

# Hit-List Worm Detection and Bot Identification in Large Networks Using Protocol Graphs

M. Patrick Collins<sup>1</sup> and Michael K. Reiter<sup>2</sup>

<sup>1</sup> CERT/Network Situational Awareness, Software Engineering Institute,  
Carnegie Mellon University; [mcollins@cert.org](mailto:mcollins@cert.org)

<sup>2</sup> Department of Computer Science, University of North Carolina  
at Chapel Hill; [reiter@cs.unc.edu](mailto:reiter@cs.unc.edu)

**Abstract.** We present a novel method for detecting hit-list worms using *protocol graphs*. In a protocol graph, a vertex represents a single IP address, and an edge represents communications between those addresses using a specific protocol (e.g., HTTP). We show that the protocol graphs of four diverse and representative protocols (HTTP, FTP, SMTP, and Oracle), as constructed from monitoring for fixed durations on a large intercontinental network, exhibit stable graph sizes and largest connected component sizes. Moreover, we demonstrate that worm propagations, even of a sophisticated hit-list variety in which the attacker has advance knowledge of his targets and always connects successfully, perturb these properties. We demonstrate that these properties can be monitored very efficiently even in very large networks, giving rise to a viable and novel approach for worm detection. We also demonstrate extensions by which the attacking hosts (bots) can be identified with high accuracy.

**Key words:** Graph analysis, Anomaly detection, Large networks

## 1 Introduction

Large numbers of Internet worms have prompted researchers to develop a variety of anomaly-based approaches to detect these attacks. Examples include monitoring the number of failed connection attempts by a host (e.g., [5, 16, 27]), or the connection rate of a host to new targets (e.g., [24, 19, 17]). These systems are designed to detect abnormally frequent connections and often rely on evidence of connection failure, such as half-open TCP connections. To avoid detection by these systems, an attacker can use a *hit list* [20] generated previous to the attack or generated by another party [3]. An attacker using an accurate hit list contacts only targets known to be running an accessible server, and therefore will not trigger an alarm predicated on connection failure. By constraining the number of attack connections initiated by each attacker-controlled *bot*, the attacker could compromise targets while evading detection by most (if not all) techniques that monitor the behavior of individual hosts or rely on connection failures.

In this paper, we propose a new detection method, based on monitoring *protocol graphs*. A protocol graph is a representation of a traffic log for a single protocol. In this graph, the vertices represent the IP addresses used as clients

or servers for a particular protocol (e.g., FTP), and the edges represent communication between those addresses. We expect that a protocol graph will have properties that derive from its underlying protocol’s design and use. For example, we expect that since Oracle communications require password authentication and HTTP interactions do not, a protocol graph representing Oracle will have more connected components than a protocol graph representing HTTP, though the HTTP graph’s connected components will likely be larger.

Our detection approach focuses on two graph properties: the number of vertices comprising the graph (“graph size”) and the number of vertices in the largest connected component of the graph (“largest component size”) for traffic logs collected in a fixed duration. We hypothesize that while an attacker may have a hit list identifying servers within a network, he will not have accurate information about the activity or audience for those servers. As a consequence, a hit-list attack will either artificially inflate the number of vertices in a protocol graph, or it will connect disjoint components, resulting in a greater than expected largest component size.

To test this, we examine protocol graphs generated from traffic of several common protocols as observed in a large (larger than a /8) network. Specifically, we examine HTTP, SMTP, Oracle and FTP. Using this data, we confirm that protocol graphs for these protocols have predictable graph and largest component sizes. We then inject synthetic hit-list attacks into the network, launched from one or more attacker-controlled bots, to determine if these attacks detectably modify either graph size or largest component size of the observed protocol graphs. The results of our study indicate that monitoring graph size and particularly largest component size is an effective means of hit-list worm detection for a wide range of attack parameters and protocols. For example, if tuned to yield one false alarm per day, our techniques reliably detect aggressive hit-list attacks and detect even moderate hit-list attacks with regularity, whether from one or many attacker-controlled bots.

Once an alarm is raised, an important component of diagnosis is determining which of the vertices in the graph represent bots. We show how to use protocol graphs to achieve this by measuring the *number* of connected components resulting from the removal of high-degree vertices in the graph. We demonstrate through extensions to our analysis that we can identify bots with a high degree of accuracy for FTP, SMTP and HTTP, and with somewhat less (though still useful) accuracy for Oracle. We also show that our bot identification accuracy exceeds what can be achieved by examining vertex degree alone.

While there are many conceivable measures of a protocol graph that might be useful for detecting worms, any such measure must be efficient to monitor if detection is to occur in near-real-time. The graph size and largest component size are very advantageous in this respect, in that they admit very efficient computation via well-known *union-find* algorithms (see [7]). A union-find algorithm implements a collection of disjoint sets of elements supporting two operations: two sets in the collection can be merged (**union**), and the set containing a particular element can be located (**find**). In our application, the elements of sets are

IP addresses, and the sets are the connected components of the protocol graph. As such, when a new communication record is observed, the set containing each endpoint is located (two `find` operations) and, if these two sets are distinct, they can be merged (a `union` operation). Using well-known techniques, communication records can be processed in amortized time that is effectively a small constant per communication record, and in space proportional to the number of IP addresses observed. By comparison, detection approaches that track connection rates to new targets (e.g., [24, 17]) require space proportional to the number of unique connections observed, which can far exceed the number of unique IP addresses observed. While our attacker identification that is performed following an alarm incurs costs similar to these prior techniques, we emphasize that it can be proceed simultaneously with reactive defenses being deployed and so need not be as time-critical as detection itself.

To summarize, the contributions of our paper include (i) defining protocol graphs and detailing their use as a hit-list attack detection technique; (ii) demonstrating through trace-driven analysis on a large network that this technique is effective for detecting hit-list attacks; (iii) extending protocol graph analysis to infer the locations of bots when hit-list worms are detected; and (iv) describing efficient algorithms by which worm detection and bot identification can be performed, in particular with detection being even more efficient than techniques that focus on localized behavior of hosts.

Our paper proceeds as follows. Section 2 summarizes previous relevant work. Section 3 describes protocol graphs and the data we use in our analysis. Section 4 examines the size of graphs and their largest components under normal circumstances, and introduces our anomaly detection technique. In Section 5, we test our technique through simulated hit-list attacks. We extend our approach to identify attackers in Section 6. Section 7 addresses implementation issues. Section 8 summarizes our results and discusses ongoing and future research.

## 2 Previous Work

Several intrusion-detection and protocol-identification systems have used graph-based communication models. Numerous visualization tools (e.g., [12, 26, 25]) present various attributes of communication graphs for visual inspection. Staniford et al.’s GrIDS system [21] generates graphs describing communications between IP addresses or more abstract entities within a network, such as the computers comprising a department. A more recent implementation of this approach by Ellis et al. [6] has been used for worm detection. Karagiannis et al. [9] develop a graphical traffic profiling system called BLINC for identifying applications from their traffic. Stolfo et al.’s [22] Email Mining Toolkit develops graphical representations of email communications and uses them to detect email viruses and worms.

In all of these cases, the systems detect (or present data to a human to detect) phenomena of interest based primarily on localized (e.g., per-vertex or vertex neighborhood) properties of the graph. GrIDS generates rules describing how

internal departments or organizations communicate, and can develop threshold rules (e.g., “trigger an alarm if the vertex has degree more than 20”). Ellis’ approach uses combinations of *link predicates* to identify a host’s behavior. Karagiannis’ approach expresses these same communications using subgraph models called *graphlets*. Stolfo et al.’s approach identifies *cliques* per user, to whom the user has been observed sending the same email, and flags emails that span multiple cliques as potentially containing a virus or worm. In comparison to these efforts, our work focuses on aggregate graph behavior (graph size and largest component size) as opposed to localized properties of the graph or individual vertices. Moreover, some of these approaches utilize more protocol semantics (e.g., the event of sending an email to multiple users [22], or the expected communication patterns of an application [9]) that we do not consider here in the interest of both generality and efficiency.

