botnet infection process is in state I_1. Once we obtain an estimate of δ_1, we conclude that the contribution that a single vulnerable IP address makes to δ_0 is $\delta_0 - \delta_1$. Consequently we have that:

$$\varphi = \frac{\delta_0}{\delta_0 - \delta_1} \Rightarrow j \in \{1, 2, ..., \frac{\delta_0}{\delta_0 - \delta_1}\} \tag{4.1}$$

The number of vulnerable hosts in the enterprise network that we can save from infection is about $\frac{\delta_0}{\delta_0 - \delta_1} - \nu$. The vulnerability rate at any state I_j of the botnet infection process and the size of the online host population in the enterprise network take the following form:

$$\delta_j = \delta_0 - j\delta_1 \Rightarrow \gamma = \frac{\delta_0 - j(\delta_0 - \delta_1)}{\delta_j(\delta_0 - \delta_1)} \tag{4.2}$$

We now have at hand all the necessary elements to build the model graph. A pertinent part of the model graph generation algorithm is given below. The model graph generated by the algorithm applies to the entire infectious time, and thus is usable for any $\nu < \varphi$. The definition of various variables used by the algorithm that have not been covered in our previous and current discussions is provided in Table 4.1. Due to reasons that become clear later on in this report, vectors and matrices used in the algorithm are considered as in the Matlab programming language, namely the first element of each vector starts at index 1 rather than 0, and the first element of each matrix starts at (row, column) indices $(1, 1)$ rather than $(0, 0)$. When elaborating on the algorithm, when necessary we refer to the example model graph shown in Figure 4.1 along with the corresponding example botnet versus network setting.

Estimation of φ and hence of the possible values of j provides us with a definition of the possible infected host numbers in the model graph. Thus, at this point we know that the possible infected host numbers are $\{1, 2, ..., \varphi\}$. The mean time at which infection of the host represented by the vertex labeled with infected host number j takes place, for $j \neq 1$, is equal to the sum of the mean amount of time that it took the botnet infection process to reach state I_{j-1} and the mean amount of time that the botnet infection process remains in state I_{j-1}, namely $E(T_{j-1})$. Given that state I_1 is reached at time 0.0, infected host number 2 is created at relative time $E(T_1)$. According to our previous postulate, infected host number 3 is created at relative time $E(T_1)+E(T_2)$. In general, infected host number j, for $j \neq 1$, is created at relative time $E(T_1) + E(T_2) + ... + E(T_{j-1})$.

Thus, the mean times of infection of vulnerable hosts in the enterprise network, and hence the mean times at which the corresponding infected host numbers are created, are a function of $E(T_j)$, which we can estimate via the stochastic model represented by Equations 2.1 and 2.2 [lines 1-6]. The mean times of creation of infected host numbers estimated in relation to the example botnet versus network setting are shown in the example model graph as attributes of the edges that enter those infected host numbers. As we wrote earlier in this report, there is a variability around the mean times at which the botnet infection process enters its states I_j, for $j \neq 1$. In this research we define a variability time window for each infected host number j, for $j \neq 1$, to represent as much as possible those variabilities. A variability time window is confined by a left endpoint and a right endpoint.

The left endpoint of the variability time window of an infected host number j is halfway between the mean time of creation of infected host number $j - 1$ and the mean time of creation of infected host number j. The left endpoint itself is not part of the variability time window. In the case $j = 2$, the left endpoint is 0.0 as we consider the time of creation of infected host number 1 to be exact rather than a mean. The right endpoint of the variability time window in question is halfway between the mean time of creation of infected host number j and the mean time of creation of infected host number $j + 1$. In the case $j = \varphi$, the right endpoint is set to be as far from the mean time of creation of infected host number j from the right as the left endpoint is from the left. That specific right endpoint is considered during estimation of the infectious time [line 7].

The right endpoint itself is part of the variability time window. The previous formulation ensures that variability time windows are allocated evenly among infected host numbers. In the model graph, the endpoints of the variability time window of an infected host number j are kept as attributes of the vertex labeled j as in

Υ	Vector in which indices represent infected host numbers and elements represent the corresponding infection time	ϑ	Relative time elapsed until the botnet infection process reaches the infected host number that precedes the one currently under consideration
Ψ	Vector in which indices represent infected host numbers and elements represent the corresponding number of infection attempts	ζ	Refers to a specific infection attempt initiated by a specific infected host number
Λ	Matrix in which row indices represent infected host numbers and column indices represent infection attempts. Each element represents the relative time in which the infected host number represented by the row index initiates the infection attempt represented by the column index	Φ	Matrix whose row indices and column indices are defined as in Λ. Each element represents the probability of success of the infection attempt represented by the column index and initiated by the infected host number represented by the row index
Θ	Matrix in which row indices represent infected host numbers and column indices represent possible numbers of offsprings starting from 0. Each element represents the probability that the infected host number represented by the row index generates the number of offsprings represented by the column index	Γ	Matrix in which row indices and column indices represent infected host numbers. Each element represents the probability that the infected host number represented by the column index is an offspring of the infected host number represented by the row index
Ξ	Matrix whose rows represent restricted weak integer compositions	Δ	Vector in which indices represent infected host numbers and elements represent the corresponding number of offsprings
μ	Number of summands in restricted weak integer compositions	α_j	Lower bound on the values of the summand that corresponds to infected host number j in restricted weak integer compositions
σ_j	Upper bound on the values of the summand that corresponds to infected host number j in restricted weak integer compositions	ρ	Infected host number whose infection time precedes the time in which a specific infection attempt was initiated
ς	Infected host number targeted by a specific infection attempt	ω	Botnet infectious time

Table 4.1: Definitions of additional variables used in the model graph generation algorithm.

the example model graph. An estimate of the infectious time enables us to estimate for each infected host number the number of infection attempts initiated by the corresponding infected host [lines 8-10]. Note that β can be approximated from the weakly connected subgraph in consideration. We estimate for each infected host number the mean time of occurrence of each infection attempt that corresponds to that infected host number. The mean time of occurrence of an infection attempt tells us what specific infected host number that infection attempt has potential to create.

We discover the variability time window within which the mean time of occurrence in question falls, and thus conclude that it is creation of the infected host number the variability time window in question belongs to that is targeted by the infection attempt. The probability that the infection attempt causes an infection is equal to the vulnerability rate of the enterprise network at the time in which that infection attempt is initiated. If the infection attempt has potential to create infected host number j, then the probability of success of the infection attempt is δ_{j-1}. We iterate this reasoning for all infection attempts [lines 11-28]. For each infected host number, let us denote with ξ the overall number of infection attempts initiated by the corresponding infected host, with η_i the probability of success of infection attempt i, with τ a random variable that holds the actual number of successes, and ϖ the possible numbers of successes.

For the sake of clarity, with success of an infection attempt we mean the event of that infection attempt causing an infection. Note that at this point we have already estimated ξ and each η_i. We can notice that the infection attempts are independent Bernoulli trials with possibly varying probabilities of success, consequently we can conclude that τ has a Poisson Binomial distribution that can be estimated as follows:

$$G(i) = \sum_{s=1}^{\xi} \left(\frac{\eta_s}{1-\eta_s} \right)^i \tag{4.3}$$

$$P(\tau = \varpi) = \begin{cases} \prod_{i=1}^{\xi}(1-\eta_i) & \varpi = 0 \\ \frac{1}{\varpi}\sum_{i=1}^{\varpi}(-1)^{i-1}P(\tau = \varpi - i)G(i) & \varpi > 0 \end{cases}$$

A successful infection attempt implies creation of an offspring of the infected host number, therefore Equation 4.3 provides the probability distribution of the number of offsprings of the infected host number in question. We estimate the probability distribution of the number of offsprings of each infected host number besides φ [lines 29-33]. Infected host number φ is the last to be created, and thus it cannot have any offsprings as at that point the vulnerability rate in the enterprise network hits 0.0. In the example botnet versus network setting, this is how part of the probability distribution of the number of offsprings of infected host number 1 looks like: $P(\tau = 0) = 0.054867; P(\tau = 1) = 0.196391; P(\tau = 2) = 0.300714; P(\tau = 3) = 0.257415$, etc. The sum of the numbers of offsprings of all infected host numbers amounts to $\varphi - 1$ as all infected host numbers but 1 are offsprings of some other infected host number.

We draw on number theory to identify the most likely assignment of numbers of offsprings to infected host numbers. We can notice that each possible assignment of numbers of offsprings to infected host numbers is a weak composition of $\varphi - 1$. Recall from number theory that the weak composition of an integer is an ordered sequence of positive integers or 0 whose sum is equal to that integer. A weak composition that relates to the example botnet versus network setting is the following: $(2, 1, 1, 0, 0)$. In our context that weak composition indicates that infected host number 1 has 2 offsprings, infected host number 2 has 1 offspring, and so forth. We generate weak compositions of $\varphi - 1$ via the algorithm for generating restricted integer compositions discussed by Opdyke in (Opdyke, 2009). The compositions generated by that algorithm can be constrained simultaneously by upper and lower bounds on the number of summands and upper and lower bounds on the values of those summands.

We know that infected host number φ will have 0 offsprings, therefore we focus on the remaining infected host numbers by setting the number of summands to precisely $\varphi - 1$. Practical experiments with this research showed that the probability distribution of the number of offsprings of each infected host number constantly indicated a most likely number of offsprings that was smaller than the actual number of offsprings. For each infected host number j, we use that specific most likely number of offsprings as a lower bound on the value of the summand that corresponds to that infected host number. In other words, summing up the numbers of offsprings with the highest probabilities in the distributions of τ discussed previously produces a number that is less than $\varphi - 1$.

For each infected host number j, we sum up the numbers of offsprings with the highest probabilities in the distributions of τ estimated for infected host numbers other than j, and thus subtract the resulting sum from $\varphi - 1$. We use the final result as an upper bound on the values of the summand that corresponds to infected host number j. Once we generate the weak compositions constrained by the parameters just discussed, we estimate the likelihood of each one of those weak compositions by multiplying the probabilities associated with the values of the respective summands. The weak composition with the highest probability forms the assignment of numbers of offsprings to infected host numbers that we apply in the model graph [lines 34-43]. The number of offsprings of each infected host number is kept as an attribute of the corresponding vertex as in the example model graph.

We now estimate the probability that an infected host number q is offspring of an infected host number j. Each one of the infection attempts initiated by the infected host corresponding to infected host number j, which has potential to create infected host number q, has a probability of success $\varrho = \delta_{q-1}$. That postulate stems from the fact that the infection rate in the enterprise network by the time the botnet infection process enters state I_q is δ_{q-1}. Let us denote with χ the number of infection attempts initiated by the infected host corresponding to infected host number j and whose probabilities of success amount to δ_{q-1}. Also, let κ denote a random variable that holds the number of successes of those χ infection attempts. We can notice that $\kappa \sim \text{Binom}(\chi, \varrho)$.

The probability that infected host number q is offspring of infected host number j is equal to the probability that the infection attempts in question cause an infection, which in turn is equal to the probability that $\kappa = 1$. That probability is obtained from the formula for the binomial distribution as in the following equation:

$$P(\kappa = 1) = \chi\,\varrho(1 - \varrho)^{\chi - 1} \tag{4.4}$$

We iterate the above estimation for each infected host number q and j, for $q \neq j$ and $j < \varphi$, and thus conclude that the parent of infected host number q is that specific infected host number j whose probability returned by Equation 4.4 is the highest [lines 44-61].

```
 1: $\vartheta \leftarrow 0.0$
 2: $\Upsilon(1) \leftarrow 0.0$
 3: **for** $j = 2$ to $\varphi$ **do**
 4:     $\Upsilon(j) \leftarrow \vartheta + \frac{1}{\beta\delta_{j-1\,j-1}}$
 5:     $\vartheta \leftarrow \Upsilon(j)$
 6: **end for**
 7: $\omega \leftarrow \Upsilon(\varphi) + \frac{\Upsilon(\varphi)-\Upsilon(\varphi-1)}{2}$
 8: **for** $j = 1$ to $\varphi$ - 1 **do**
 9:     $\Psi(j) \leftarrow (\omega - \Upsilon(j)) * \beta$
10: **end for**
11: **for** $j = 1$ to $\varphi$ **do**
12:     **for** $\zeta = 1$ to $\Psi(j)$ **do**
13:         $\Lambda(j,\zeta) \leftarrow \Upsilon(j) + ((1/\beta) * \zeta)$
14:         $\rho \leftarrow$ infected host number such that $\Upsilon(\rho)$ precedes $\Lambda(j,\zeta)$
15:         **if** $\rho = 1$ **then**
16:             $\varsigma \leftarrow 2$
17:         **else if** $\rho = \varphi$ **then**
18:             $\varsigma \leftarrow \varphi$
19:         **else**
20:             **if** $\Lambda(j,\zeta) > \Upsilon(\rho) + \frac{\Upsilon(\rho+1)-\Upsilon(\rho)}{2}$ **then**
21:                 $\varsigma \leftarrow \rho + 1$
22:             **else**
23:                 $\varsigma \leftarrow \rho$
24:             **end if**
25:         **end if**
26:         $\Phi(j,\zeta) \leftarrow \delta_{\varsigma-1}$
27:     **end for**
28: **end for**
29: **for** $j = 1$ to $\varphi$ **do**
30:     **for** $\zeta = 1$ to $\Psi(j) + 1$ **do**
```

```
31:       $\Theta(j,\zeta) \leftarrow$ feed Equation 4.1 for $\tau = (\zeta - 1)$ with $\Phi(j,q), \forall q \in \{1,2,...,\Psi(j)\}$
32:     **end for**
33: **end for**
34: $\mu \leftarrow \varphi - 1$
35: **for** $j = 1$ to $\varphi - 1$ **do**
36:     $tmp\_var \leftarrow$ column index of Max$(\Theta(j,q)), \forall q \in \{1,2,...,\varphi\}$
37:     $\alpha_j \leftarrow tmp\_var - 1$
38: **end for**
39: **for** $j = 1$ to $\varphi - 1$ **do**
40:     $\sigma_j \leftarrow \varphi - 1 - \sum_{q=1,q\neq j}^{\varphi-1} \alpha_q$
41: **end for**
42: $\Xi \leftarrow$ feed restricted integer composition generator with $\mu$, $\alpha_j$, and $\sigma_j$, $\forall j \in$
    $\{1,2,...,\varphi - 1\}$
43: $\Delta \leftarrow$ specific row $\upsilon$ of $\Xi$ with Max$(\prod_{q=1}^{\varphi-1} \Theta(q, \Xi(\upsilon,q) + 1))$
44: **for** $j = 1$ to $\varphi - 1$ **do**
45:     **for** $q = j + 1$ to $\varphi$ **do**
46:       $\varrho \leftarrow \delta_{q-1}$
47:       $\chi \leftarrow 0$
48:       **for** $\zeta = 1$ to $\Psi(j)$ **do**
49:         **if** $\Phi(j,\zeta) = \varrho$ **then**
50:           $\chi = \chi + 1$
51:         **end if**
52:       **end for**
53:       **if** $\chi = 0$ **then**
54:         $\Gamma(j,q) \leftarrow 0.0$
55:       **else if** $\chi = 1$ **then**
56:         $\Gamma(j,q) \leftarrow \varrho$
57:       **else**
58:         $\Gamma(j,q) \leftarrow \chi\,\varrho(1-\varrho)^{\chi-1}$
59:       **end if**
60:     **end for**
61: **end for**
```

A relevant question that rises is why we need to estimate the number of offsprings of each infected host number while we can determine the parent-offspring relation between all possible infected host numbers. We could have deterministically derived the number of offsprings of each infected host number j by simply counting the infected host numbers that have j as a parent. Nevertheless, doing so does not produce accurate results. Our practical experiments showed that it is quite common that several infected host numbers appear as parents of the same infected host number with equal probabilities. At the modeling level it is virtually unfeasible for us to determine which one of those equally likely but mutually exclusive parents is indeed the actual parent. Because of that reason, the model graph contains mutually exclusive concurrent edges. The concurrent edges observed in our example model graph are shown in Figure 4.2, while one of the several possible instances of the example model graph with those edges removed is shown on the bottom part of Figure 4.1.

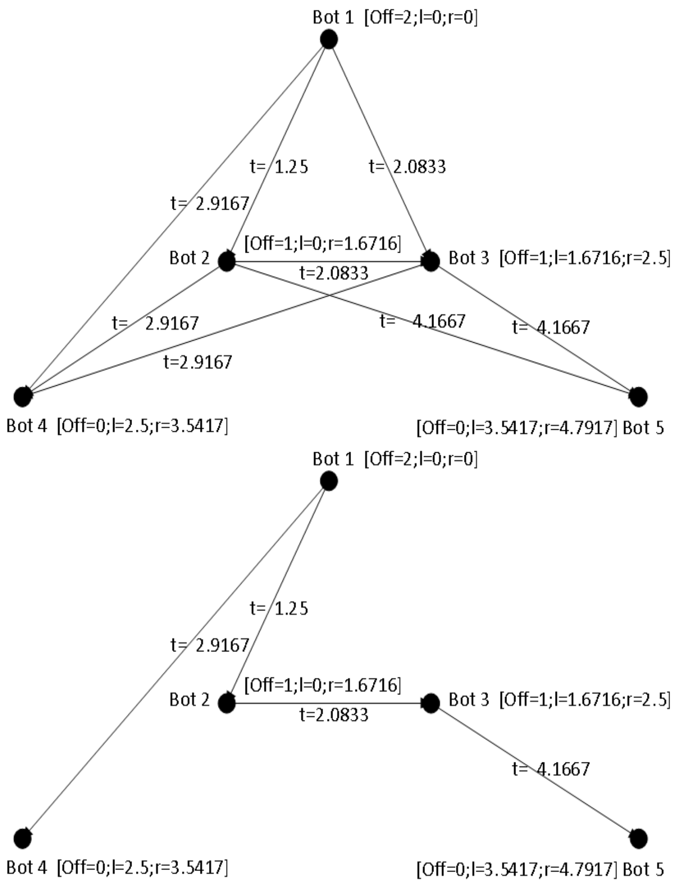


Figure 4.1: Example of a model graph as generated by the algorithm (top), and an instance of that model graph with mutually exclusive concurrent edges removed (bottom).

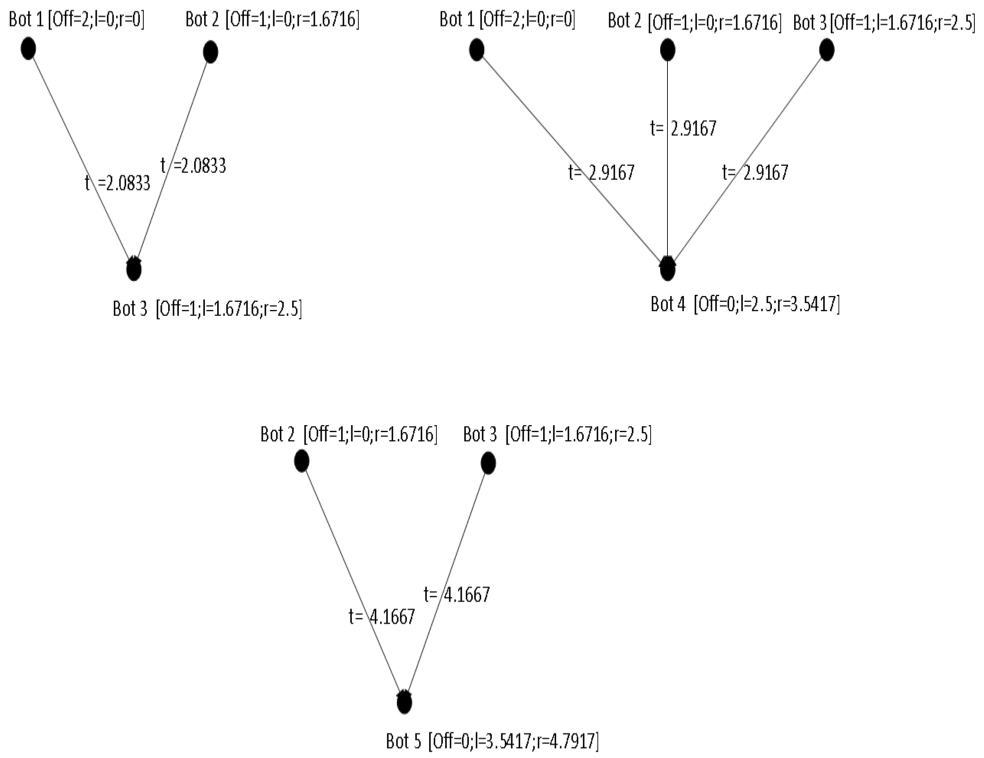

Bot 1 [Off=2;l=0;r=0]
Bot 2 [Off=1;l=0;r=1.6716]
t =2.0833
t =2.0833
Bot 3 [Off=1;l=1.6716;r=2.5]
Bot 1 [Off=2;l=0;r=0]
Bot 2 [Off=1;l=0;r=1.6716]
Bot 3 [Off=1;l=1.6716;r=2.5]
t= 2.9167
t= 2.9167
t= 2.9167
Bot 4 [Off=0;l=2.5;r=3.5417]
Bot 2 [Off=1;l=0;r=1.6716]
Bot 3 [Off=1;l=1.6716;r=2.5]
t= 4.1667
t= 4.1667
Bot 5 [Off=0;l=3.5417;r=4.7917]

Figure 4.2: Mutually exclusive concurrent edges in the model graph generated by the algorithm.

Chapter 5

Subgraph Isomorphism Search

The purpose behind the model graph is to use it as basis for proving or disproving that a given weakly connected subgraph within the data graph is caused by a botnet. We do so by searching for an isomorphism between any subgraph of the model graph with infected host number 1 as root and the weakly connected subgraph under analysis. If an isomorphism between the two is found, we conclude that the inferred dynamics of the propagation of a suspected botnet match the actual propagation dynamics observed on the network. As our experimentation with this research has shown, an exact match between the two is seldom observed. We tolerate deviations from the inferred dynamics by defining an error model based on the variability time window of each infected host number along with indication of mutually exclusive concurrent edges in the model graph. We assign two vertex attributes to each vertex in the weakly connected subgraph.

One vertex attribute, which in this research we refer to as possible identity (ID), holds a possible infected host number that might correspond to the vertex. At the beginning of our processing of the weakly connected subgraph we identify the vertex whose incoming edge has the earliest timestamp, and thus consider it as root. If at the end our search has not found an isomorphism, we try with the succeeding vertices as root one at a time according to the age of the timestamps of their incoming edges. Once we set a specific vertex in the weakly connected subgraph to be the root, the time on the edges the enter the vertices of the weakly connected subgraph currently in consideration are considered relative to the time on the edge that enters the root. The ID attribute of the root vertex is set to 1. For each vertex, we examine each relative time on each edge that enters the vertex to search for the variability time window in the model graph it lies within.

The infected host number that has that variability time window becomes an ID of the vertex. The other vertex attribute, which in this research we refer to as offsprings number (ON), holds the number of offsprings of the corresponding vertex in the weakly connected subgraph whose IDs are among the infected host numbers that are indicated in the model graph as possible offsprings of the infected host number held by the ID attribute of the vertex in question. We identify the most recent relative time that enters a leaf vertex in the weakly connected subgraph, and thus consider that specific part of the model graph that extends up to that relative time. For each infected host number in that part of the model graph we consider the number of offsprings along with the specific offsprings generated by that infected host number up to the relative time in question.

At that point we are ready to start the search process. We first see whether we can validate the ID attribute of the root vertex in the weakly connected subgraph. We do so by checking whether the value of the ON attribute of the root vertex is greater than or equal to the predicted number of offsprings of infected host number 1. We allow values of the ON attribute that are less than the predicted number of offsprings of infected host number 1 by a threshold. Our practical experiments showed that an adequate value of that threshold is 1, given the small number of vertices we are dealing with. We then proceed in a similar fashion with the vertex whose ID attribute has a value of 2. We check whether the value of the ON attribute of that vertex is greater than or equal to the predicted number of offsprings of infected host number 2 minus 1. Once we localize infected host number 2 in the weakly connected subgraph, we move on to the vertex whose ID attribute has a value of 3, and so forth.

We develop an incremental auto-corrective feature within our isomorphism search process. Each time we localize an infected host number in the weakly connected subgraph, we take the relative time on the edge that enters that corresponding vertex and set it directly as the infection time of that infected host number within the model graph generation process. We then replay the model graph generation process, and thus update the model graph along with the values of the attributes of the remaining vertices of the weakly connected subgraph. For example, if we have localized infected host number 2 in the weakly connected subgraph, we take the relative time on the edge that enters the vertex whose ID is 2, and thus use that time as the infection time of infected host number 2 within the model graph generation process.

Instead of using the stochastic model to estimate the infection time of infected host number 2, we take that infection time directly from the weakly connected subgraph as just discussed. Replaying the graph generation process with such an intervention incorporated produces a more accurate variability time window for infected host number 3, and thus increases our chances of correctly localizing infected host number 3 in the weakly connected subgraph. Once we localize infected host number 3, we follow the same technique to increase our chances of correctly localizing infected host number 4, and so forth. After processing the last leaf node in the weakly connected subgraph, for each vertex in the weakly connected subgraph we verify that the value of its ON attribute does not vary more than 1 in either direction with respect to the number of offsprings of the infected host number indicated by the ID attribute of that vertex. That check is performed by consulting the numbers of offsprings of the vertices in the pertinent part of the model graph.

Chapter 6

Experimental Evaluation

6.1 Vulnerability Rate Estimation

We implemented our approach to vulnerability rate estimation in the Matlab programming language (Mathworks Inc., 2010), and thus tested the effectiveness of that approach experimentally in relation to simulated botnet propagation dynamics in a Georgia Tech Network Simulator (GTNetS) (Riley *et al.*, 2004). The GTNetS tool enabled us to simulate moderate to large scale networks. We mostly worked with simulated networks whose size had an order of magnitude of $3x10^5$ possible nodes and 10^5 actual nodes. The topology of each one of those simulated networks was a random tree as provided by the GTNetS tool. The distribution of nodes throughout constituent subnetworks was also conducted by the GTNetS tool in a random fashion. The GTNetS tool provides for simulation of worms over simulated networks. We used that functionality to simulate botnet propagation dynamics in various simulated networks.

We modified the GTNetS code such as to simulate techniques for vulnerability scanning and botnet membership assessment. That additional code allowed us to log the properties of each node in a sample of nodes from each simulated network of reference, i.e. determine whether or not the node is vulnerable and whether or not the node is infected. We also developed code within GTNetS to randomly generate a sample of nodes from each simulated network of reference, and thereafter randomly distribute those nodes into pools. In that code, we could programmatically set both the sample size and the number of pools. The GTNetS tool itself allowed for setting the vulnerability rate in each simulated network of reference. In this report we refer to that rate as true vulnerability rate.

After botnet propagation dynamics were simulated long enough to affect the whole simulated network, the true vulnerability rate became true infection rate as at that point in time each vulnerable node became an infected node. Thus, estimating the vulnerability rate at the beginning of each simulation was equivalent to estimating the infection rate at the end of that simulation as both of those true rates had the same value. We tried the Matlab implementation of the estimation approach in practice in relation to a large number of experiments with various simulation parameters, namely true vulnerability and infection rates, actual network sizes, sample sizes, and numbers of pools. In empirical terms, the estimation approach proved to support a ratio of the sample size to the actual network size of up to 1:1000 when employed in networks of medium or large sizes.

The outcome of each experiment was an estimate of the vulnerability or infection rate, which we compared to the true rate that was in place during that experiment in order to derive the corresponding estimation error. Figure 6.1 shows our findings from those experiments. The ratios of the sample size to the actual network size employed varied from 1:100 to 1:1000, while the numbers of pools employed varied from a few, namely 11, to as many as 1200. Clearly if defenders could inspect each or almost each actual node for vulnerability in an amount of time that meets botnet mitigation requirements, the random sampling would not apply and moreover the calculation of vulnerability rates would have been straightforward.

Because of that postulate, we deemed that given the order of magnitude of the employed actual network size, the lower bound 1:100 was approximately a minimum ratio of the sample size to the actual network size for the random sampling to find applicability in our problem domain. The motive for the upper bound 1:1000 is its representation of a ceiling ratio of the sample size to the actual network size, which the estimation approach can support reliably. The sample sizes employed in the experiments are a direct derivation from the actual network sizes and the aforementioned ratios. We followed ratios of the numbers of pools to the sample sizes from 1:1 to 1:10, which explains the numbers of pools employed, namely from 11 to 1200.

The sign of the observed estimation errors represents the direction of departure of the estimated vulnerability rates from the respective true rates. An estimation error is positive when the estimated vulnerability rate is lower than the true rate, and negative when the true rate is lower than the estimated vulnerability rate. Note that the experimental results conveyed in Figure 6.1 do not reflect the use of the logit model. The pertinent prediction outcomes of the logit model, namely numbers of pools found to be highly likely optimal for given sample sizes and actual network sizes discussed earlier in this section, are depicted in Figure 6.2.

Figures 6.1 and 6.2 are such that the experimental results that those two figures show can be correlated. For each ratio of the sample size to the actual network size employed, we can observe the associated optimal number of pools from Figure 6.2. With the optimal number of pools at hand, we can observe the associated estimation error from Figure 6.1 by referring to the graph that corresponds to the ratio of the sample size to the actual network size under consideration. The overall highest estimation errors observed during these experiments regard a ratio of the sample size to the actual network size of 1:1000. This comes natural as for such ratio the number of nodes in the sample is considerably small with respect to the whole population of nodes in a network of reference.

The estimation errors diminish as we move towards lower ratios of the sample size to the actual network size. Figure 6.3 shows a direct comparison between estimated rates and true rates for a ratio of the sample size to the actual network size of 1:1000. For lower ratios of the sample size to the actual network size, the estimated rates line follows more closely the true rates line on the left part of Figure 6.3, while the corresponding estimation error line on the right part of Figure 6.3 fluctuates closer to zero as it moves along the true rates that we tried in these experiments.

6.2 Botnet Mitigation

We implemented the overall approach to early stage botnet detection and containment in the Matlab and Perl programming languages. More specifically, we implemented in Matlab the part of this research that regards statistical inference of botnet dynamics. A large portion of that code uses functions from the Matlab's Statistics Toolbox. We implemented in Perl the subgraph isomorphism search part of this research. That code relies on the Graph::Directed module, i.e. a Comprehensive Perl Archive Network (CPAN) module that provides functions for creating and processing directed graphs. The implementation takes the form of a unified tool referred to as *variant* which allows an analyst to interface with the Matlab code. The Perl code is transparent as it is invoked by Matlab. Matlab has an instruction, *perl*(), that allows for calling Perl code directly from within Matlab code. The input of the *variant* tool is comprised of pcap files generated by tcpdump or any other derivative network sniffing tool.

The output produced by the *variant* tool comprises a list of IP addresses found to be infected by bots along with thorough details of inferred dynamics and of the subgraph isomorphism search process. In fact a considerable part of the experimentation data that we provide in this section were produced from the *variant* tool. We wrote bot code in order to test the effectiveness of this research in practice in a controlled fashion. The bot code exploits a network service that we wrote in Perl especially for being able to experiment with that bot code. The vulnerability lies in incomplete validation of input data to an *open*() function. If input data, which in our case is a file name, contains a pipe, that function by definition attempts to execute whatever comes after the pipe. The exploit works by injecting $'|\, perl\ -e\ < bot\_payload >'$ into the *open*() function in question. We also wrote a program to collect propagation data from bots.