Several empirical studies have attempted to map out the structure of application networks. Such studies of which we are aware have been conducted by actively crawling the application network in a depth- or breadth-first manner, starting from some seed set of known participants. For example, Broder et al. [4] studied web interconnectivity by characterizing the links between pages. Ripeanu et al. [14] and Saroiu et al. [15] similarly conducted such studies of Gnutella and BitTorrent, respectively. Pouwelse et al. [13] use a similar probe and crawl approach to identify BitTorrent networks over an 8-month period. Our work differs from these in that our techniques are purely passive and are assembled (and evaluated) for the purpose of worm detection.

Our protocol graphs are more closely related to the *call graphs* studied by Aiello et al. [2] in the context of the AT&T voice network. In a call graph, each vertex represents a phone number and each (directed) edge denotes a call placed from one vertex to another. Aiello et al. observe that the size of the largest connected component of observed call graphs is  $\Theta(|V|)$ , where  $V$  denotes the vertices of the graph. These call graphs are similar to our protocol graphs, the primary differences being that call graphs are directed (the protocol graphs we study are undirected) and that they are used to characterize a different domain (telephony, versus data networks here). However, Aiello et al. studied call graphs to understand their basic structure, but not with attention to worm detection (and in fact we are unaware of worms in that domain).

### 3 Preliminaries

In this section, we investigate the construction and composition of protocol graphs. Protocol graphs are generated from traffic logs; our analyses use CISCO Netflow, but graphs can also be constructed using `tcpdump` data or server logs.

This section is structured as follows. Section 3.1 describes the construction of protocol graphs and our notation for describing them and their properties. Section 3.2 describes our source data.

### 3.1 Protocol Graphs

We consider a log file (set)  $A = \{\lambda_1, \dots, \lambda_n\}$  of traffic records. Each record  $\lambda$  has fields for IP addresses, namely source address  $\lambda.\text{sip}$  and destination address  $\lambda.\text{dip}$ . In addition,  $\lambda.\text{server}$  denotes the address of the server in the protocol interaction ( $\lambda.\text{server} \in \{\lambda.\text{sip}, \lambda.\text{dip}\}$ ), though we emphasize that we require  $\lambda.\text{server}$  only for evaluation purposes; it is not used in our detection or attacker identification mechanisms.

Given  $A$ , we define an undirected graph  $G(A) = \langle V(A), E(A) \rangle$ , where

$$V(A) = \bigcup_{\lambda \in A} \{\lambda.\text{sip}, \lambda.\text{dip}\} \quad E(A) = \bigcup_{\lambda \in A} \{(\lambda.\text{sip}, \lambda.\text{dip})\}$$

The largest connected component of a graph  $G(A)$  is denoted  $C(A) \subseteq V(A)$ . Note that by construction,  $G(A)$  has no connected component of size one (i.e., an isolated vertex); all components are of size two or greater.<sup>3</sup>

We denote by  $A_\pi$  a log file that is recorded during the interval  $\pi \subseteq [00:00\text{GMT}, 23:59\text{GMT}]$  on some specified date. We define  $\mathcal{V}_\Pi^{\text{dur}}$  and  $\mathcal{C}_\Pi^{\text{dur}}$  to be random variables of which  $|V(A_\pi)|$  and  $|C(A_\pi)|$ , for logs  $A_\pi$  collected in  $\text{dur}$ -length time intervals  $\pi \subseteq \Pi$ , are observations. For example, in the following sections we will focus on  $\Pi = [00:00\text{GMT}, 11:59\text{GMT}]$  (denoted  $\text{am}$ ) and  $\Pi = [12:00\text{GMT}, 23:59\text{GMT}]$  (denoted  $\text{pm}$ ), and take  $|V(A_\pi)|$  and  $|C(A_\pi)|$  with  $\pi \subseteq \text{am}$  of length 60 seconds (s) as an observation of  $\mathcal{V}_{\text{am}}^{60\text{s}}$  and  $\mathcal{C}_{\text{am}}^{60\text{s}}$ , respectively. We denote the mean and standard deviation of  $\mathcal{V}_\Pi^{\text{dur}}$  by  $\mu(\mathcal{V}_\Pi^{\text{dur}})$  and  $\sigma(\mathcal{V}_\Pi^{\text{dur}})$ , respectively, and similarly for  $\mathcal{C}_\Pi^{\text{dur}}$ .

### 3.2 Data Set

The source data for these analyses are CISCO Netflow traffic summaries collected on a large (larger than a /8) ISP network. We use collectors at the border of the network’s autonomous intranets in order to record the internal and cross border network activity. Therefore, all protocol participants that communicate between intranets or with the Internet are observed. Netflow reports flow logs, where a flow is a sequence of packets with the same addressing information that are closely related in time. Flow data is a compact summary of network traffic and therefore useful for maintaining records of traffic across large networks.

Flow data does not include payload information, and as a result we identify protocol traffic by using port numbers. Given a flow record, we convert it to a log record  $\lambda$  of the type we need by setting  $\lambda.\text{server}$  to the IP address that has the corresponding service port; e.g., in a flow involving ports 80 and 3946, the protocol is assumed to be HTTP and the server is the IP address using port 80. Protocol graphs constructed using log files with payload could look for banners within the payload to identify services.

The protocols used for analysis are listed below.

<sup>3</sup> It is possible for various logging mechanisms, under specific circumstances, to record a flow from a host to itself. We eliminate those records for this work.

- HTTP: The HTTP dataset consists of traffic where the source or destination port is port 80 and the other port is ephemeral ( $\geq 1024$ ). HTTP is the most active protocol on the monitored network, comprising approximately 50% of the total number of bytes crossing the network during the workday.
- SMTP: SMTP consists of TCP traffic where the source or destination port is port 25 and the other port is ephemeral. After HTTP, SMTP is the most active protocol on the monitored network, comprising approximately 30% of the total number of bytes.
- Oracle: The Oracle dataset consists of traffic where one port is 1521 and the other port is ephemeral. While Oracle traffic is a fraction of HTTP and SMTP traffic, it is a business-critical application. More importantly, Oracle connections are password-protected and we expect that as a consequence any single user will have access to a limited number of Oracle servers.
- FTP: The FTP dataset consists of records where one port is either 20 or 21, and the other port is ephemeral. While FTP provides password-based authentication, public FTP servers are still available.

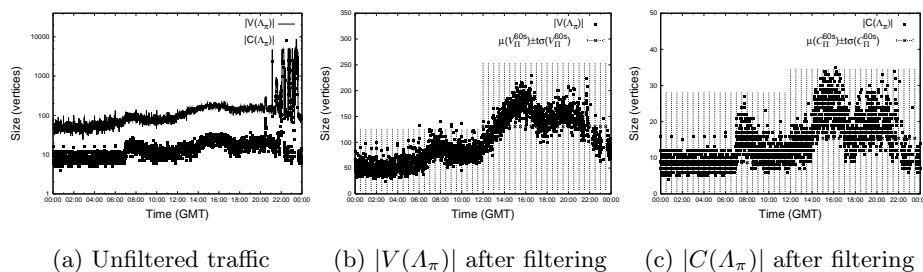
We study these protocols due to their diversity in patterns of activity; e.g., we expect an individual web client to contact multiple web servers, but we expect an individual Oracle client to contact far fewer Oracle servers. That said, this list does not include all protocols that we would like to analyze. A notable omission is peer-to-peer file sharing protocols; we omit these since the monitored network blocks all ports commonly associated with peer-to-peer file-sharing applications (BitTorrent, eDonkey, Kazaa, etc.).