Upon infection of a host, each bot reports to the collector program with data that comprises infection time, namely a timestamp that the bot gets on the infected host by the time of infection, the IP address of the infected host, and the IP address of the host that initiated the infection. Such monitoring data enabled us to directly compare the results reported by the *variant* tool to the actual facts in the propagation of the test botnet. We conducted the practical experiments with this research in the Emulab network testbed (White *et al.*, 2002). Our test network consisted of 100 hosts with IP addresses in the range 10.1.1.2 – 10.1.1.101. A portion of these hosts were made vulnerable to the test botnet by simply running on them the vulnerable network service discussed previously. We sniffed network traffic in our test network via the tcpdump tool. We do not need all TCP packets that flow over the network of reference, but only those that start the three-way handshake. The actual tcpdump filter that we used in our experiments in conjunction with the *variant* tool is the following: $'((tcp[tcpflags] \ \& \ tcp - syn \ != \ 0) \ or \ udp) \ and \ ip'$

We conducted several rounds of trials in relation to various botnet versus network settings. Due to space limitations, in our following discussion we refer to experiments in which the bots of the test botnet initiated an average number of infection attempts of 3 per minute. In those experiments our test network had $\delta_0 = 0.3$. The IP addresses of the vulnerable hosts varied from 10.1.1.2 to 10.1.1.31. The inferred propagation dynamics as incorporated in the model graph are shown in Table 6.1. The mean infectious time estimated was 9.3784 minutes. We do not show the complete parent-offspring relations between infected host numbers through the corresponding model graph as its size along with the density of those relations makes that graph hardly readable when depicted on one page. Nevertheless, the possible offsprings of the first eight infected host numbers are given in Table 6.2.

Recall from the previous chapters that possible offsprings of an infected host number as expressed in the model graph are only candidates for being offsprings of that infected host number. Due to concurrent edges, only a part of those possible offsprings will be actual offsprings of the infected host number in question at each run. In the experiments in question our approach began processing weakly connected subgraphs formed within data graphs at $v = 8$. As propagation of a botnet is a stochastic process, the times at which vulnerable hosts got infected by bots was different at each trial. Figure 6.4 shows host infection times until the botnet propagation process enters state I_8 for five of those trials from the rounds of trials conducted in this research, which in this section we refer to as $G1$, $G2$, $G3$, $G4$, and $G5$. Figure 6.4 also shows predicted host infection times in the model graph for comparative purposes.

Also, due to the stochastic nature of the botnet propagation process the actual number of offsprings of each infected host number may vary from trial to trial. Table 6.3 shows on one hand the predicted number of offsprings of each one of the first eight infected host numbers in the model graph until the botnet propagation process enters state I_8. And on other hand, Table 6.3 shows the actual number of offsprings of each one of those infected host numbers in each one of the $G1$-$G5$ trials. We do not show the same kind of data about the other trials due to space limitations. A direct consequence of the variability around host infection times in each trial is that the variability time windows used by our approach to recognize infected host numbers varies from trial to trial. That is due to the incremental auto-corrective feature of our approach.

Bot Num.	Inf. Time	Num. Offsprings	Act. Time	Attempts
1	0.0	4	9.378388	28
2	1.1494	3	8.228962	24
3	1.7447	2	7.633724	22
4	2.1562	2	7.222202	21
5	2.4767	2	6.901689	20
6	2.7434	2	6.635022	19
7	2.9748	1	6.403541	19
8	3.1819	1	6.196501	18
9	3.3713	1	6.007107	18
10	3.5476	1	5.830741	17
11	3.7143	1	5.664074	16
12	3.8738	1	5.504584	16
13	4.0281	1	5.350263	16
14	4.179	1	5.199434	15
15	4.3278	1	5.050624	15
16	4.4759	1	4.902476	14
17	4.6247	1	4.753667	14
18	4.7756	1	4.602837	13
19	4.9299	1	4.448516	13
20	5.0894	1	4.289026	12
21	5.256	0	4.122360	12
22	5.4324	0	3.945993	11
23	5.6218	0	3.756599	11
24	5.8288	0	3.549560	10
25	6.0603	0	3.318078	9
26	6.327	0	3.051411	9
27	6.6475	0	2.730899	8
28	7.059	0	2.319376	6
29	7.6542	0	1.724138	5
30	8.8037	0	0.0	0

Table 6.1: Pertinent data from the inferred dynamics of the test botnet in the Emulab network testbed.

1 ⇒	2	4	5	6	7	9	11	13	15	17	19	21	23	25	26	28	30
2 ⇒	3	4	5	6	8	10	12	14	16	18	20	22	24	25	26	29	
3 ⇒	4	5	6	7	9	11	13	16	18	20	22	24	25	26	29		
4 ⇒	5	6	8	10	12	14	16	18	20	22	24	25	27	29			
5 ⇒	6	8	10	12	14	16	18	20	22	24	25	26	29				
6 ⇒	7	9	11	13	16	18	20	22	24	25	26	29					
7 ⇒	9	11	13	15	17	19	21	23	25	26	28	30					
8 ⇒	10	12	14	16	18	21	22	24	25	27	29						

Table 6.2: Possible offsprings of the first eight infected host numbers in the model graph.

Bot Number	M	G1	G2	G3	G4	G5
1	2	3	2	2	2	4
2	2	2	1	2	2	2
3	1	1	1	2	1	1
4	1	1	1	1	0	0
5	1	0	1	0	1	0
6	0	0	1	0	1	0
7	0	0	0	0	0	0
8	0	0	0	0	0	0

Table 6.3: Comparison in numbers of offsprings of each infected host number between the model graph and some of the weakly connected subgraphs targeted by our approach.

Bot Number	M	G1	G2
1	0.0 - 0.0	0.0 - 0.0	0.0 - 0.0
2	0.0 - 1.4470	0.0000 - 1.4470	0.0000 - 1.4470
3	1.4470 - 1.9504	1.6376 - 2.1410	0.9643 - 1.4677
4	1.9504 - 2.3164	2.2118 - 2.5778	1.2058 - 1.5718
5	2.3164 - 2.6100	2.5003 - 2.7938	1.4903 - 1.7838
6	2.6100 - 2.8591	2.8063 - 3.0554	1.7933 - 2.0424
7	2.8591 - 3.0784	3.1157 - 3.3350	2.1124 - 2.3317
8	3.0784 - 3.2766	3.4435 - 3.6417	2.4335 - 2.6317

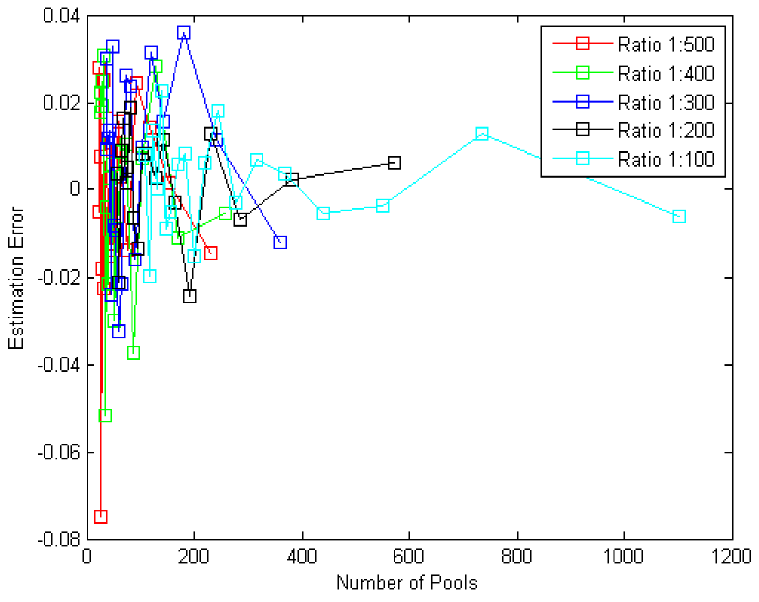
Bot Number	G3	G4	G5
1	0.0 - 0.0	0.0 - 0.0	0.0 - 0.0
2	0.0000 - 1.4470	0.0000 - 1.4470	No Match
3	0.6276 - 1.1310	1.2976 - 1.8010	1.4470 - 1.9504
4	1.2351 - 1.5287	1.8724 - 2.2384	2.0219 - 2.3879
5	1.4936 - 1.7872	2.1603 - 2.4538	2.4936 - 2.7872
6	1.7967 - 2.0457	2.4667 - 2.7157	No Match
7	2.1157 - 2.3350	2.7824 - 3.0017	No Match
8	2.4335 - 2.6317	3.1035 - 3.3017	No Match

Table 6.4: Evolution of variability time windows throughout the processing of some of the weakly connected subgraphs by our approach.

Table 6.4 shows how those variability time windows evolve throughout the processing of weakly connected subgraphs in the $G1$-$G5$ trials with respect to the variability time windows predicted in the model graph. As the data discussed so far seem to suggest, the internals of the processing by our approach of weakly connected subgraphs, which are formed by the same botnet in a network whose characteristics do not change from trial to trial, vary from trial to trial according to the departure of the actual propagation dynamics from the mean propagation dynamics. If that departure exceeds the boundaries of the error model that accompanies the model graph, then the botnet will go undetected. Trials $G1$-$G4$ are a few examples in which our approach detected the corresponding weakly connected subgraphs as botnets.

The weakly connected subgraphs that were derived by our approach in those trials are given in Figure 6.5. These are a few examples of weakly connected subgraphs that our approach found to be isomorphic to that part of the model graph that extends up to $v = 8$. Note that our approach does not need to know v a priori. The actual value of v emerges when our approach has completed processing of weakly connected subgraphs within the data graph. $G5$ is an example of a trial in which the actual propagation dynamics of the test botnet exceeded the boundaries of the error model that accompanies the model graph. In cases such as $G5$ our approach fails to detect the botnet. The weakly connected subgraph as actually exhibited in $G5$ and as derived from our approach is given in Figure 6.6.

Although possible, departures beyond the boundaries of the error model that accompanies the model graph are relatively rare to a random degree. Empirical observations of the effectiveness of our approach in ten rounds of trials are shown in Figure 6.7. Replaying those rounds of trials will produce different results. Nevertheless, the overall hit rates oscillate around similar high values. To our experience this research does not produce false positives when applied to an enterprise network with standard Windows and/or Linux network services. We suspect that P2P applications might represent a challenge in that regard due to their worm-like behavior. Nevertheless, we do not expect P2P applications to represent a limitation of this research as we deem that the worm-like behavior of most P2P applications is exhibited in terms of topology and not as persistent propagation dynamics. However, this research did not include a practical validation of the robustness of our approach from a false positives perspective with respect to a representative set of real-world P2P applications. Consequently we leave that issue open, and thus consider it as possible future work.



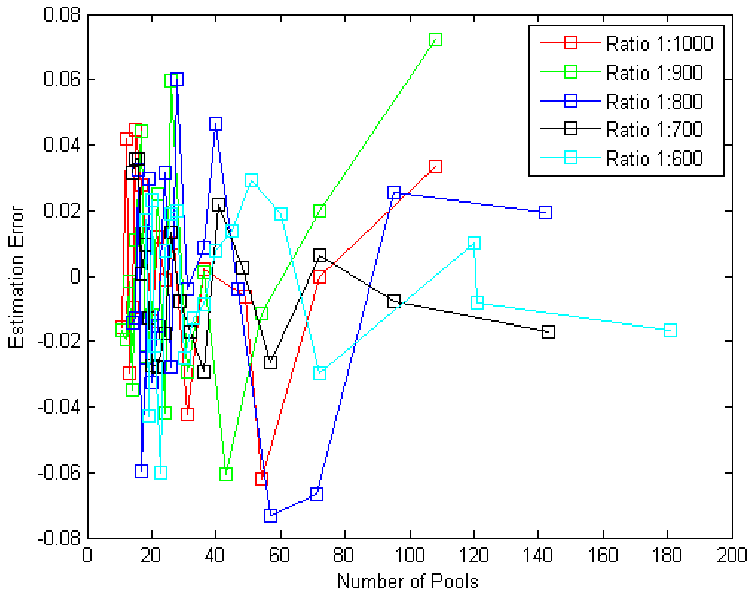


Figure 6.1: Plot of observed estimation errors versus numbers of pools for ratios of the sample size to the actual network size from 1:100 to 1:1000.

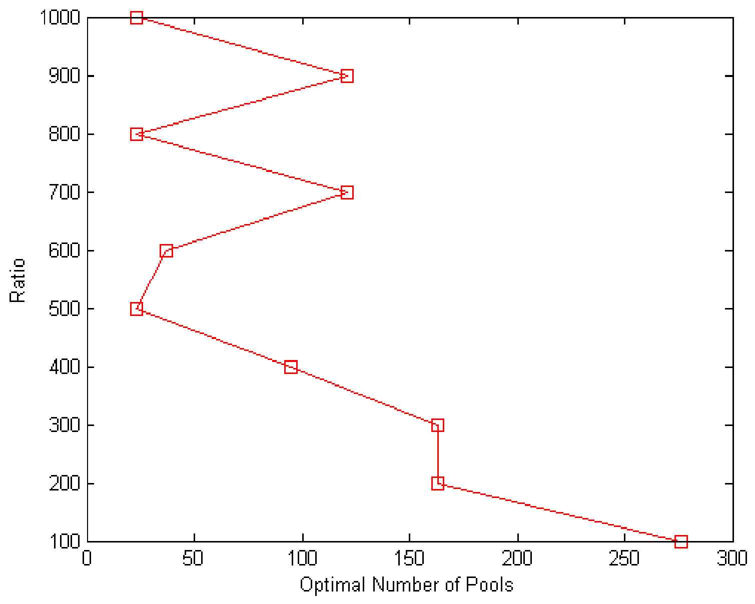
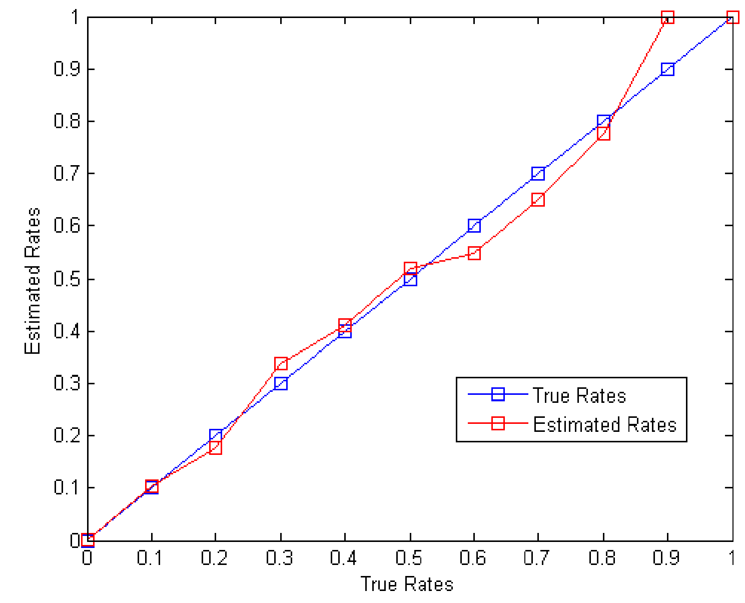


Figure 6.2: Optimal numbers of pools found by the logit model for various ratios of the sample size to the actual network size.



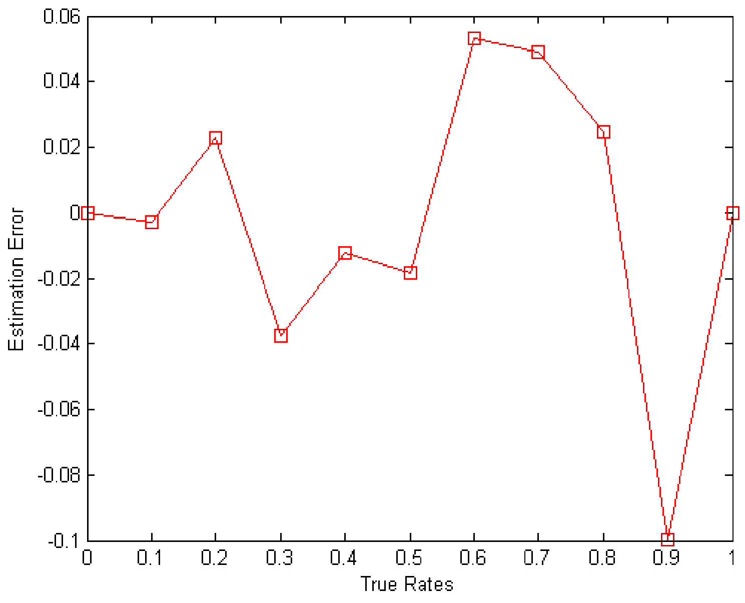


Figure 6.3: Comparison between estimated rates and true rates (top), and estimation errors observed for those true rates (bottom), for a ratio of the sample size to the actual network size of 1:1000.

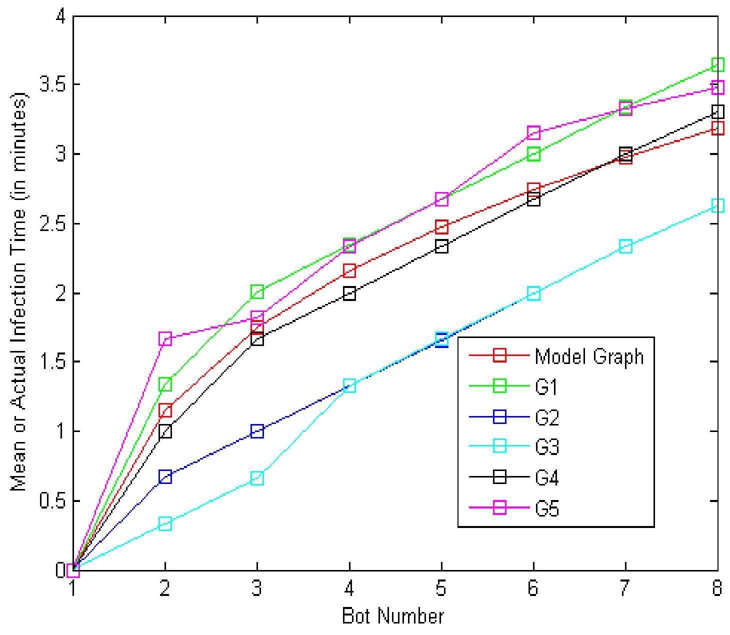


Figure 6.4: Host infection times in the model graph and in some of the weakly connected subgraphs targeted by our approach.

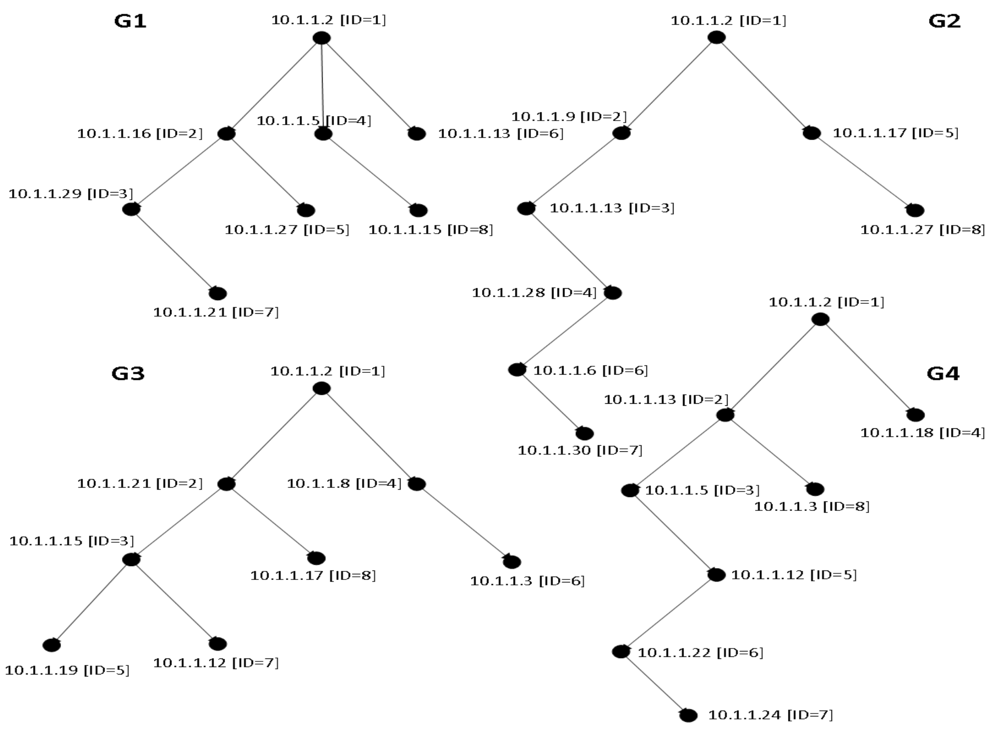


Figure 6.5: Some of the weakly connected subgraphs as derived and detected by our approach.

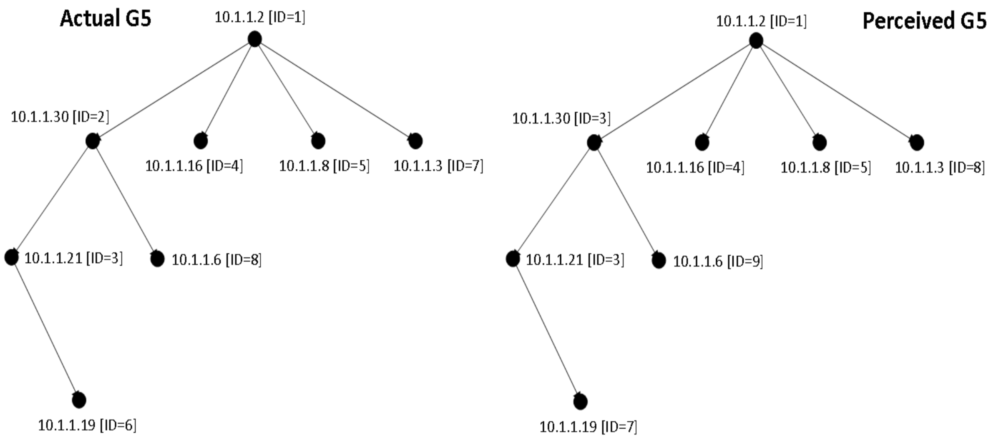


Figure 6.6: Weakly connected subgraph G5 in its actual form as generated by the test botnet (left), and in a derived form as built by our approach (right).

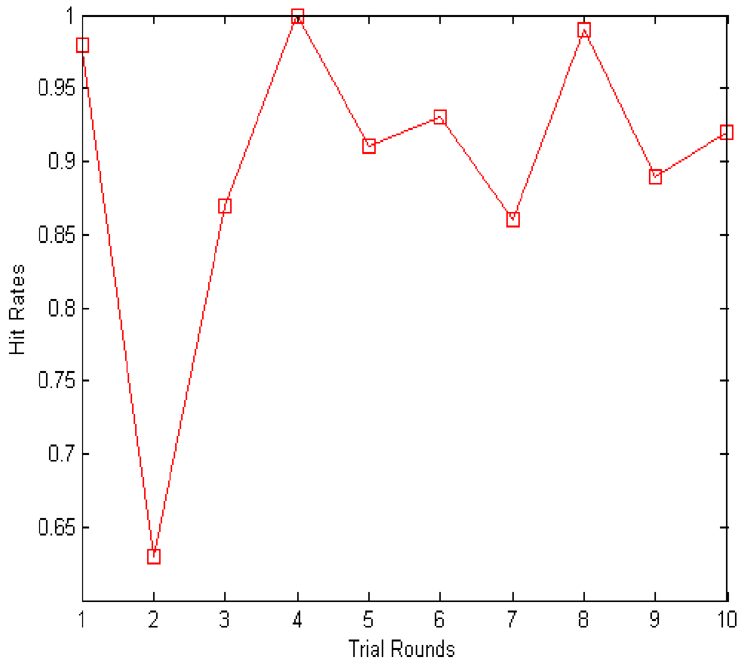


Figure 6.7: Hit rates observed empirically for various rounds of trials with the test botnet in the Emulab network testbed.

Chapter 7

Overview of *variant*: A Botnet Mitigation Tool

The *variant* tool is a workable and functional prototype implementation of the research discussed in this report. The *variant* tool produces consistent results each time it is run. Our motivation behind choosing the name *variant* for this tool lies in the fact that botnet dynamics targeted by this research vary among network infections. Even the same botnet launched in the same network several times may exhibit different dynamics at each one of those individual launches. That phenomenon is due to the stochastic nature of a botnet infection process that belongs to the category of malware considered in this research. As we wrote earlier in this report, the *variant* tool takes in input pcap files generated by tcpdump or any other derivative network sniffing tool. In the case of a positive botnet interception, the output produced by the *variant* tool consists of a textual description of the final form of the weakly connected subgraph formed by the botnet infection process within the data graph until the moment of intervention.

Typical forms of that specific output are shown in the bottom half of Figure 7.1 and in the bottom half of Figure 7.6. In the case of absent formation of weakly connected subgraphs within the data graph, or in the case of lack of isomorphism between the model graph and weakly connected subgraphs within the data graph, the *variant* tool simply notifies that no botnets were detected. The *variant* tool can function in total independence from any human intervention, and thus can conduct automatic early state botnet detection and containment tasks. Nevertheless, we decided to equip the *variant* tool with an interactive interface in order to enable a network security analyst to explore the internals of each processing session. That interactive interface also helped us debug the overall approach as it progressed throughout a processing session.

The *variant* tool provides a command line interface to the network security analyst. The available commands and their possible respective options are shown in the top half of Figure 7.1. The *variant* tool is designed to process simultaneously several pcap files created by network sniffers deployed at various points in an enterprise network. The `netpackets` command sorts entries from those pcap files and places them in a unified file, which is then processed by other commands. The logit model that we developed in this research can be solved via the `logit` command as shown in Figures 7.2, 7.3, and 7.4. The botnet virulence can be estimated via the `virulence` command, which accepts three options, namely `-i`, `-v`, and `-b`. Those options allow for estimating the infection rate only, the vulnerability rate only, and both infection rate and vulnerability rate, respectively. A typical session is shown in Figure 7.5.

The various plots of statistical data that we used in this research to characterize the effectiveness of various components of our approach can be drawn via the `plot` command. That command is not required for conducting real-world botnet containment tasks. Nevertheless, we developed the functionality behind the command in question as we use the *variant* tool also as a research tool. The `propagation` command is used to display data on the botnet propagation process in an enterprise network. It is accompanied by two options, namely `-a` and `-d`. The `-a` option displays propagation data based on reports generated by bots and gathered from the collector program during an experiment in the Emulab network testbed. That option is only needed when the *variant* tool is employed as a research tool. The `-d` option displays propagation data that characterize botnet propagation dynamics as inferred by the *variant* tool.

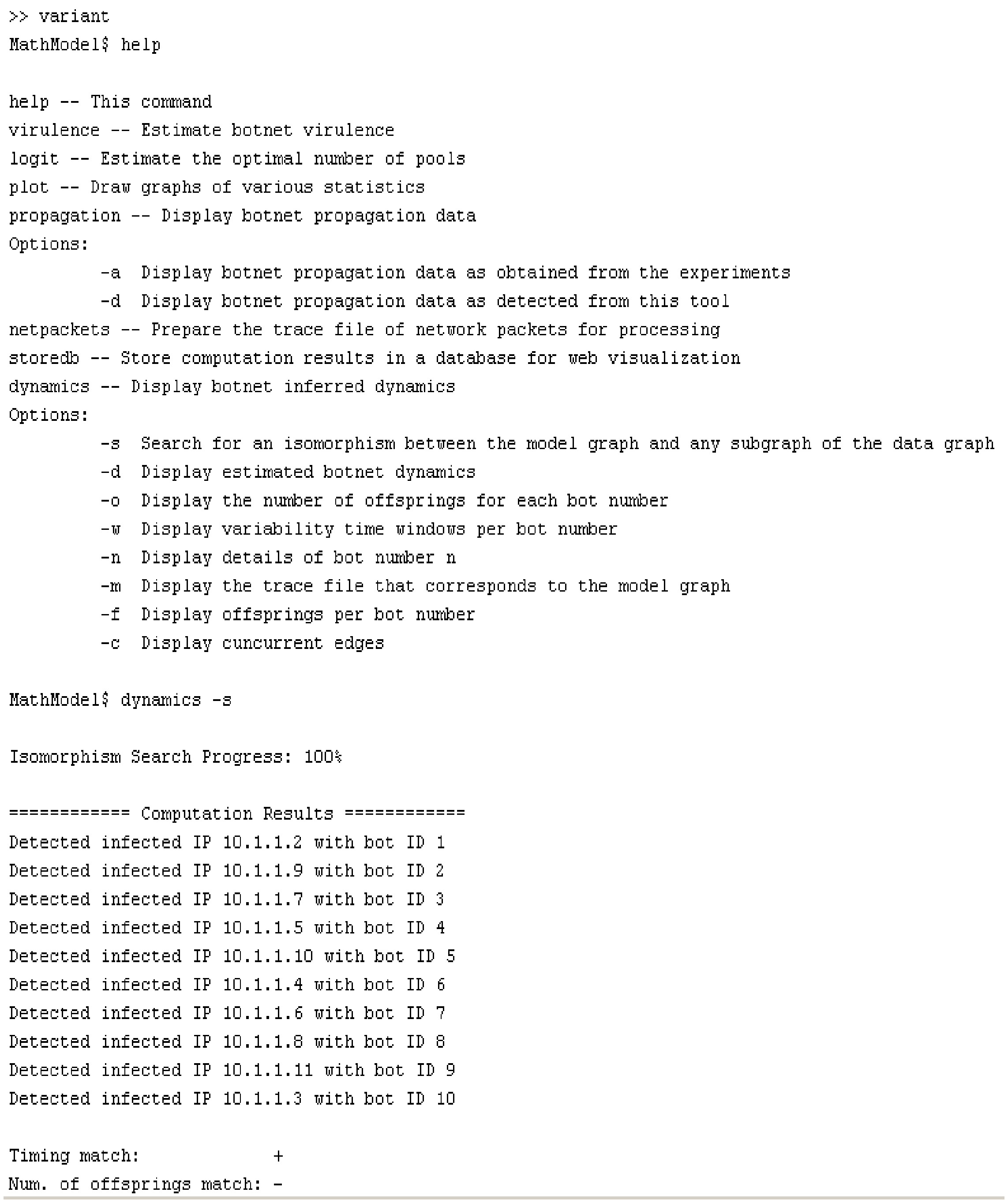
A typical session of displaying propagation data is shown in Figure 7.6. A detailed insight into the inferred dynamics can be obtained via the `dynamics` command. As we wrote earlier in this report, our overall approach is equipped with an incremental auto-corrective feature. The *variant* tool keeps intermediate findings as a table that it stores in a file named `IntermediateResults.dat`. Each entry in that table represents a specific phase or step within the inference process. The table in question is taken into account in full from the `dynamics` command, and thus the data displayed by that command are generated on the basis that the findings represented by entries in that table hold. In other words, the data displayed by the `dynamics` command correspond to the degree of progress made by the inference process, which is denoted by the last entry in the table in question.

If we need to display inferred dynamics at the beginning of the inference process, we would have to remove all possible entries from the table in question. Similarly, if we need to display inferred dynamics at the end of the inference process, we would have to have all table entries in place at the moment of running the `dynamics` command. The `-d` option allows for obtaining the infection time of each infected host as labeled by a bot number. A typical session is shown on top of Figure 7.7. The `-o` option allows for viewing the estimated number of offsprings for each infected host along with an estimation of the botnet infectious time and an estimated period of time during which the infected host can cause other infections in the enterprise network. The `-o` option also allows for viewing the average number of infection attempts that are predicted to originate from each infected host as labeled by a bot number.