## 4 Building a Hit-List Worm Detector

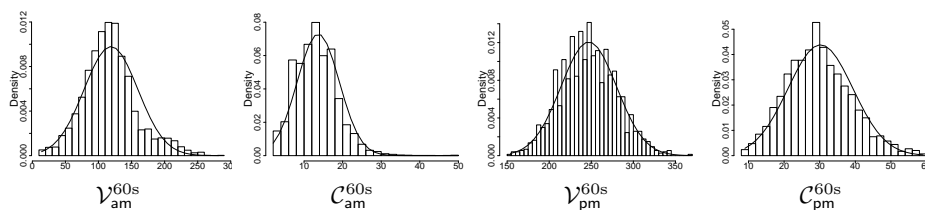
In this section we describe the general behavior of protocol graphs over time, and show that the distributions of  $\mathcal{V}_H^{\text{dur}}$  and  $\mathcal{C}_H^{\text{dur}}$  can be satisfactorily modeled as normal for appropriate choices of  $H$  and  $\text{dur}$  (Section 4.1). The parameters of these distributions change as a function of the protocol (HTTP, SMTP, FTP, Oracle), the interval in which logging occurs ( $H$ ), and the duration of log collection ( $\text{dur}$ ). Nevertheless, in all cases the graph and largest component sizes are normally distributed, which enables us to postulate a detection mechanism for hit-list worms and estimate the false alarm rate for any detection threshold (Section 4.2).

### 4.1 Graph Behavior Over Time

Figure 1 is a plot of the observed values of  $|V(A_\pi)|$  and  $|C(A_\pi)|$  for Oracle traffic on the monitored network for Monday, March 5th, 2007. Each logging interval  $\pi$  begins at a time indicated on the  $x$ -axis, and continues for  $\text{dur} = 60\text{s}$ . Traffic between servers internal to the monitored network and their clients (internal or external to the monitored network) was recorded. Plots including external servers show the same business-cycle dependencies and stability. However, we



**Fig. 1.** Oracle traffic on March 5, 2007; start time of  $\pi$  is on  $x$ -axis;  $\text{dur} = 60\text{s}$



**Fig. 2.** Distributions for Oracle over March 12–16, 2007, fitted to normal distributions.

ignore external servers because the vantage point of our monitored network will not allow us to see an external attack on an external server.

Figure 1(a) is plotted logarithmically due to the anomalous activity visible after 18:00GMT. At this time, multiple bots scanned the monitored network for Oracle servers. These types of disruptive events are common to all the training data; we identify and eliminate these scans using Jung et al.’s sequential hypothesis testing method [8]. In this method, scanners are identified when they attempt to connect to servers that are not present within the targeted network. This method will not succeed against hit-list attackers, as a hit-list attacker will only communicate with servers that are present on the network. Figure 1(b)–(c) is a plot of the same activity after the scan events are removed: Figure 1(b) plots  $|V(\mathcal{A}_\pi)|$ , while Figure 1(c) plots  $|C(\mathcal{A}_\pi)|$ .

Once scans are removed from the traffic logs, the distribution of traffic can be satisfactorily modeled with normal distributions. More precisely, we divide the day into two intervals, namely  $\text{am} = [00:00\text{GMT}, 11:59\text{GMT}]$  and  $\text{pm} = [12:00\text{GMT}, 23:59\text{GMT}]$ . For each protocol we consider, we define random variables  $\mathcal{V}_{\text{am}}^{60\text{s}}$  and  $\mathcal{V}_{\text{pm}}^{60\text{s}}$ , of which the points on the left and right halves of Figure 1(b) are observations for Oracle, respectively. Similarly, we define random variables  $\mathcal{C}_{\text{am}}^{60\text{s}}$  and  $\mathcal{C}_{\text{pm}}^{60\text{s}}$ , of which the points on the left and right halves of Figure 1(c) are observations, respectively. By taking such observations from all of March 12–16, 2007 for each of  $\mathcal{V}_{\text{am}}^{60\text{s}}$ ,  $\mathcal{V}_{\text{pm}}^{60\text{s}}$ ,  $\mathcal{C}_{\text{am}}^{60\text{s}}$  and  $\mathcal{C}_{\text{pm}}^{60\text{s}}$ , we fit a normal distribution to

each effectively; see Figure 2.<sup>4</sup> On the left half of Figure 1(b), we plot  $\mu(\mathcal{V}_{\text{am}}^{60\text{s}})$  as a horizontal line and  $t \sigma(\mathcal{V}_{\text{am}}^{60\text{s}})$  as error bars with  $t = 3.5$ . We do similarly with  $\mu(\mathcal{V}_{\text{pm}}^{60\text{s}})$  and  $t \sigma(\mathcal{V}_{\text{pm}}^{60\text{s}})$  on the right half, and with  $\mu(\mathcal{C}_{\text{am}}^{60\text{s}})$  and  $t \sigma(\mathcal{C}_{\text{am}}^{60\text{s}})$  and  $\mu(\mathcal{C}_{\text{pm}}^{60\text{s}})$  and  $t \sigma(\mathcal{C}_{\text{pm}}^{60\text{s}})$  on the left and right halves of Figure 1(c), respectively. The choice of  $t = 3.5$  will be justified below.

In exactly the same way, we additionally fit normal distributions to  $\mathcal{V}_{\text{am}}^{30\text{s}}$ ,  $\mathcal{C}_{\text{am}}^{30\text{s}}$ ,  $\mathcal{V}_{\text{pm}}^{30\text{s}}$ , and  $\mathcal{C}_{\text{pm}}^{30\text{s}}$  for each protocol, with equally good results. And, of course, we could have selected finer-granularity intervals than half-days (am and pm), resulting in more precise means and standard deviations on, e.g., an hourly basis. Indeed, the tails on the distributions of  $\mathcal{V}_{\text{am}}^{60\text{s}}$  and  $\mathcal{C}_{\text{am}}^{60\text{s}}$  in Figure 2 are a result of the coarse granularity of our chosen intervals, owing to the increase in activity at 07:00GMT (see Figure 1(b)). We elect to not refine our am and pm intervals here, however, for presentational convenience.

## 4.2 Detection and the False Alarm Rate

Our detection system is a simple hypothesis testing system; the null hypothesis is that an observed log file  $A$  does not include a worm propagation. Recall from Section 4.1 that for a fixed interval  $\Pi \in \{\text{am}, \text{pm}\}$ , graph size  $\mathcal{V}_{\Pi}^{\text{dur}}$  and largest component size  $\mathcal{C}_{\Pi}^{\text{dur}}$  normally distributed with mean and standard deviation  $\mu(\mathcal{V}_{\Pi}^{\text{dur}})$  and  $\sigma(\mathcal{C}_{\Pi}^{\text{dur}})$ , respectively. As such, for a dur-length period  $\pi \subseteq \Pi$ , we raise an alarm for a protocol graph  $G(A_{\pi}) = \langle V(A_{\pi}), E(A_{\pi}) \rangle$  if either of the following conditions holds:

$$|V(A_{\pi})| > \mu(\mathcal{V}_{\Pi}^{\text{dur}}) + t \sigma(\mathcal{V}_{\Pi}^{\text{dur}}) \quad (1)$$

$$|C(A_{\pi})| > \mu(\mathcal{C}_{\Pi}^{\text{dur}}) + t \sigma(\mathcal{C}_{\Pi}^{\text{dur}}) \quad (2)$$

Recall that for a normally distributed random variable  $\mathcal{X}$  with mean  $\mu(\mathcal{X})$  and standard deviation  $\sigma(\mathcal{X})$ ,

$$\Pr[\mathcal{X} \leq x] = \frac{1}{2} \left[ 1 + \operatorname{erf} \left( \frac{x - \mu(\mathcal{X})}{\sigma(\mathcal{X})\sqrt{2}} \right) \right]$$

where  $\operatorname{erf}(\cdot)$  is the “error function” [10]. This enables us to compute the contribution of condition (1) to the false rejection (alarm) rate  $\text{frr}$  for a given threshold  $t$  as  $1 - \Pr[\mathcal{V}_{\Pi}^{\text{dur}} \leq \mu(\mathcal{V}_{\Pi}^{\text{dur}}) + t \sigma(\mathcal{V}_{\Pi}^{\text{dur}})]$ , and similarly for the contribution of condition (2) to  $\text{frr}$ . Conversely, since  $\operatorname{erf}^{-1}(\cdot)$  exists, given a desired  $\text{frr}$  we can compute a threshold  $t$  so that our  $\text{frr}$  is not exceeded:

$$t = \sqrt{2} \operatorname{erf}^{-1} \left( \frac{1}{2} - \frac{\text{frr}}{2} \right) \quad (3)$$

Note that the use of  $\frac{\text{frr}}{2}$  in (3) ensures that each of conditions (1) and (2) contribute at most half of the target  $\text{frr}$  and consequently that both conditions combined will yield at most the target  $\text{frr}$ .