A typical use of the `-o` option is shown in the middle of Figure 7.7. The variability time windows for each infected host can be displayed via the `-w` option, as shown on the bottom of Figure 7.7. The option formed by `-` and a bot number allows for viewing the estimated probability distribution of the number of offsprings of the infected host labeled with that specific bot number. A typical session involving the use of that option is shown on top of Figure 7.8. The `-f` option displays possible offsprings of each infected host in terms of bot numbers, as shown on the bottom of Figure 7.8. Mutually exclusive concurrent edges, i.e. concurrent parent-child relationships between infected hosts, can be viewed via the `-c` option, as shown in Figure 7.9. As we wrote earlier in this report, botnet dynamics are inferred by the Matlab code within the *variant* tool.

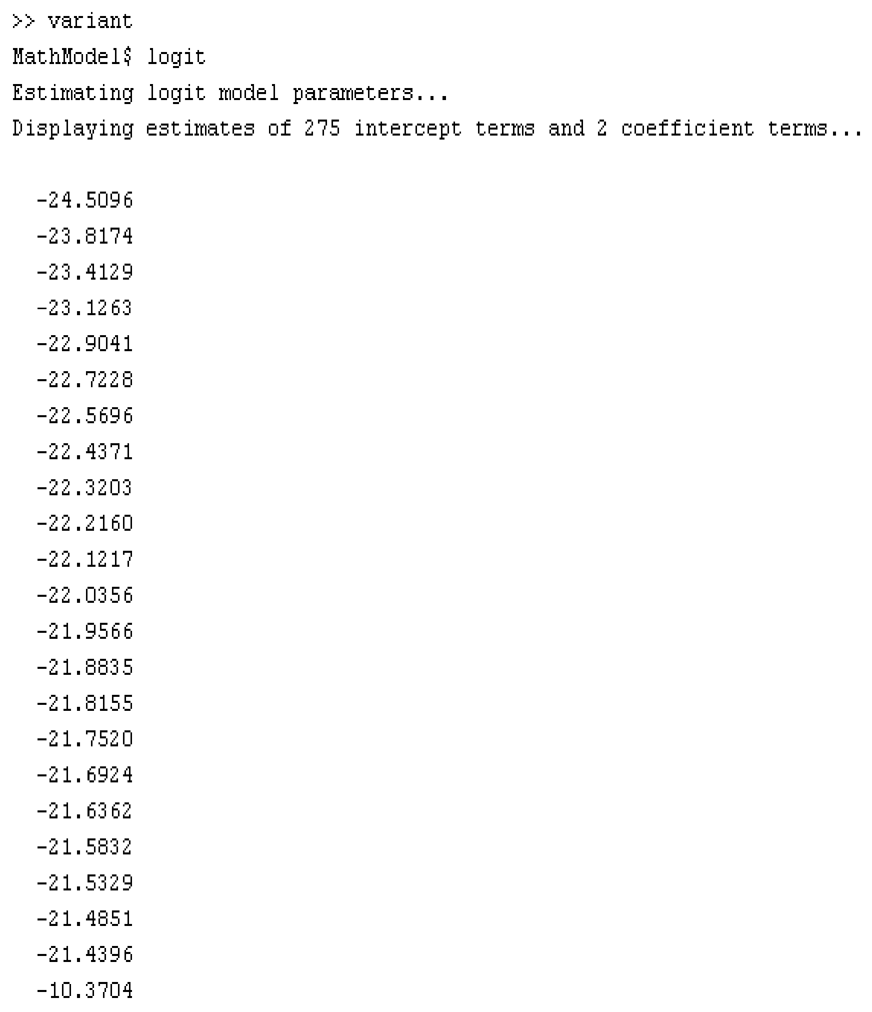
The inferred dynamics are written in a file named `ModelGraph.dat`. The Perl code within the *variant* tool reads the content of that file, and hence uses it to generate the model graph. The content in question can be viewed via the `-m` option, as shown in Figure 7.10. Each record of the file `ModelGraph.dat` is formatted as follows. The first field indicates the infection time of a host labeled by a bot number that is represented by the fifth field. The third field indicates a bot number that corresponds to the host that initiated the infection. The presence of an `X` instead of a bot number denotes that the host that initiated the infection is not located within the perimeter of the enterprise network. The sixth and the seventh fields indicate the boundaries of the variability time window that applies to the infected host. The last field indicates the estimated number of offsprings that the infected host is predicted to have after exhaustion of the vulnerable host population.

The subgraph isomorphism search is conducted via the `-s` option, as shown on the bottom of Figure 7.1. In conclusion, the network security analyst can issue the `storedb` command to store in a MySQL database all of the data discussed in this chapter. Those data can then be visualized via a web application, which lies outside the scope of this report.



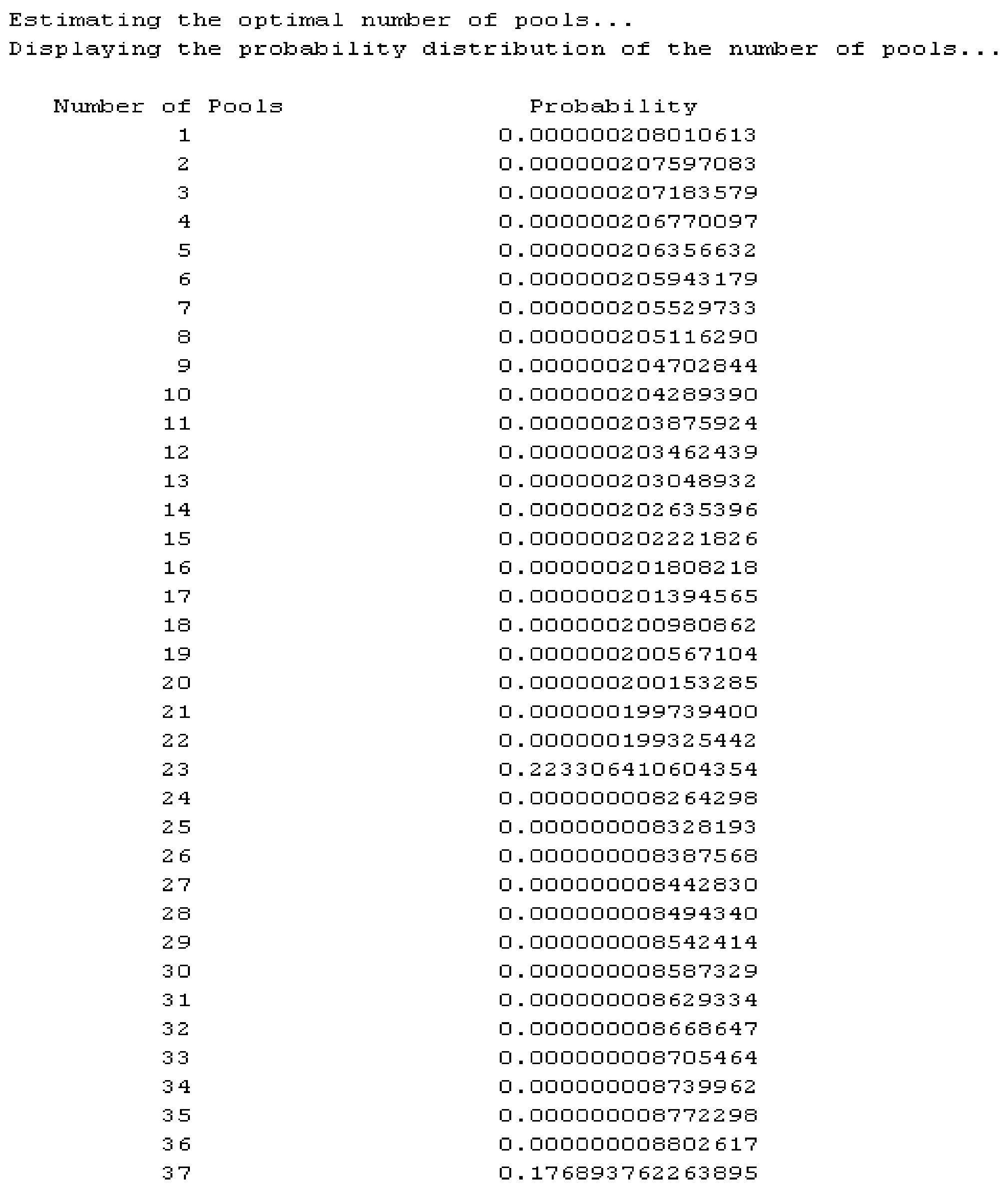
```
>> variant
MathModel$ help

help -- This command
virulence -- Estimate botnet virulence
logit -- Estimate the optimal number of pools
plot -- Draw graphs of various statistics
propagation -- Display botnet propagation data
Options:
         -a  Display botnet propagation data as obtained from the experiments
         -d  Display botnet propagation data as detected from this tool
netpackets -- Prepare the trace file of network packets for processing
storedb -- Store computation results in a database for web visualization
dynamics -- Display botnet inferred dynamics
Options:
         -s  Search for an isomorphism between the model graph and any subgraph of the data graph
         -d  Display estimated botnet dynamics
         -o  Display the number of offsprings for each bot number
         -w  Display variability time windows per bot number
         -n  Display details of bot number n
         -m  Display the trace file that corresponds to the model graph
         -f  Display offsprings per bot number
         -c  Display cuncurrent edges

MathModel$ dynamics -s

Isomorphism Search Progress: 100%

============ Computation Results ============
Detected infected IP 10.1.1.2 with bot ID 1
Detected infected IP 10.1.1.9 with bot ID 2
Detected infected IP 10.1.1.7 with bot ID 3
Detected infected IP 10.1.1.5 with bot ID 4
Detected infected IP 10.1.1.10 with bot ID 5
Detected infected IP 10.1.1.4 with bot ID 6
Detected infected IP 10.1.1.6 with bot ID 7
Detected infected IP 10.1.1.8 with bot ID 8
Detected infected IP 10.1.1.11 with bot ID 9
Detected infected IP 10.1.1.3 with bot ID 10

Timing match:             +
Num. of offsprings match: -
```

Figure 7.1: Command line interface of the *variant* tool along with a typical botnet search session.



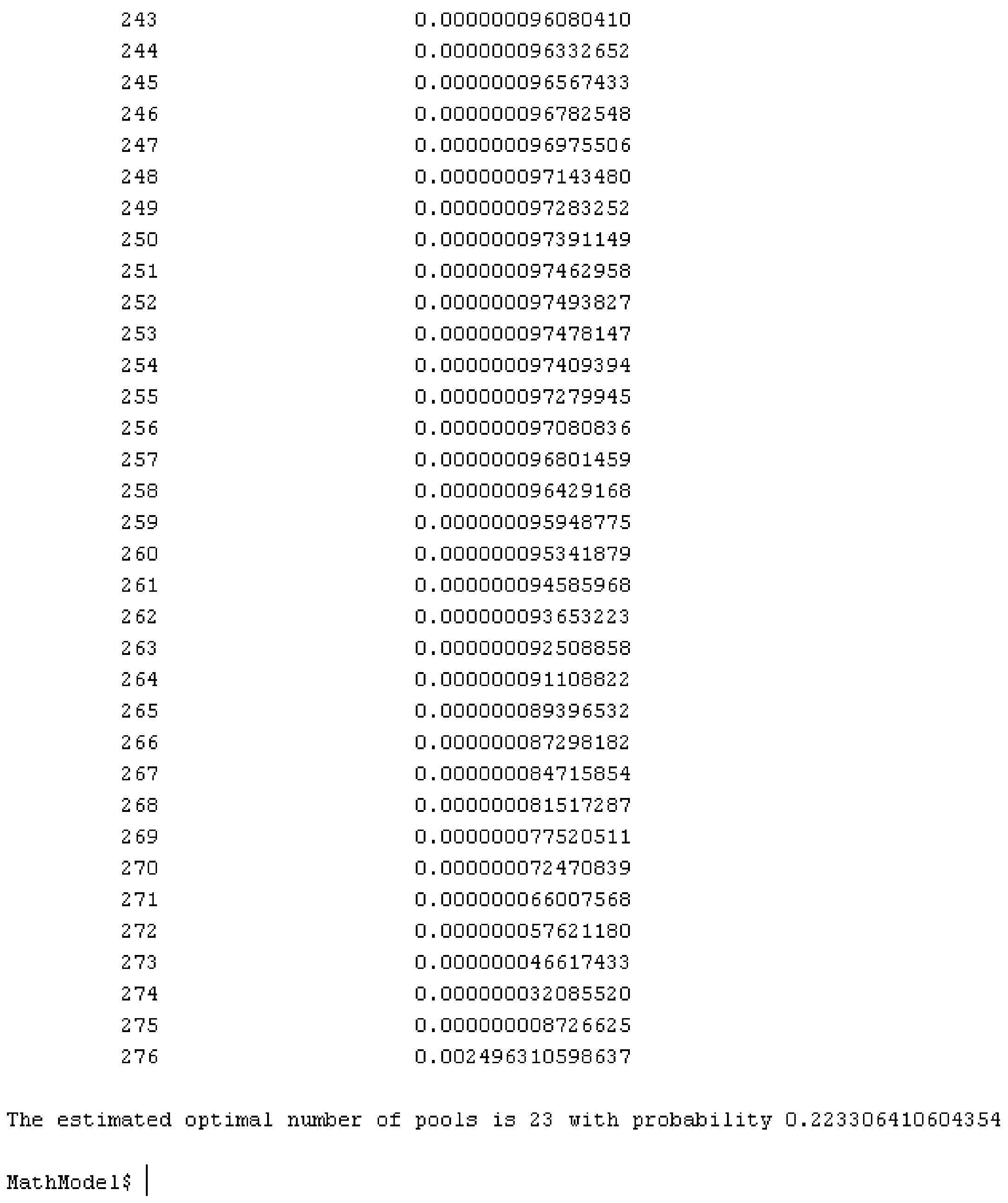
```
>> variant
MathModel$ logit
Estimating logit model parameters...
Displaying estimates of 275 intercept terms and 2 coefficient terms...

  -24.5096
  -23.8174
  -23.4129
  -23.1263
  -22.9041
  -22.7228
  -22.5696
  -22.4371
  -22.3203
  -22.2160
  -22.1217
  -22.0356
  -21.9566
  -21.8835
  -21.8155
  -21.7520
  -21.6924
  -21.6362
  -21.5832
  -21.5329
  -21.4851
  -21.4396
  -10.3704
```

Figure 7.2: Estimation of the intercept terms and coefficient terms of the logit model.



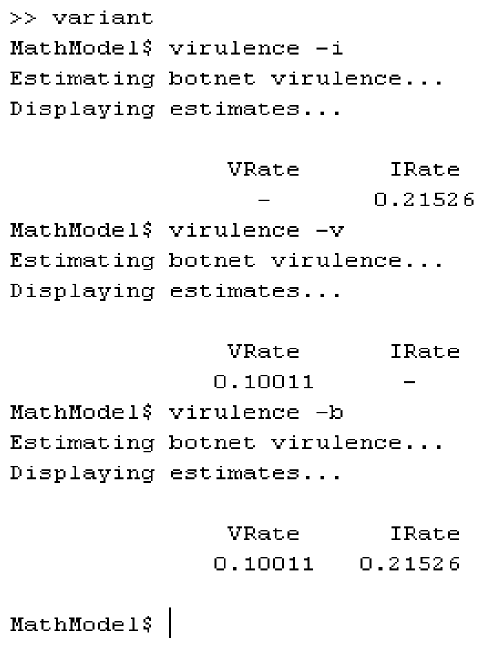
```
Estimating the optimal number of pools...
Displaying the probability distribution of the number of pools...

   Number of Pools                 Probability
           1                   0.000000208010613
           2                   0.000000207597083
           3                   0.000000207183579
           4                   0.000000206770097
           5                   0.000000206356632
           6                   0.000000205943179
           7                   0.000000205529733
           8                   0.000000205116290
           9                   0.000000204702844
          10                   0.000000204289390
          11                   0.000000203875924
          12                   0.000000203462439
          13                   0.000000203048932
          14                   0.000000202635396
          15                   0.000000202221826
          16                   0.000000201808218
          17                   0.000000201394565
          18                   0.000000200980862
          19                   0.000000200567104
          20                   0.000000200153285
          21                   0.000000199739400
          22                   0.000000199325442
          23                   0.223306410604354
          24                   0.000000008264298
          25                   0.000000008328193
          26                   0.000000008387568
          27                   0.000000008442830
          28                   0.000000008494340
          29                   0.000000008542414
          30                   0.000000008587329
          31                   0.000000008629334
          32                   0.000000008668647
          33                   0.000000008705464
          34                   0.000000008739962
          35                   0.000000008772298
          36                   0.000000008802617
          37                   0.176893762263895
```

Figure 7.3: Probability distribution of the optimal number of pools.