<sup>4</sup> For all protocols, the observed Shapiro-Wilk statistic is in excess of 0.94.

Finally, recall that each  $A_\pi$  represents one  $\text{dur}$ -length time period  $\pi$ , and  $\text{frr}$  is expressed as a fraction of the log files, or equivalently,  $\text{dur}$ -length time intervals, in which a false alarm occurs. We can obviously extrapolate this  $\text{frr}$  to see its implications for false alarms over longer periods of time. For the remainder of this paper, we will take as our goal a false alarm frequency of one per day (with  $\text{dur} = 60\text{s}$ ), yielding a threshold of  $t = 3.5$ . This threshold is chosen simply as a representative value for analysis, and can be adjusted to achieve other false alarm frequencies.

This estimate depends on accurate calculations for  $\mu(\mathcal{V}_H^{\text{dur}})$ ,  $\mu(\mathcal{C}_H^{\text{dur}})$ ,  $\sigma(\mathcal{V}_H^{\text{dur}})$ , and  $\sigma(\mathcal{C}_H^{\text{dur}})$  for the time interval  $H$  in which the monitoring occurs. In the remainder of this paper, we will compute these values based on data collected on March 12–16, 2007.

## 5 Protocol Graph Change During Attack

We showed in Section 4 that, for the protocols examined,  $\mathcal{C}_{\text{am}}^{\text{dur}}$ ,  $\mathcal{V}_{\text{am}}^{\text{dur}}$ ,  $\mathcal{C}_{\text{pm}}^{\text{dur}}$  and  $\mathcal{V}_{\text{pm}}^{\text{dur}}$  are normally distributed (Section 4.1), leading to a method for computing the false alarm rate for any given detection threshold (Section 4.2). In this section, we test the effectiveness of this detection mechanism against simulated hit-list attacks. Section 5.1 describes the model of attack used. Section 5.2 describes the experiment and our evaluation criteria. The detection results of our simulations are discussed in Section 5.3.

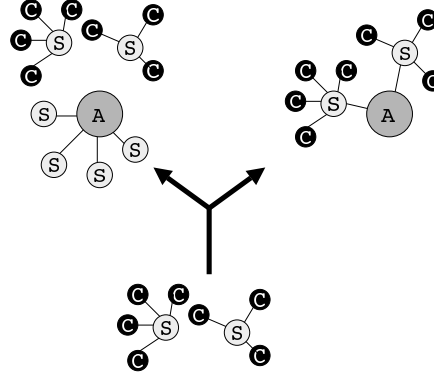
### 5.1 Attack and Defense Model

We simulate hit-list attacks, as described by Staniford et al. [20]. A *hit list* is a list of target servers identified before the actual attack. An apparent example of a hit-list worm is the Witty worm: reports by Shannon and Moore [18] hypothesized that Witty initially spread via a hit list. Further analyses by Kumar et al. [11] identified Witty’s “patient zero” and demonstrated that this host behaved in a distinctly different fashion from subsequently-infected Witty hosts, lending support to the theory that patient zero used a hit list to infect targets.

We hypothesize that an attacker who has a hit list for a targeted network will be detectable by examining  $|C(A_\pi)|$  and  $|V(A_\pi)|$  where  $A_\pi$  is a log file recorded during a time interval  $\pi$  in which a hit-list worm propagated. We assume that the attacker has the hit list but no knowledge of the targeted servers’ current activity or audience. If this is the case, the attacker contacting his hit list will alter the observed protocol graph through *graph inflation* or *component inflation*.

Figure 3 shows how these attacks impact protocol graphs. Graph inflation occurs when an attacker communicates with servers that are not active during the observation period  $\pi$ . When this occurs, the attacker artificially inflates the number of vertices in the graph, resulting in a value of  $|V(A_\pi)|$  that is detectably large. The vertices of a protocol graph include both clients and servers, while the attacker’s hit list will be composed exclusively of servers. As a result, we expect that graph inflation will require communicating with many of the hit-list elements (roughly  $t \sigma(\mathcal{V}_H^{\text{dur}})$  for  $\text{dur}$ -length  $\pi \subseteq H$ ) to trigger condition (1).

Component inflation occurs when the attacker communicates with servers already present in  $\Lambda_\pi$  during the observation period  $\pi$ . When this occurs, the attacker might merge components in the graph, and  $|C(\Lambda_\pi)|$  will then be detectably large. In comparison to graph inflation, component inflation can happen very rapidly; e.g., it may trigger condition (2) if an attacker communicates with only two servers. However, if the graph already has a small number of components (as is the case with SMTP), or the attacker uses multiple bots to attack, then the attack may not be noticed.



**Fig. 3.** Illustration of attacks, where “C”, “S” and “A” denote a client, server and attacker-controlled bot, respectively. Attack on left affects total graph size ( $|V(\Lambda_\pi)|$ ), and so depicts graph inflation. Attack on right affects largest component size ( $|C(\Lambda_\pi)|$ ), and so depicts component inflation.

## 5.2 Experiment Construction

The training period for our experiments was March 12–16, 2007. We considered two different values for  $\text{dur}$ , namely 30s and 60s. Table 1 contains the means and standard deviations for  $\mathcal{V}_{\text{am}}^{\text{dur}}$ ,  $\mathcal{C}_{\text{am}}^{\text{dur}}$ ,  $\mathcal{V}_{\text{pm}}^{\text{dur}}$  and  $\mathcal{C}_{\text{pm}}^{\text{dur}}$  for the training period and for each choice of  $\text{dur}$ , which are needed to evaluate conditions (1) and (2). As shown in Table 1, the largest components for HTTP and SMTP were close to the total sizes of the protocol graphs on average.

r.v.	HTTP		SMTP		Oracle		FTP	
	$\mu$	$\sigma$	$\mu$	$\sigma$	$\mu$	$\sigma$	$\mu$	$\sigma$
$\mathcal{V}_{\text{am}}^{30\text{s}}$	10263	878	2653	357	65.3	18.7	291.9	57.0
$\mathcal{C}_{\text{am}}^{30\text{s}}$	9502	851	2100	367	17.52	4.00	65.30	8.10
$\mathcal{V}_{\text{pm}}^{30\text{s}}$	16460	2540	3859	336	128.7	32.4	359.8	67.1
$\mathcal{C}_{\text{pm}}^{30\text{s}}$	15420	2420	3454	570	30.60	6.28	80.02	8.23
$\mathcal{V}_{\text{am}}^{60\text{s}}$	14760	1210	4520	634	111.8	28.1	467.4	76.9
$\mathcal{C}_{\text{am}}^{60\text{s}}$	13940	1180	4069	650	12.92	4.24	37.3	11.3
$\mathcal{V}_{\text{pm}}^{60\text{s}}$	23280	3480	6540	935	240.3	31.7	555.5	94.8
$\mathcal{C}_{\text{pm}}^{60\text{s}}$	22140	3320	6200	937	28.84	8.44	45.9	12.2

**Table 1.** Means and standard deviations (to three significant digits on standard deviation) for  $\mathcal{V}_{\text{am}}^{\text{dur}}$ ,  $\mathcal{C}_{\text{am}}^{\text{dur}}$ ,  $\mathcal{V}_{\text{pm}}^{\text{dur}}$  and  $\mathcal{C}_{\text{pm}}^{\text{dur}}$  for  $\text{dur} \in \{30\text{s}, 60\text{s}\}$  on March 12–16, 2007.

An important point illustrated in Table 1 is that the graph sizes can differ by orders of magnitude depending on the protocol. This demonstrates the primary argument for generating per-protocol graphs: the *standard deviations* in graph size and largest component size for HTTP and SMTP are larger than the mean sizes for Oracle and FTP.

For testing, we model our attack as follows. During a period  $\pi$ , we collect a log  $\text{ctrl}_\pi$  of normal traffic. In parallel, the attacker uses a hit-list set  $\text{HitList}$  to generate its own traffic log  $\text{attk}$ . This log is merged with  $\text{ctrl}_\pi$  to create a new log  $\Lambda_\pi = \text{ctrl}_\pi \cup \text{attk}$ . We then examine conditions (1) and (2) for interval  $I \in \{\text{am}, \text{pm}\}$  such that  $\pi \subseteq I$ ; if either condition is true, then we raise an alarm. In our tests, we select periods  $\pi$  of length  $\text{dur}$  from March 19, 2007, i.e., the next business day after the training period.

To generate the  $\text{HitList}$  sets, we intersect the sets of servers which are observed as active on each of March 12–16, 2007 between 12:00GMT and 13:00GMT. The numbers of servers so observed are shown in Table 2. The attacker attacks the network over a protocol by selecting  $\text{hitListPerc}$  percentage of these servers (for the corresponding protocol) at random to form  $\text{HitList}$ . The attacker (or rather, his bots) then contacts each element of  $\text{HitList}$  to generate the log file  $\text{attk}$ .

More precisely, we allow the attacker to use multiple bots in the attack; let  $\text{bots}$  denote the bots used by the attacker.  $\text{bots}$  do not appear in the log  $\text{ctrl}_\pi$ , so as to decrease chances of triggering condition (2). Each bot  $\text{bot}_i \in \text{bots}$  is assigned a hit list  $\text{HitList}_i$  consisting of a random  $\frac{|\text{HitList}|}{|\text{bots}|}$  fraction of  $\text{HitList}$ . Each bot’s hit list is drawn randomly from  $\text{HitList}$ , but hit lists do not intersect. That is,  $\bigcup_i \text{HitList}_i = \text{HitList}$  and for  $i \neq j$ ,  $\text{HitList}_i \cap \text{HitList}_j = \emptyset$ . By employing hit lists that do not intersect, we again decrease the chances of triggering condition (2).  $\text{attk}$  is generated by creating synthetic attack records from each  $\text{bot}_i$  to all members of  $\text{HitList}_i$ . In our simulations, members of  $\text{HitList}_i$  do not participate in the attack after being contacted by  $\text{bot}_i$ ; i.e., each  $\lambda \in \text{attk}$  is initiated by a member of  $\text{bots}$ . That said, this restriction is largely irrelevant to our results, since neither  $|V(\Lambda_\pi)|$  nor  $|C(\Lambda_\pi)|$  is sensitive to whether a member of  $\text{HitList}_i$  is contacted by another member of  $\text{HitList}_i$  (after it is “infected”) or by  $\text{bot}_i$  itself.

Protocol	Servers
SMTP	2818
HTTP	8145
Oracle	262
FTP	1409

**Table 2.** Count of servers observed between 12:00GMT and 13:00GMT on each of March 12–16, 2007.

### 5.3 True Alarms

Table 3 shows the effectiveness of the detection mechanism as a function of  $\text{dur}$  for different  $\text{hitListPerc}$  values.  $\text{hitListPerc}$  varies between 25% and 100%. The value in each cell is the percentage of attacks that were detected by the system.

Table 3 sheds light on the effectiveness of our approach. The most aggressive worms we considered, namely those that contacted  $\text{hitListPerc} \geq 75\%$  of known

		HTTP				SMTP				Oracle				FTP			
		hitListPerc =				hitListPerc =				hitListPerc =				hitListPerc =			
dur	bots	25	50	75	100	25	50	75	100	25	50	75	100	25	50	75	100
30s	1	73	80	95	100	28	74	100	100	100	100	100	100	100	100	100	100
	3	72	80	95	100	25	50	97	100	33	95	100	100	100	100	100	100
	5	60	80	92	100	23	45	98	100	16	87	99	100	100	100	100	100
60s	1	68	80	100	100	20	50	70	80	100	100	100	100	100	100	100	100
	3	65	68	100	100	10	35	65	70	28	100	100	100	100	100	100	100
	5	65	63	100	100	5	30	60	55	12	100	100	100	100	100	100	100

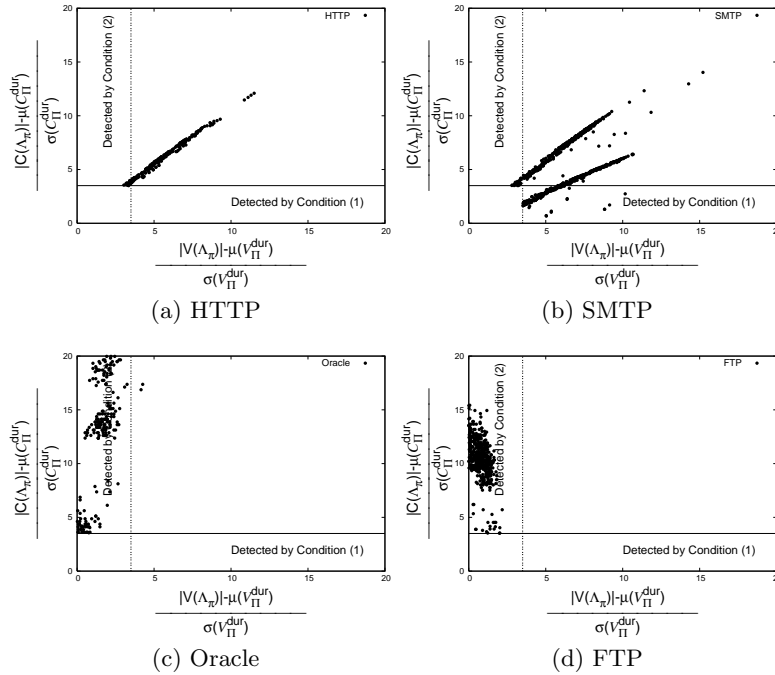
**Table 3.** True alarm percentages for combined detector (conditions (1) and (2))

servers (see Table 2) within  $\text{dur} = 30\text{s}$ , were easily detected: our tests detected these worms more than 90% of the time for all protocols and all numbers of  $|\text{bots}|$ , and at least 95% of the time except in one case.

The table also sheds light on approaches an adversary might take to make his worm more stealthy. First, the adversary might decrease `hitListPerc`. While this does impact detection, our detection capability is still useful: e.g., as `hitListPerc` is decreased to 50% in  $\text{dur} = 30\text{s}$ , the true detection rates drop, but remain 80% or higher for all protocols except SMTP. Second, the adversary might increase  $\text{dur}$ . If the adversary keeps `hitListPerc`  $\geq 75\%$ , then increasing  $\text{dur}$  from 30s to 60s appears to have no detrimental effect on the true alarm rate of the detector for HTTP, Oracle or FTP, and it remains at 60% or higher for SMTP, as well.