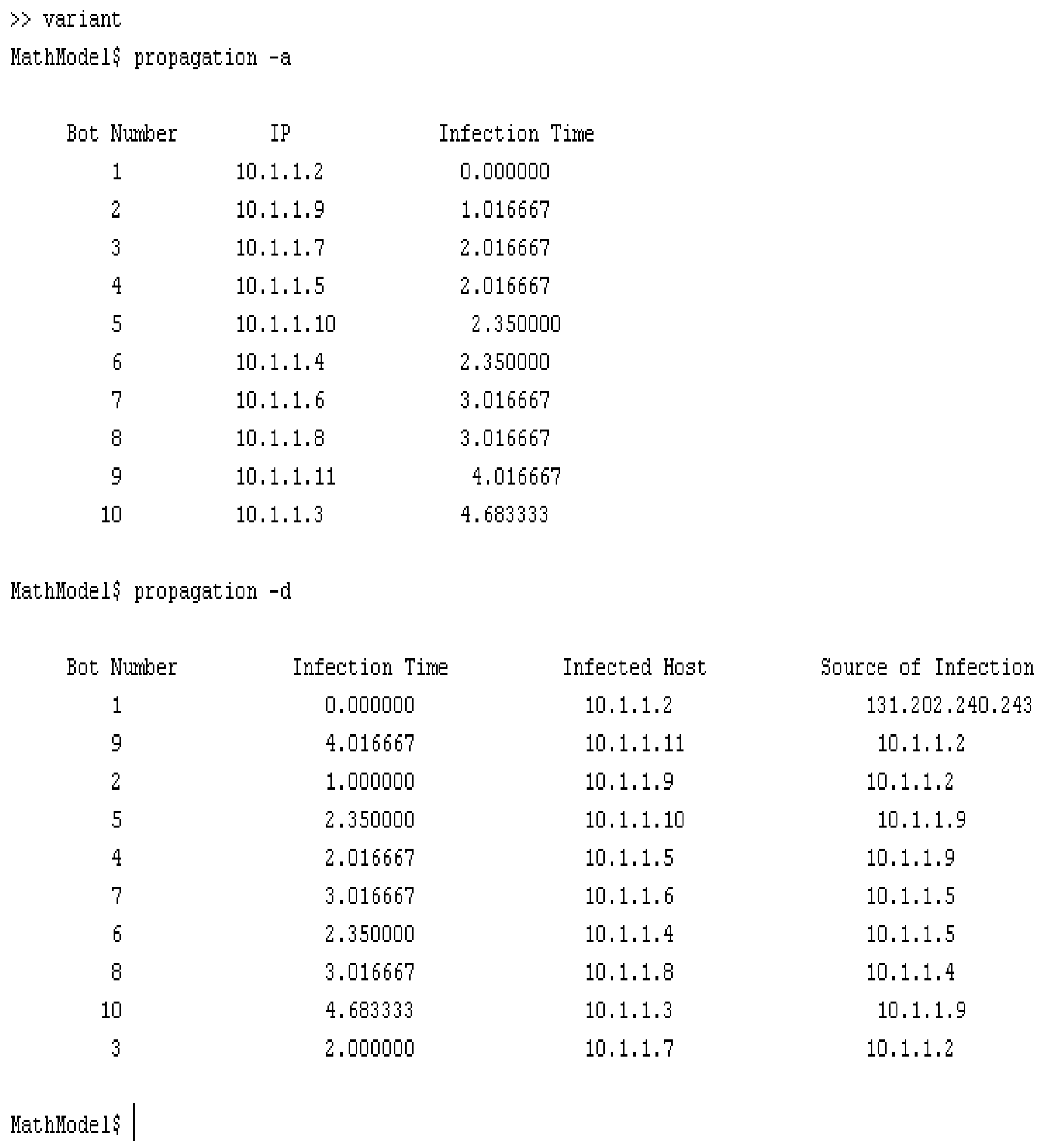
```
        243                     0.000000096080410
        244                     0.000000096332652
        245                     0.000000096567433
        246                     0.000000096782548
        247                     0.000000096975506
        248                     0.000000097143480
        249                     0.000000097283252
        250                     0.000000097391149
        251                     0.000000097462958
        252                     0.000000097493827
        253                     0.000000097478147
        254                     0.000000097409394
        255                     0.000000097279945
        256                     0.000000097080836
        257                     0.000000096801459
        258                     0.000000096429168
        259                     0.000000095948775
        260                     0.000000095341879
        261                     0.000000094585968
        262                     0.000000093653223
        263                     0.000000092508858
        264                     0.000000091108822
        265                     0.000000089396532
        266                     0.000000087298182
        267                     0.000000084715854
        268                     0.000000081517287
        269                     0.000000077520511
        270                     0.000000072470839
        271                     0.000000066007568
        272                     0.000000057621180
        273                     0.000000046617433
        274                     0.000000032085520
        275                     0.000000008726625
        276                     0.002496310598637

The estimated optimal number of pools is 23 with probability 0.223306410604354

MathModel$ |
```

Figure 7.4: Reporting the optimal number of pools.



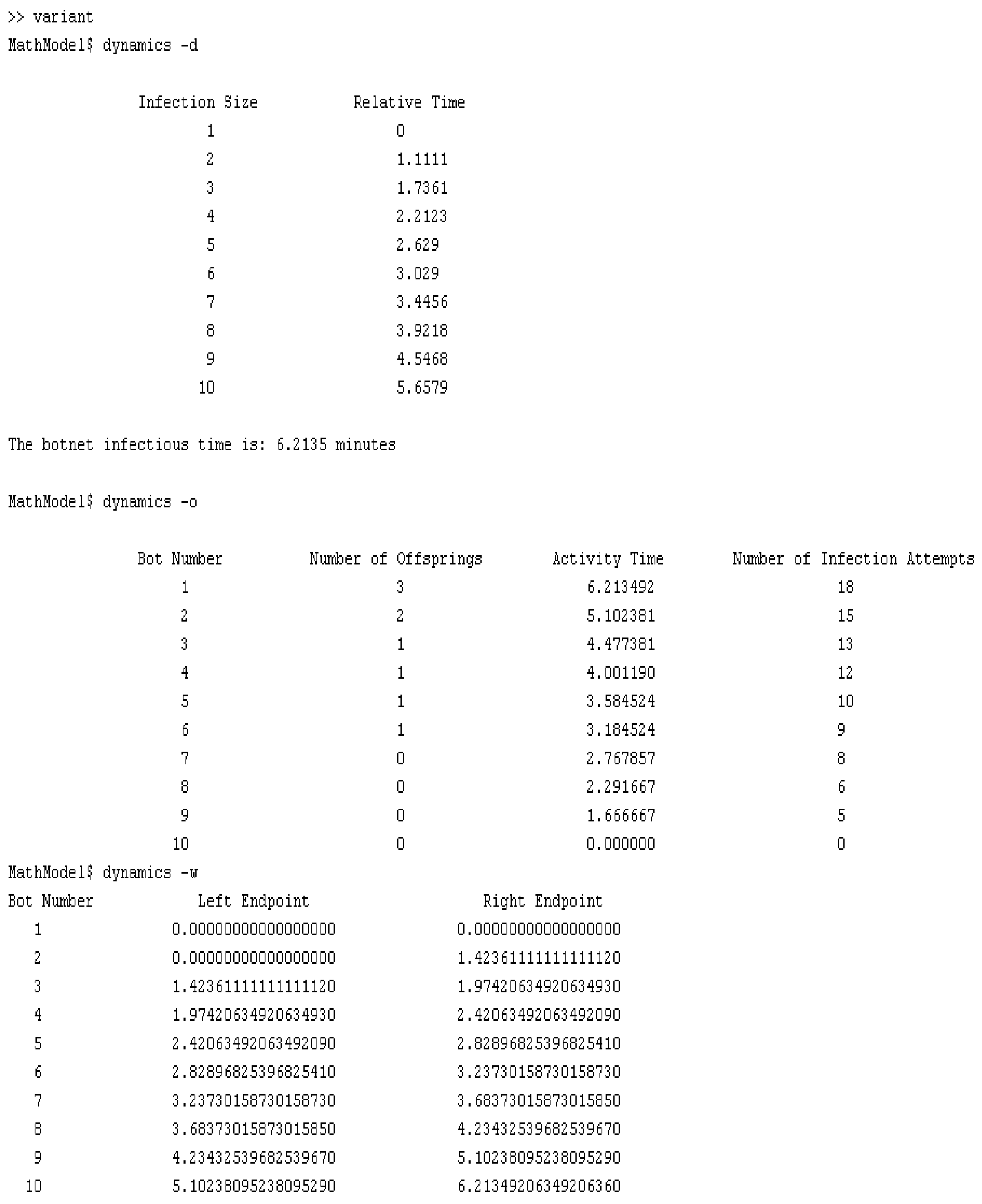
```
>> variant
MathModel$ virulence -i
Estimating botnet virulence...
Displaying estimates...

                VRate       IRate
                  -        0.21526
MathModel$ virulence -v
Estimating botnet virulence...
Displaying estimates...

                VRate       IRate
               0.10011        -
MathModel$ virulence -b
Estimating botnet virulence...
Displaying estimates...

                VRate       IRate
               0.10011    0.21526

MathModel$ |
```

Figure 7.5: A session of botnet virulence estimation.



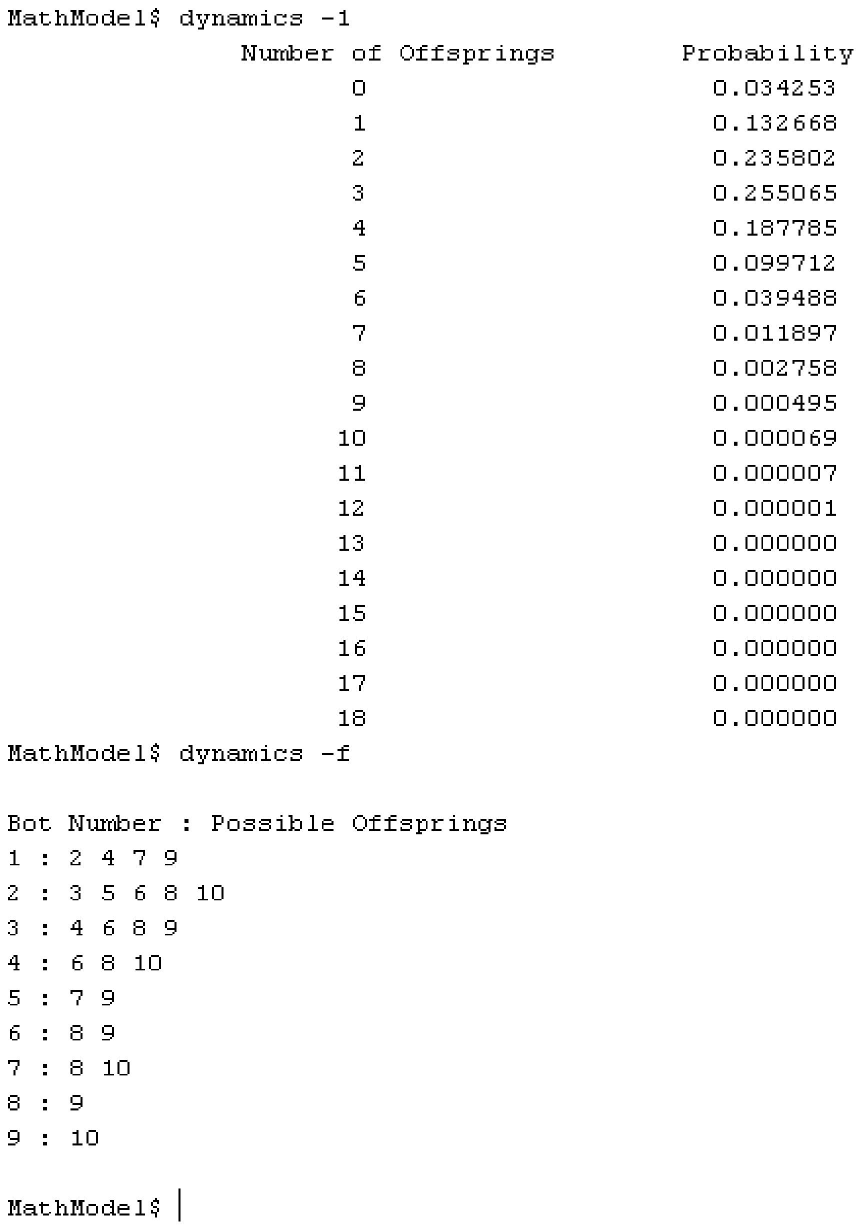
```
>> variant
MathModel$ propagation -a

     Bot Number        IP            Infection Time
         1          10.1.1.2           0.000000
         2          10.1.1.9           1.016667
         3          10.1.1.7           2.016667
         4          10.1.1.5           2.016667
         5          10.1.1.10           2.350000
         6          10.1.1.4           2.350000
         7          10.1.1.6           3.016667
         8          10.1.1.8           3.016667
         9          10.1.1.11           4.016667
        10          10.1.1.3           4.683333

MathModel$ propagation -d

     Bot Number          Infection Time          Infected Host          Source of Infection
         1                  0.000000               10.1.1.2                131.202.240.243
         9                  4.016667               10.1.1.11                10.1.1.2
         2                  1.000000               10.1.1.9                10.1.1.2
         5                  2.350000               10.1.1.10                10.1.1.9
         4                  2.016667               10.1.1.5                10.1.1.9
         7                  3.016667               10.1.1.6                10.1.1.5
         6                  2.350000               10.1.1.4                10.1.1.5
         8                  3.016667               10.1.1.8                10.1.1.4
        10                  4.683333               10.1.1.3                 10.1.1.9
         3                  2.000000               10.1.1.7                10.1.1.2

MathModel$ |
```

Figure 7.6: Botnet propagation characteristics as reported by bots during experimental network infections (top), and as detected from the `variant` tool (bottom).