Third, the adversary might increase  $|\text{bots}|$ . Note that whereas the previous two attempts to evade detection necessarily slow the worm propagation, increasing  $|\text{bots}|$  while keeping `hitListPerc` and  $\text{dur}$  fixed need not—though it obviously requires the adversary to have compromised more hosts prior to launching his hit-list worm. Intuitively, increasing  $|\text{bots}|$  might decrease the likelihood of detection by our technique by reducing the probability that one  $\text{bot}_i$  will merge components of the graph and thereby trigger condition (2). (Recall that bots’ individual hit lists do not intersect.) However, Table 3 suggests that in many cases this is ineffective unless the adversary simultaneously decreases `hitListPerc`: with `hitListPerc`  $\geq 75\%$ , all true detection rates with  $|\text{bots}| = 5$  remain above 92% with the exception of SMTP (at 60% for  $\text{dur} = 60\text{s}$ ). The effects of increasing  $|\text{bots}|$  may become more pronounced with larger numbers, though if  $|\text{bots}|$  approaches  $t \sigma(\mathcal{V}_H^{\text{dur}})$  then the attacker risks being detected by condition (1) immediately.

Figure 4 compares the effectiveness of conditions (1) and (2) for each of the test protocols. Each plot in this figure is a scatter plot comparing the deviation of  $|C(A_\pi)|$  against the deviation of  $|V(A_\pi)|$  during attacks. Specifically, values on the horizontal axis are  $(|V(A_\pi)| - \mu(\mathcal{V}_H^{\text{dur}})) / \sigma(\mathcal{V}_H^{\text{dur}})$ , and values on the vertical axis are  $(|C(A_\pi)| - \mu(\mathcal{C}_H^{\text{dur}})) / \sigma(\mathcal{C}_H^{\text{dur}})$ , for  $H \supseteq \pi$ . The points on the scatter plot represent the true alarms summarized in Table 3, though for presentational convenience only those true alarms where  $(|V(A_\pi)| - \mu(\mathcal{V}_H^{\text{dur}})) / \sigma(\mathcal{V}_H^{\text{dur}}) \leq 20$



**Fig. 4.** Contributions of conditions (1) and (2) to true alarms in Table 3. For clarity, only true alarms where  $\frac{|V(A_\pi)| - \mu(V_\Pi^{\text{dur}})}{\sigma(V_\Pi^{\text{dur}})} \leq 20$  or  $\frac{|C(A_\pi)| - \mu(C_\Pi^{\text{dur}})}{\sigma(C_\Pi^{\text{dur}})} \leq 20$  are plotted.

or  $(|C(A_\pi)| - \mu(C_\Pi^{\text{dur}}))/\sigma(C_\Pi^{\text{dur}}) \leq 20$  are shown. Each plot has reference lines at  $t = 3.5$  on the horizontal and vertical axes to indicate the trigger point for each detection mechanism. That is, a “•” above the horizontal  $t = 3.5$  line indicates a test in which condition (2) was met, and a “•” to the right of the vertical  $t = 3.5$  line indicates a test in which condition (1) was met.

We would expect that if both conditions were effectively equivalent, then every “•” would be in the upper right “quadrant” of each graph. HTTP (Figure 4(a)) shows this behavior, owing to the fact that in HTTP, the largest component size and graph size are nearly the same in average and in standard deviation; see Table 1. As such, each bot contacts only the largest component with high probability, and so adds to the largest component all nodes that it also adds to the graph. A similar phenomenon occurs with SMTP (Figure 4(b)), though with different scales in **am** and **pm**, yielding the two distinct patterns shown there. However, the other graphs demonstrate different behaviors. Figure 4(c) and (d) shows that the growth of  $|C(A_\pi)|$  is an effective mechanism for detecting disruptions in both Oracle and FTP networks. The only protocol where graph inflation appears to be a more potent indicator than component inflation is SMTP. From this, we conclude that component inflation (condition (2)) is a more potent de-

tector than graph inflation (condition (1)) when the protocol’s graph structure is disjoint, but that each test has a role to play in detecting attacks.

## 6 Bot Identification

Once an attack is detected, individual attackers (bots) are identifiable by how they deform the protocol graph. As discussed in Section 5.1, we expect a bot to impact the graph’s structure by connecting otherwise disjoint components. We therefore expect that removing a bot from a graph  $G(A)$  will separate components and so the number of connected components will increase.

To test this hypothesis, we consider the effect of removing all records  $\lambda$  involving an individual IP address from  $A$ . Specifically, for a log file  $A$  and an IP address  $v$ , define:

$$A^{-v} = \{\lambda \in A : \lambda.\text{sip} \neq v \wedge \lambda.\text{dip} \neq v\}$$

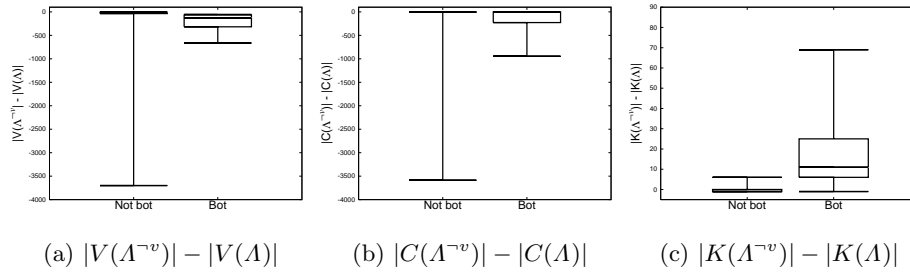
As such,  $G(A)$  differs from  $G(A^{-v})$  in that the latter includes neither  $v$  nor any  $v' \in V(A)$  of degree one that is adjacent only to  $v$  in  $G(A)$ .

In order to detect a bot, we are primarily interested in comparing  $G(A)$  and  $G(A^{-v})$  for vertices  $v$  of high degree in  $G(A)$ , based on the intuition that bots should have high degree. Figure 5 examines the impact of eliminating each of the ten highest-degree vertices  $v$  in  $G(A)$  from each log file  $A$  for FTP discussed in Section 5 that resulted in a true alarm. Specifically:

- Figure 5(a) represents  $|V(A^{-v})| - |V(A)|$ , i.e., the difference in the number of vertices due to eliminating  $v$  and all isolated neighbors, which will be negative;
- Figure 5(b) represents  $|C(A^{-v})| - |C(A)|$ , i.e., the difference in the size of the largest connected component due to eliminating  $v$  and all isolated neighbors, which can be negative or zero; and
- Figure 5(c) represents  $|K(A^{-v})| - |K(A)|$ , i.e., the difference in the number of connected components due to eliminating  $v$  and all isolated neighbors, which can be positive, zero, or  $-1$  if eliminating  $v$  and its isolated neighbors eliminates an entire connected component.

Each boxplot separates the cases in which  $v$  is a bot (right) or is not a bot (left). In each case, five horizontal lines from bottom to top mark the minimum, first quartile, median, third quartile and maximum values, with the lines for the first and third quartiles making a “box” that includes the median line. The five horizontal lines and the box are evident, e.g., in the “bot” boxplot in Figure 5(c). However, because some horizontal lines are on top of one another in other boxplots, the five lines or the box is not evident in all cases.