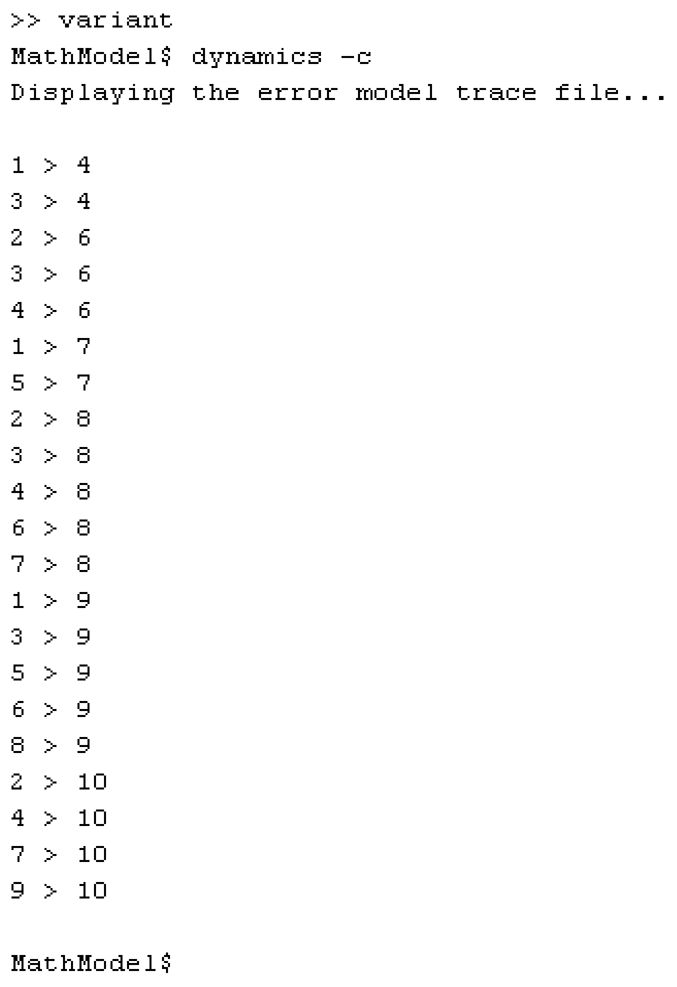
```
>> variant
MathModel$ dynamics -d

               Infection Size          Relative Time
                      1                     0
                      2                     1.1111
                      3                     1.7361
                      4                     2.2123
                      5                     2.629
                      6                     3.029
                      7                     3.4456
                      8                     3.9218
                      9                     4.5468
                     10                     5.6579

The botnet infectious time is: 6.2135 minutes

MathModel$ dynamics -o

               Bot Number          Number of Offsprings        Activity Time        Number of Infection Attempts
                   1                         3                    6.213492                     18
                   2                         2                    5.102381                     15
                   3                         1                    4.477381                     13
                   4                         1                    4.001190                     12
                   5                         1                    3.584524                     10
                   6                         1                    3.184524                     9
                   7                         0                    2.767857                     8
                   8                         0                    2.291667                     6
                   9                         0                    1.666667                     5
                  10                         0                    0.000000                     0
MathModel$ dynamics -w
Bot Number             Left Endpoint                   Right Endpoint
   1             0.00000000000000000             0.00000000000000000
   2             0.00000000000000000             1.42361111111111120
   3             1.42361111111111120             1.97420634920634930
   4             1.97420634920634930             2.42063492063492090
   5             2.42063492063492090             2.82896825396825410
   6             2.82896825396825410             3.23730158730158730
   7             3.23730158730158730             3.68373015873015850
   8             3.68373015873015850             4.23432539682539670
   9             4.23432539682539670             5.10238095238095290
  10             5.10238095238095290             6.21349206349206360
```

Figure 7.7: A portion of botnet dynamics inferred statistically by the `variant` tool.



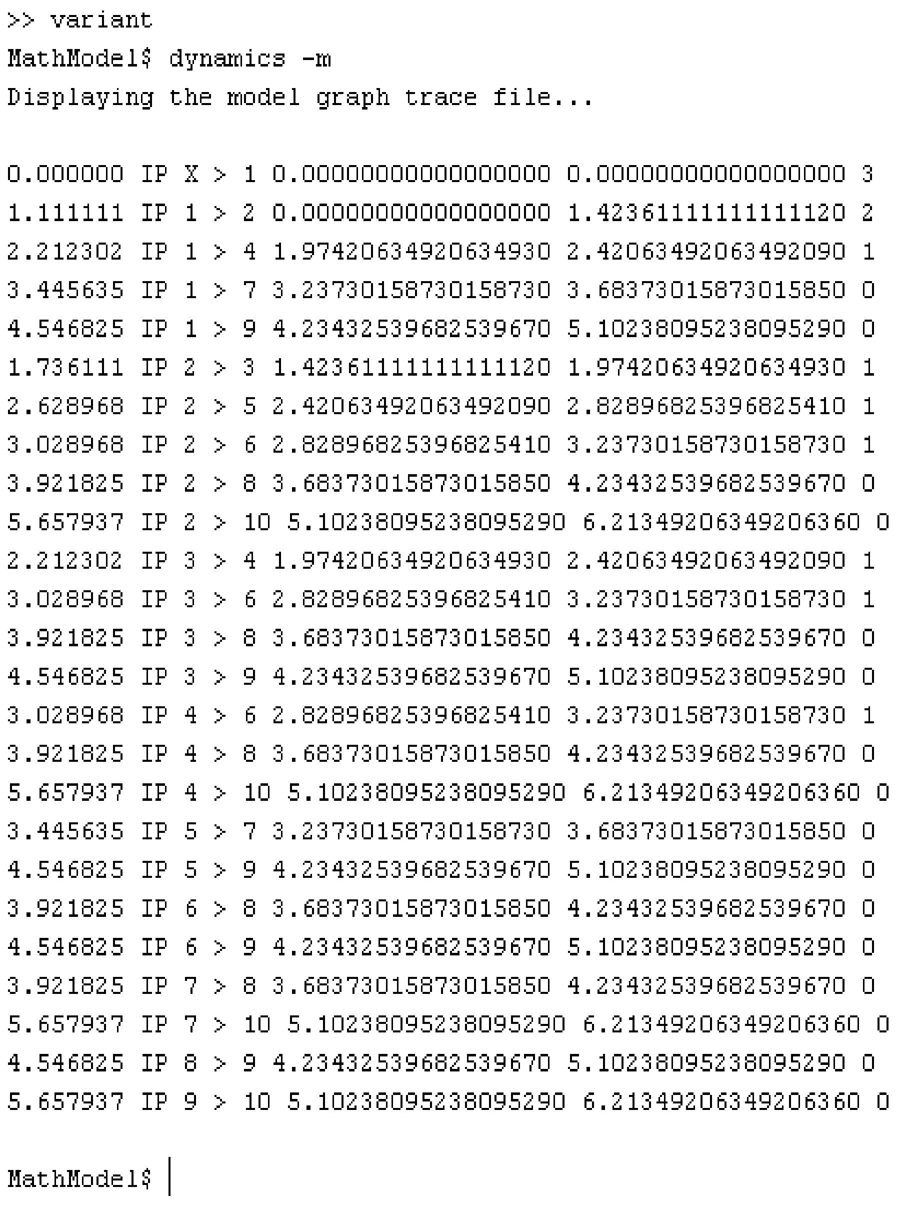
```
MathModel$ dynamics -1
               Number of Offsprings        Probability
                      0                      0.034253
                      1                      0.132668
                      2                      0.235802
                      3                      0.255065
                      4                      0.187785
                      5                      0.099712
                      6                      0.039488
                      7                      0.011897
                      8                      0.002758
                      9                      0.000495
                     10                      0.000069
                     11                      0.000007
                     12                      0.000001
                     13                      0.000000
                     14                      0.000000
                     15                      0.000000
                     16                      0.000000
                     17                      0.000000
                     18                      0.000000
MathModel$ dynamics -f

Bot Number : Possible Offsprings
1 : 2 4 7 9
2 : 3 5 6 8 10
3 : 4 6 8 9
4 : 6 8 10
5 : 7 9
6 : 8 9
7 : 8 10
8 : 9
9 : 10

MathModel$ |
```

Figure 7.8: Probability distribution of the number of offsprings of an infected host (top), and possible offsprings of each infected host in terms of bot numbers (bottom).



```
>> variant
MathModel$ dynamics -c
Displaying the error model trace file...

1 > 4
3 > 4
2 > 6
3 > 6
4 > 6
1 > 7
5 > 7
2 > 8
3 > 8
4 > 8
6 > 8
7 > 8
1 > 9
3 > 9
5 > 9
6 > 9
8 > 9
2 > 10
4 > 10
7 > 10
9 > 10

MathModel$
```

Figure 7.9: Mutually exclusive concurrent edges in the model graph.



```
>> variant
MathModel$ dynamics -m
Displaying the model graph trace file...

0.000000 IP X > 1 0.0000000000000000 0.0000000000000000 3
1.111111 IP 1 > 2 0.0000000000000000 1.4236111111111120 2
2.212302 IP 1 > 4 1.9742063492063493 2.4206349206349209 1
3.445635 IP 1 > 7 3.2373015873015873 3.6837301587301585 0
4.546825 IP 1 > 9 4.2343253968253967 5.1023809523809529 0
1.736111 IP 2 > 3 1.4236111111111120 1.9742063492063493 1
2.628968 IP 2 > 5 2.4206349206349209 2.8289682539682541 1
3.028968 IP 2 > 6 2.8289682539682541 3.2373015873015873 1
3.921825 IP 2 > 8 3.6837301587301585 4.2343253968253967 0
5.657937 IP 2 > 10 5.1023809523809529 6.2134920634920636 0
2.212302 IP 3 > 4 1.9742063492063493 2.4206349206349209 1
3.028968 IP 3 > 6 2.8289682539682541 3.2373015873015873 1
3.921825 IP 3 > 8 3.6837301587301585 4.2343253968253967 0
4.546825 IP 3 > 9 4.2343253968253967 5.1023809523809529 0
3.028968 IP 4 > 6 2.8289682539682541 3.2373015873015873 1
3.921825 IP 4 > 8 3.6837301587301585 4.2343253968253967 0
5.657937 IP 4 > 10 5.1023809523809529 6.2134920634920636 0
3.445635 IP 5 > 7 3.2373015873015873 3.6837301587301585 0
4.546825 IP 5 > 9 4.2343253968253967 5.1023809523809529 0
3.921825 IP 6 > 8 3.6837301587301585 4.2343253968253967 0
4.546825 IP 6 > 9 4.2343253968253967 5.1023809523809529 0
3.921825 IP 7 > 8 3.6837301587301585 4.2343253968253967 0
5.657937 IP 7 > 10 5.1023809523809529 6.2134920634920636 0
4.546825 IP 8 > 9 4.2343253968253967 5.1023809523809529 0
5.657937 IP 9 > 10 5.1023809523809529 6.2134920634920636 0

MathModel$ |
```

Figure 7.10: File content serving as basis for building the model graph.

Chapter 8

Literature Review

8.1 Virulence Estimation

Dagon et al. (Dagon *et al.*, 2006) discuss a model of botnet propagation that shows how time zones and hence time and geographic location affect malware spread dynamics. This model relies on estimates of the number of vulnerable hosts and infected hosts in a time zone, among other factors, and hence on the vulnerability and infection rates in the network of networks that extends over that time zone. All these parameters are calculated empirically from botnet data gathered over a period of six months via a sinkhole. Those empirical estimates reflect the specific vulnerabilities exploited by the botnets that were subject to the post-mortem study, and thus are quite reusable in the case the mitigation efforts are directed against a botnet that exploits the same vulnerabilities as those botnets.

Our approach to vulnerability rate estimation complements the model of Dagon et al. in that it creates the conditions for this model to be applied against any botnet, regardless of whether or not that botnet exploits the same vulnerabilities as the botnets that were subject to the post-mortem study. The most recent related work as of this writing consists in a logic matrix approach to modeling the propagation of worms in P2P networks provided by Fan and Xiang in (Fan and Xiang, 2010). The worm propagation model of Fan and Xiang is based on P2P network vulnerability and infection rates, which the authors consider as follows. If n is the total number of peers in a P2P network, the authors use a row logic vector V of length n such as each element v_j of V, for $j \in \{1, 2, ..., n\}$, indicates whether or not peer j in the P2P network is vulnerable to the worm.

Vulnerability rate is the ratio of the number of elements of V that indicate the corresponding peer is vulnerable to the total number of peers in the P2P network of reference, namely n. Similarly, the authors use a row logic vector S of length n with its element s_j indicating whether or not peer j in the P2P network has been infected by the worm. Infection rate is the ratio of the number of elements of S that indicate the corresponding peer is infected to n. The practicality of the worm propagation model of Fan and Xiang is limited as it requires inspection of each individual peer in a P2P network to determine: a) whether or not the peer is vulnerable to the worm, and b) whether or not the peer is infected by the worm. In P2P networks whose size is not small, as is the case of most real world P2P networks, inspecting each individual peer is not practical.

Furthermore, point b is equivalent to estimating the total population of infected peers, which is a task that is known to require considerable time (Dagon *et al.*, 2006), and hence does not allow for a timely response. Our work complements the worm propagation model of Fan and Xiang as it manages to estimate vulnerability rates via random sampling, and thus does not require individual inspection of each node in a network of reference. In (Chen and Ji, 2005), Chen and Ji discuss a model of the propagation dynamics of malware that spread by taking into account network topology. The authors use the concept of malware network, which is constructed by removing invulnerable nodes and the edges associated with these nodes in a network of reference.

Given the topology of a malware network, infection rates, i.e. rates at which infected nodes can infect their vulnerable neighbors, death rates, i.e. rates at which infected nodes become vulnerable, and an initial infected node, the model of Chen and Ji estimates the number of infected nodes at a given point in time t. The model is based on a directed graph representation of a malware network, in which each edge is associated with a vulnerability rate and each node is associated with a death rate. When applied to a concrete computer network, the model requires measures of vulnerability rates and death rates for each couple of neighboring nodes in the corresponding malware network, respectively.

Thus, the model requires examination of each couple of neighboring nodes in the malware network. Furthermore, construction of the malware network out of a network of reference requires inspection of each individual node in the network to determine whether or not it is vulnerable. From the practicality perspective, the limitation of the model stems from the two aforementioned observations. Besides malware and hence bot propagation models, effective target scanning techniques such as those discussed in (Chen and Ji, 2007; Zou *et al.*, 2005) that increase malware virulence rely directly or indirectly on vulnerability and/or infection rates. In (Zou *et al.*, 2005), Zou et al. leverage the so called simple epidemic model provided by Daley and Gani in (Daley and Gani, 1999) to devise a technique that a worm can use to reduce its scanning space without ignoring potential vulnerable nodes.

That scanning technique relies on measures such as the number of infected nodes at a specific point in time t along with the number of vulnerable nodes. Zou et al. do not estimate those measures, but assume that they are available, and thus demonstrate the effectiveness of their scanning technique upon estimates taken from post-mortem studies of the Code Red worm. In (Chen and Ji, 2007), Chen and Ji provide the design of an optimal scanning technique that leverages knowledge of the vulnerable-host distribution in a network of reference. The authors do not calculate estimates of vulnerable-host distributions, but assume that those distributions are available or obtainable. The authors validate their optimal scanning technique using estimates extracted from post-mortem studies of the Witty and Code Red worms. In conclusion, from the mitigation perspective the scanning techniques of Zou et al. in (Zou *et al.*, 2005) and Chen and Ji in (Chen and Ji, 2007) are subject to the same applicability limitation as the work of Dagon et al. in (Dagon *et al.*, 2006).

8.2 Botnet Mitigation

Nagaraja et al. (Nagaraja *et al.*, 2010) conduct graph theoretical analysis of communication graphs in order to identify hosts that are infected by bots. A communication graph is defined as an undirected graph in which vertices represent hosts and edges represent network communications between hosts. Communication networks are built upon traces of network packets sniffed from the backbone network of an ISP or from an enterprise network. Nagaraja et al. leverage the fact that P2P botnets are making use of network overlay structures (Stover *et al.*, 2007; Porras *et al.*, 2007, 2009). Those network overlay structures provide for low delay network communication from any infected host to any other infected host via low-latency paths, low maintenance overhead, robustness in the face of network churn, and tolerance to infected host failures.

Furthermore, the lack of centralization in such network overlay structures enables a botmaster to join and command the botnet from any arbitrary infected host, and thus evade tracking. Nagaraja et al. develop an inference algorithm that searches for network overlay structures within a communication graph, and thus marks the hosts that are members of any such structures localized within the communication graph as being suspected of carrying a bot. Although the accuracy of the overall approach of Nagaraja et al. is not the best possible, that approach can be used in conjunction with existing botnet detection techniques in order to complement them (Nagaraja *et al.*, 2010). The main advantage of our research lies in earlier identification of the subgraph that comprises hosts infected by bots. The botnet detection approach of Nagaraja et al. does not detect infected hosts until they settle to network overlay structures that are visible from the network traffic monitors deployed.

Such waiting time window allows the occurrence of host infections that are conducted during the amount of time which elapses from the first few host infections in the network of reference to the establishment of network overlay structures within the communication graph. Our research identifies the subgraph that comprises infected hosts from the first few host infections in the network of reference, and thus is not subject to such as large waiting time window. From a botnet containment perspective, the concrete gain that is obtained from our approach consists in avoiding the occurrence of those host infections which would have taken place throughout the waiting time window. Our research has a similar advantage over the botnet detection and bot identification approach of Collins and Reiter described in (Collins and Reiter, 2007). That approach applies to bots that employ hit-lists for target selection (Staniford *et al.*, 2002), and employs protocol graph modeling.

Protocol graphs are similar to the communication graphs of Nagaraja et al., with the only difference being that the edges in a protocol graph represent network communications between hosts using a specific application protocol. Collins and Reiter postulate that hit-list bot infections either inflate the number of vertices in a communication graph or connect disjoint connected subgraphs of a communication graph. Their empirical observations of network traffic in a large network indicate that: (1) protocol graphs for application protocols such as HTTP, FTP, SMTP, and Oracle have predictable sizes; and (2) the size of the largest connected subgraph of each one of those protocol graphs is also predictable. Collins and Reiter employs those predictable sizes as threshold sizes. The presence of a botnet is detected by deviations of those two graph properties from the threshold sizes.

Once a botnet is detected, Collins and Reiter determine which vertices in a protocol graph represent infected hosts by counting the number of connected subgraphs that are created after removal of high degree vertices. That technique is based on the postulate that removing a vertex which represents an infected host from a protocol graph causes a separation of connected subgraphs, and hence increases the number of overall connected subgraphs in the protocol graph. Similarly to the approach of Nagaraja et al., the approach of Collins and Reiter is also subject to a waiting time window. That approach does not detect a botnet and hence identify bots until the number of vertices in a protocol graph or the number of vertices in the largest connected subgraph of that protocol graph exceeds preliminarily predicted threshold sizes. The necessities of hosts in a network of reference to communicate with other hosts change dynamically according to the inner working of system services deployed and the decision making of host users.

For the threshold sizes to accommodate such dynamism, those threshold sizes have to tolerate infrequent but quite possible increments of the number of vertices in a protocol graph or of the number of vertices in the largest connected subgraph of that protocol graph. The waiting time window consists in the amount of time that elapses during the occurrence of increments of those two graph properties which are due to host infections rather than the dynamism in question until the threshold sizes are exceeded. In (Ellis *et al.*, 2004), Ellis et al. construct abstract communication networks to model end-to-end network communications between hosts. Abstract communication networks are defined similarly to the communication graphs of Nagaraja et al., with the addition that the edges in an abstract communication network are directed and also preserve the features of the network communications that they represent, including headers, payload, and timing.

Furthermore, there may be several edges between any two vertices in an abstract communication network. Ellis et al. define a worm propagation network as that part of an abstract communication network that pertains to network communications generated by the propagation of a worm. Ellis et al. postulate that a worm propagation network forms a set of spanning trees with specific properties, and thus define thresholds for those properties with respect to specific edge predicates in order to identify the worm propagation network. Examples of the spanning tree properties in question include the depth of the furthest descendant of a given vertex at specific points in time, the overall number of descendants of a given vertex, branching quantification, and the average time from one generation of vertices to the next. Our research follows a similar path as the worm detection approach of Ellis et al. in that we construct a graph that models end-to-end network communications between hosts in a network of reference, and thus identify the subgraph that models the propagation of bots.

Our research complements the research of Ellis et al. The authors do not provide in (Ellis *et al.*, 2004) a concrete method for estimating thresholds for the aforementioned spanning tree properties, while in our research we draw on the intersection of graph theory and probability theory to estimate in practice temporal and spatial graph properties, which in turn allow for identifying the subgraph that models the propagation of bots. The edge predicates in an abstract communication network are developed upon patterns in reconnaissance, protocol usage, and payload content, which persist over host infections. Consequently, constructing those edge predicates requires a detailed analysis of worm behavior, which would slow down any containment process. The time required for the overall worm behavior to become detectable would allow worms to propagate further (Gopalan *et al.*, 2006).

Our research exploits botnet propagation dynamics, and thus is not subject to that dependency. The botnet containment approach that we discuss in this report moves along the line of the Livshits and Cui's research on detection and containment of Javascript worms, which the authors discuss in (Livshits and Cui, 2008). Livshits and Cui employ a proxy to intercept, examine and possibly tag the pages that flow between a browser and a Web application. When a Web browser uploads content that includes HTML code, the proxy creates a tag, i.e. a long integer, adds the tag to the page, and thereafter allows the page augmented with the tag to be uploaded and hence stored at the server. When the tagged page is requested by a Web browser, the proxy examines the page and thus identifies the tag. The proxy creates a new session ID, which it associates with the tag, removes the tag from the page, and then passes the page augmented with the session ID to the Web browser.

The session ID is stored in a cookie within the Web browser. When the Web browser that originally requested the tagged page attempts to upload that page to the server, the cookie is sent along with the page in question. The proxy inspects the cookie and thus identifies the tag the cookie is associated with. The proxy then adds a new tag to the page, and hence creates a list of tags for that page. This is the point in which Livshits and Cui initially create and thereafter update propagation graphs. A propagation graph is a directed graph in which each vertex represents a tag and edges represent causality. With reference to our previous discussion, initially we would have a vertex that represents the old tag, a vertex that represents the new tag, and a directed edge that extends from the former to the latter. Each vertex carries the IP address of the client that the corresponding tag originates from.

The steps described above are iterated, but this time the page will be augmented with a list of tags and each iteration will possibly update the propagation graph. Overall, a propagation graph indicates propagation of a page between IP addresses.

Livshits and Cui define the diameter of a propagation graph as the maximum distance between any two vertices in that propagation graph, and thus postulate that a Javascript worm is detected when the diameter in question exceeds a user provided threshold. When a Javascript worm is detected, the proxy suppresses the aforementioned upload requests in order to prevent the Javascript worm from propagating any further. Similarly to the research of Livshits and Cui, we rely on graph theoretical representation and analysis to model and capture the propagation of bots in a network of reference, resulting in a containment approach founded on infected population detection.

The realm of the research of Livshits and Cui consists in a single Web site, given that the same-origin policy of the JavaScript language prohibits the spread of Javascript worms across multiple Web servers. Extending that realm to a computer network and hence targeting generic worms may not be practical. The challenge lies in content tagging, which would have consisted in determining whether or not specific content received from a host is the content which that host sends to other hosts. In other words, we would have needed to determine that some content is being propagated amongst hosts in a network of reference, and hence track that propagation by tagging content. Clearly polymorphism and metamorphism limit our ability to compare content in ingress and egress at a host. Research such as (Crandall *et al.*, 2005; Newsome *et al.*, 2005) indicates that there might be limits in the amount of polymorphism and metamorphism available to the attacker during a vulnerability exploitation, and thus even polymorphic or metamorphic exploits may contain multiple invariant substrings.

Nevertheless, generating signatures for those exploits in order to compare content in ingress and egress at a host establishes an equivalence between a possible application of the approach of Livshits and Cui to generic worms and existing local misuse approaches to worm or bot detection. Similarities or relations of content in ingress and egress at a host do not necessarily denote content propagation. In several application protocols such as FTP or SMTP input is similar to output (Ellis *et al.*, 2004). For example, a user may upload a file from a host to a FTP server, and that file may be downloaded at another host by another user later on. Furthermore, P2P applications and other common network applications such as Windows workgroups may exhibit worm-like network communication patterns (Gopalan *et al.*, 2006), and as such, make overall signatures of worm network communications quite ineffective.

Chapter 9

Conclusions

In this technical report we discussed a novel approach to the detection of early stage botnet infections in an enterprise network. That approach relies on mathematical modeling and operates from a botnet containment perspective. We showed how we use statistical inference and mathematical modeling to infer botnet propagation dynamics of a suspected botnet. We then discussed how we employ inferred dynamics within a subgraph isomorphism search process to localize infected hosts within weakly connected subgraphs formed within a graph representation of network communications between hosts. As this research relies on measures of network vulnerability rates, we devised a feasible statistical method that provides those measures in a timely fashion within the overall approach. That estimation method draws on epidemiological models in biology, is based on random sampling, and overall consists in a novel application of statistical learning and inference. The correctness of the estimation method in question lies in the validity of its underlying mathematical formulation and mathematical transformations that lead to workable estimates of network vulnerability rates. We have implemented our overall approach in practice in the Matlab and Perl programming languages, and thus in the report we discussed experimentation with this research in the Emulab network testbed. In the report we also discussed experiments with the vulnerability rate estimation method embedded within the overall approach in relation to simulated botnet propagation dynamics in a GTNetS network simulation platform. The ultimate objective of this research is to demonstrate the viability of mathematical models of botnet propagation dynamics as botnet mitigation tools. The experiments along with experimental data that we described in this technical report are indicative of the effectiveness of our overall approach, and also suggest a clear potential of mathematical models of botnet propagation dynamics for practical botnet mitigation.

Bibliography

Anderson, R. M., May, R. M. and Anderson, B. (1992) *Infectious Diseases of Humans: Dynamics and Control*. Oxford University Press, USA.

Andersson, H. and Britton, T. (2000) *Stochastic Epidemic Models and Their Statistical Analysis*, volume 151 of *Lecture Notes in Statistics*. Springer-Verlag, New York, USA.

Bhattacharyya, G. K., Karandinos, M. G. and DeFoliart, G. R. (1979) Point estimates and confidence intervals for infection rates using pooled organisms in epidemiologic studies. *American Journal of Epidemiology* **109**(2), 124–131.

Chen, Z., Gao, L. and Kwiat, K. (2003) Modeling the spread of active worms. In *Proceedings of the IEEE International Conference on Computer Communications*, pp. 1890–1900.

Chen, Z. and Ji, C. (2005) Spatial-temporal modeling of malware propagation in networks. *IEEE Transactions on Neural Networks (TNN): Special Issue on Adaptive Learning Systems in Communication Networks* **16**(5).

Chen, Z. and Ji, C. (2007) Optimal worm-scanning method using vulnerable-host distributions. *International Journal of Security and Networks (IJSN): Special Issue on Computer and Network Security* **2**(1/2).

Chiang, C. L. and Reeves, W. C. (1962) Statistical estimation of virus infection rates in mosquito vector populations. *American Journal of Hygiene* **75**, 377–391.

Choi, Y.-H., Li, L., Liu, P. and Kesidis, G. (2010) Worm virulence estimation for the containment of local worm outbreak. *Computers & Security* **29**(1), 104–123.

Collins, M. P. and Reiter, M. K. (2007) Hit-list worm detection and bot identification in large networks using protocol graphs. In *Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection*, pp. 276–295. Queensland, Australia.

Crandall, J. R., Su, Z., Wu, S. F. and Chong, F. T. (2005) On deriving unknown vulnerabilities from zero-day polymorphic and metamorphic worm exploits. In

Proceedings of the 12th ACM Conference on Computer and Communications Security, pp. 235–248. Virginia, USA.

Dagon, D., Zou, C. and Lee, W. (2006) Modeling botnet propagation using time zones. In *Proceedings of the 13th Annual Network and Distributed System Security Symposium*. San Diego, CA, USA.

Daley, D. J. and Gani, J. (1999) *Epidemic Modelling: An Introduction*. Cambridge University Press, Cambridge, United Kingdom.

Ellis, D. R., Aiken, J. G., Attwood, K. S. and D.Tenaglia, S. (2004) A behavioral approach to worm detection. In *Proceedings of the ACM Workshop on Rapid Malcode*, pp. 43–53. Washington DC, USA.

Fan, X. and Xiang, Y. (2010) Propagation modeling of peer-to-peer worms. In *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications*, pp. 1128–1135. Perth, Australia.

Fisher, R. A. (1922) On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London* **222**, 309–368.

Gopalan, P., Jamieson, K., Mavrommatis, P. and Poletto, M. (2006) Signature metrics for accurate and automated worm detection. In *Proceedings of the 4th ACM Workshop on Recurring Malcode*, pp. 65–72. Virginia, USA.

Hosmer, D. W. and Lemeshow, S. (2000) *Applied Logistic Regression*. Wiley InterScience, second edition.

Kleinbaum, D. G., Kupper, L. L., Nizam, A. and Muller, K. E. (2007) *Applied Regression Analysis and Multivariable Methods*. Duxbury Press, fourth edition.

Liljenstam, M., Nicol, D. M., Berk, V. H. and Gray, R. S. (2003) Simulating realistic network worm traffic for worm warning system design and testing. In *Proceedings of the ACM Workshop on Rapid Malcode*, pp. 24–33. Washington DC, USA.

Livshits, B. and Cui, W. (2008) Spectator: Detection and containment of javascript worms. In *Proceedings of the USENIX Annual Technical Conference*, pp. 335–348. Massachusetts, USA.

Mathworks Inc. (2010) Matlab - the language of technical computing. `http://www.mathworks.com/products/matlab/`.

Nagaraja, S., Mittal, P., Hong, C.-Y., Caesar, M. and Borisov, N. (2010) Botgrep: Finding bots with structured graph analysis. In *Proceedings of the 19th USENIX Security Symposium*, pp. 95–110. California, USA.

Newsome, J., Karp, B. and Song, D. (2005) Polygraph: Automatically generating signatures for polymorphic worms. In *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 226–241. California, USA.

Opdyke, J. D. (2009) A unified approach to algorithms generating unrestricted and restricted integer compositions and integer partitions. *Journal of Mathematical Modelling and Algorithms* **9**(1), 53–97.

Porras, P., Saidi, H. and Yegneswaran, V. (2007) A multi-perspective analysis of the storm (peacomm) worm. In *SRI Computer Science Laboratory technical note*. California, USA.

Porras, P., Saidi, H. and Yegneswaran, V. (2009) A foray into conficker's logic and rendezvous points. In *Proceedings of the 2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats*. Massachusetts, USA.

Ramachandran, A., Feamster, N. and Dagon, D. (2007) *Botnet Detection*, chapter Detecting Botnet Membership with DNSBL Counterintelligence, pp. 131–142. Springer US.

Riley, G. F., Sharif, M. I. and Lee, W. (2004) Simulating Internet worms. In *Proceedings of the 12th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pp. 268–274. Vollendam, Netherlands.

Rohloff, K. R. and Basar, T. (2005) Stochastic behavior of random constant scanning worms. In *Proceedings of the 14th International Conference on Computer Communications and Networks*, pp. 339–344. California, USA.

Sellke, S., Shroff, N. B. and Bagchi, S. (2008) Modeling and automated containment of worms. *IEEE Transactions on Dependable and Secure Computing* **5**(2), 71–86.

Staniford, S., Paxson, V. and Weaver, N. (2002) How to 0wn the Internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium*, pp. 149–165. California, USA.

Stover, S., Dittrich, D., Hernandez, J. and Dietrich, S. (2007) Analysis of the storm and nugache trojans: P2P is here. *;LOGIN* **32**(6).

Thompson, K. H. (1962) Estimation of the proportion of vectors in a natural population of insects. *Biometrics* **18**(4), 568–578.

Walter, S. D., Hildreth, S. W. and Beaty, B. J. (1980) Estimation of infection rates in populations of organisms using pools of variable size. *American Journal of Epidemiology* **112**(1), 124–128.

Wang, P., Sparks, S. and Zou, C. C. (2010) An advanced hybrid peer-to-peer botnet. *IEEE Transactions on Dependable and Secure Computing* **7**(2), 113–127.

White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C. and Joglekar, A. (2002) An integrated experimental environment for distributed systems and networks. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, pp. 255–270. USENIX Association, Boston, MA.

Zou, C. C., Towsley, D., Gong, W. and Cai, S. (2005) Routing worm: a fast, selective attack worm based on IP address information. In *Proceedings of the 19th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation*. Monterey, CA, USA.