This figure shows a strong dichotomy between the two graph parameters used for detection (graph size and largest component size) and the number of components. As shown in Figures 5(a) and 5(b), the impact of eliminating bots and the impact of eliminating other vertices largely overlap, for either graph size or



**Fig. 5.** Effects of eliminating high degree vertices  $v$  from FTP attack traffic logs  $A$

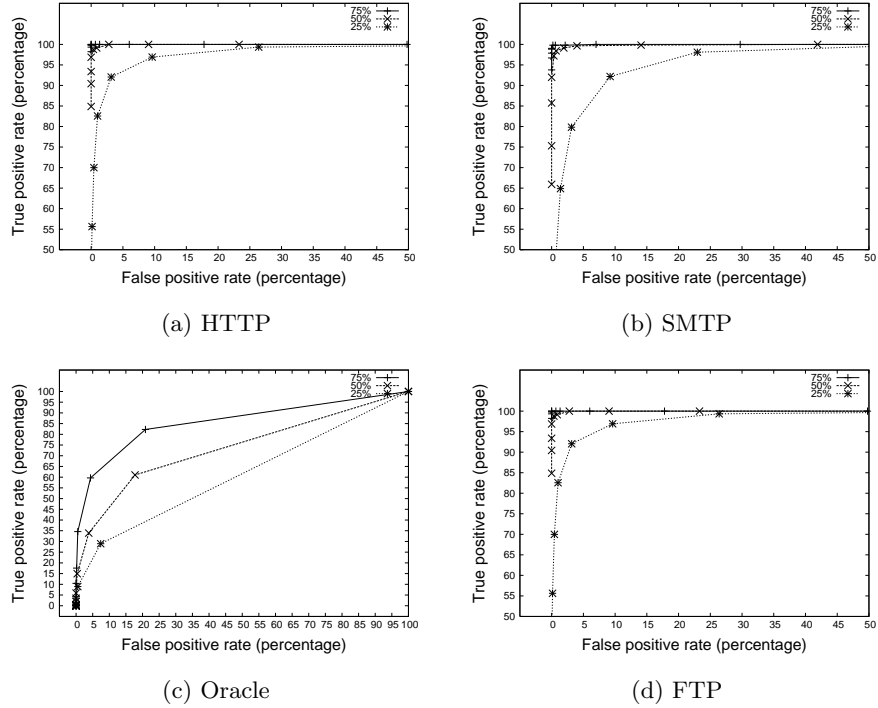
largest component size. In comparison, eliminating bots has a radically different effect on the number of components, as shown in Figure 5(c): when a non-bot vertex is eliminated, the number of components increases a small amount, or sometimes decreases. In contrast, when a bot is eliminated, the number of components *increases* strongly.

Also of note is that the change in the total number of components (Figure 5(c)) is relatively small, and small enough to add little power for attack *detection*. For example, if we were to define a random variable  $\mathcal{K}_{\text{pm}}^{\text{dur}}$  analogous to  $\mathcal{V}_{\text{pm}}^{\text{dur}}$  and  $\mathcal{C}_{\text{pm}}^{\text{dur}}$ , and then formulate a worm detection rule analogous to (1) and (2) for component count—i.e., raise an alarm for log file  $A_\pi$  where  $\pi \in \text{pm}$  had duration  $\text{dur}$ , if  $|K(A_\pi)| > \mu(\mathcal{K}_{\text{pm}}^{\text{dur}}) + t \sigma(\mathcal{K}_{\text{pm}}^{\text{dur}})$ —then roughly 80% of our hit-list attacks within FTP would go undetected by this check. This is because of the large standard deviation of this measure:  $\sigma(\mathcal{K}_{\text{pm}}^{60s}) \approx 12.5$ .

Despite the fact that the number of components does not offer additional power for attack detection, Figure 5(c) suggests that removing a high-degree vertex  $v$  from a graph  $G(A)$  on which an alarm has been raised, and checking the number of connected components that result, can provide an effective test to determine whether  $v$  is a bot. More specifically, we define the following bot identification test:

$$\text{isbot}_{A,\theta}(v) = \begin{cases} 1 & \text{if } |K(A^{-v})| - |K(A)| > \theta \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

We characterize the quality of this test using ROC curves. Each curve in Figure 6 is a plot of true positive (i.e., bot identification) rate versus false positive rate for one of the protocols we consider and for the simulated hit-list worm attacks discussed in Section 5 that yielded a true alarm with  $|\text{bots}| = 5$  (the hardest case in which to find the bots) and  $\text{hitListPerc} \in \{25\%, 50\%, 75\%\}$ . Each point on a curve shows the true and false positive rates for a particular setting of  $\theta$ . More specifically, if  $\text{hidegree} \subseteq V(A)$  is a set of highest-degree vertices in  $G(A)$ , and if  $\text{hidegreebots} \subseteq \text{hidegree}$  denotes the bots in  $\text{hidegree}$ , then any point



**Fig. 6.** Attacker identification accuracy of (4);  $\text{hitListPerc} \in \{25\%, 50\%, 75\%\}$ ,  $|\text{hidegree}| = 10$ ,  $|\text{bots}| = 5$

in Figure 6 is defined by

$$\text{true positive rate} = \frac{\sum_{v \in \text{hidegreebots}} \text{isbot}_{\Lambda, \theta}(v)}{|\text{hidegreebots}|}$$

$$\text{false positive rate} = \frac{\sum_{v \in \text{hidegree} \setminus \text{hidegreebots}} \text{isbot}_{\Lambda, \theta}(v)}{|\text{hidegree} \setminus \text{hidegreebots}|}$$

As Figure 6 shows, a more aggressive worm (i.e., as  $\text{hitListPerc}$  grows) exposes its bots with a greater degree of accuracy in this test, not surprisingly, and the absolute detection accuracy for the most aggressive worms we consider is very good for HTTP, SMTP and FTP. Moreover, while the curves in Figure 6 were calculated with  $|\text{hidegree}| = 10$ , we have found that the accuracy is very robust to increasing  $|\text{hidegree}|$  as high as 100. As such, when identifying bots, it does not appear important to the accuracy of the test that the investigator first accurately estimate the number of bots involved in the attack. We are more thoroughly exploring the sensitivity of (4) to  $|\text{hidegree}|$  in ongoing work, however.

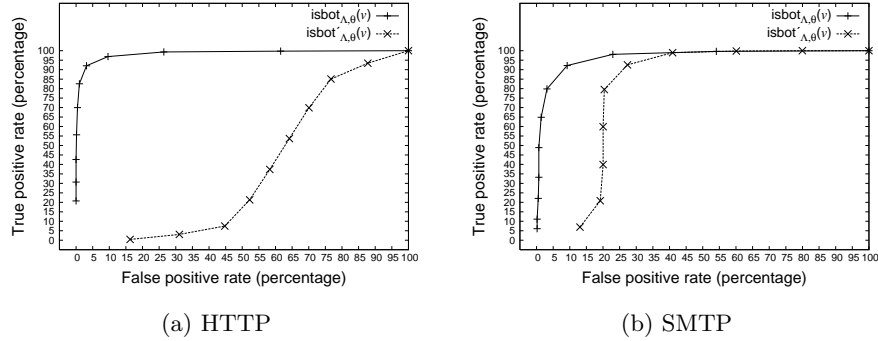


Fig. 7. Accuracy of (4) versus (5); hitListPerc = 25%, |hidegree| = 10, |bots| = 5

Because we evaluate (4) on high-degree vertices in order to find bots, a natural question is whether degree in  $G(\mathcal{A})$  alone could be used to identify bots with similar accuracy, an idea similar to those used by numerous detectors that count the number of destinations to which a host connects (e.g., [24, 17]). To shed light on this question, we consider an alternative bot identification predicate, namely

$$\text{isbot}'_{\mathcal{A},\theta}(v) = \begin{cases} 1 & \text{if } \text{degree}_{\mathcal{A}}(v) > \theta \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

where  $\text{degree}_{\mathcal{A}}(v)$  is the degree of  $v$  in  $G(\mathcal{A})$ , and compare this test to (4) in Figure 7. As this figure shows, using (5) offers much less accurate results in some circumstances, lending support to the notion that our proposal (4) for bot identification is more effective than this alternative.

## 7 Implementation

Any worm detection system must be efficient to keep up with the high pace of flows observed in some protocols. A strength of our detection approach based on conditions (1) and (2) in Section 4 is that it admits very efficient implementation by well-known *union-find* algorithms [7]. Such an algorithm maintains a collection of disjoint sets, and supports three types of operations on that collection: a `makeset` operation creates a new singleton set containing its argument; a `find` operation locates the set containing its argument; and a `union` operation merges the two sets named in its arguments into one set. The size of each set in the collection can be maintained easily because each set in the collection is disjoint: a new set created by `makeset` has size one, and the set resulting from a `union` is of size the sum of the sizes of the merged sets.

The implementation of a worm detection system using a union-find algorithm is straightforward: for each  $\lambda \in \mathcal{A}$ , the sets containing  $\lambda.\text{sip}$  and  $\lambda.\text{dip}$  are

located by `find` operations (or created via `makeset` if the address has not yet been observed in  $A$ ), and if these sets are distinct, they are merged by a `union` operation.  $|C(A)|$  is simply the size of the largest set, and  $|V(A)|$  is the sum of the sizes of the sets.

The efficiency of this implementation derives from the use of classic techniques (see [7]). A famous result by Tarjan (see [23]) shows that with these techniques, a log file  $A$  can be processed in time  $O(|A|\alpha(|V(A)|))$ , where  $\alpha(\cdot)$  is the inverse of Ackermann’s function  $A(\cdot)$ , i.e.,  $\alpha(n) = \arg \min_k : A(k) \geq n$ . Due to the rapid growth of  $A(k)$  as a function of  $k$  (see [1, 23]),  $\alpha(n) \leq 5$  for any practical value of  $|V(A)|$ . So, practically speaking, this algorithm enables the processing of flows with computation only a small constant per flow. Perhaps as importantly, this can be achieved in space  $O(|V(A)|)$ . In contrast, accurately tracking the number of unique destinations to which each vertex connects—a component of several other worm detection systems (e.g., [24, 17])—requires  $\Omega(|E(A)|)$  space, a potentially much greater cost for large networks. Hence our approach is strikingly efficient while also being an effective detection technique.

Once an alarm is raised for a graph  $G(A) = \langle V(A), E(A) \rangle$  due to it violating condition (1) or (2), identifying the bots via the technique of Section 6 requires that we find the high-degree vertices in  $V(A)$ , i.e., the vertices that have the most unique neighbors. To our knowledge, the most efficient method to do this amounts to simply building the graph explicitly and counting each vertex’s neighbors, which does involve additional overhead, namely  $O(|E(A)|)$  space and  $O(|A| \log(|E(A)|))$  time in the best algorithm of which we are aware. However, this additional cost must be incurred only after a detection and so can proceed in parallel with other reactive measures, presumably in a somewhat less time-critical fashion or on a more resource-rich platform than detection itself.

## 8 Conclusion

In this paper, we introduced a novel form of network monitoring technique based on protocol graphs. We demonstrated using logs collected from a very large intercontinental network that the graph and largest component sizes of protocol graphs for representative protocols are stable over time (Section 4.1). We used this observation to postulate tests to detect hit-list worms, and showed how these tests can be tuned to limit false alarms to any desired level (Section 4.2). We also showed that our tests are an effective approach to detecting a range of hit-list attacks (Section 5).

We also examined the problem of identifying the attacker’s bots once a detection occurs (Section 6). We demonstrated that the change in the number of connected components caused by removing a vertex from the graph can be an accurate indicator of whether this vertex represents a bot, and specifically can be more accurate than examining merely vertex degrees.

Finally, we examined algorithms for implementing both hit-list worm detection and bot identification using our techniques (Section 7). We found that hit-list worm detection, in particular, can be implemented using more efficient

algorithms than many other worm detection approaches, using classic union-find algorithms. For networks of the size we have considered here, such efficiencies are not merely of theoretical interest, but can make the difference between what is practical and what is not. Our bot identification algorithms are of similar performance complexity to prior techniques, but need not be executed on the critical path of detection.

Since a protocol graph captures only the traffic of a single protocol, our detector could be circumvented by a worm that propagates within a variety of different protocols. A natural extension of our techniques for detecting such a worm would be to consider graphs that involve multiple protocols at once, though we have not evaluated this possibility and leave this for future work.

### Acknowledgements

We are grateful to Dawn Song for initial discussions on topics related to this paper, and to the anonymous reviewers for comments that helped to improve the paper. This work was supported in part by NSF grant CNS-0433540.

### References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1975.
2. W. Aiello, F. Chung, and L. Lu. A random graph model for massive graphs. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, pages 171–180, 2000.
3. S. Antonatos, P. Akritidis, E. P. Markatos, and K. G. Anagnostakis. Defending against hitlist worms using network address space randomization. In *WORM '05: Proceedings of the 2005 ACM Workshop on Rapid Malcode*, pages 30–40, New York, NY, USA, 2005.
4. A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. In *Proc. of the WWW9 Conference*, pages 309–320, Amsterdam, Holland, 2000.
5. S. Chen and Y. Tang. Slowing down Internet worms. In *Proceedings of the 24th International Conference on Distributed Computing Systems*, pages 312–319, Tokyo, Japan, March 2004.
6. D. Ellis, J. Aiken, A. McLeod, D. Keppler, and P. Amman. Graph-based worm detection on operational enterprise networks. Technical Report MTR-06W0000035, MITRE Corporation, April 2006.
7. Z. Galil and G. F. Italiano. Data structures and algorithms for disjoint set union problems. *ACM Computing Surveys*, 23:319–344, September 1991.
8. J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *Proceedings of the 2004 IEEE Symposium on Security and Privacy*, May 2004.
9. T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: multilevel traffic classification in the dark. In *Proceedings of ACM SIGCOMM '05*, pages 229–240, New York, NY, USA, 2005.
10. E. Kreyszig. *Advanced Engineering Mathematics, 9th Edition*. J. Wiley and Sons, 2005.

11. A. Kumar, V. Paxson, and N. Weaver. Exploiting underlying structure for detailed reconstruction of an Internet scale event. In *Proceedings of the ACM Internet Measurement Conference*, New Orleans, LA, USA, October 2005.
12. K. Lakkaraju, W. Yurcik, and A. Lee. NVisionIP: NetFlow visualizations of system state for security situational awareness. In *Proceedings of the 2004 Workshop on Visualization for Computer Security*, October 2006.
13. J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. A measurement study of the BitTorrent peer-to-peer file-sharing system. Technical Report PDS-2004-007, Delft University of Technology, April 2004.
14. M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing*, 6(1), 2002.
15. S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking 2002*, San Jose, CA, USA, 2002.
16. S. Schechter, J. Jung, and A. Berger. Fast detection of scanning worm infections. In *7th International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 59–81, French Riviera, France, September 2004.
17. V. Sekar, Y. Xie, M. K. Reiter, and H. Zhang. A multi-resolution approach to worm detection and containment. In *Proceedings of the 2006 International Conference on Dependable Systems and Networks*, pages 189–198, June 2006.
18. C. Shannon and D. Moore. The spread of the Witty worm. *IEEE Security and Privacy*, 2(4):46–50, 2004.
19. S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *Proceedings of the ACM/USENIX Symposium on Operating System Design and Implementation*, December 2005.
20. S. Staniford, V. Paxson, and N. Weaver. How to Own the Internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium*, pages 149–167, August 2002.
21. S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS – A graph-based intrusion detection system for large networks. In *Proceedings of the 19th National Information Systems Security Conference*, pages 361–370, 1996.
22. S. J. Stolfo, S. Hershkop, C. Hu, W. Li, O. Nimeskern, and K. Wang. Behavior-based modeling and its application to email analysis. *ACM Transactions on Internet Technology*, 6(2):187–221, May 2006.
23. R. E. Tarjan. *Data Structures in Network Algorithms*, volume 44 of *Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics, 1983.
24. J. Twycross and M. W. Williamson. Implementing and testing a virus throttle. In *Proceedings of the 12th USENIX Security Symposium*, pages 285–294, August 2003.
25. C. Wright, F. Monrose, and G. Masson. Using visual motifs to classify encrypted traffic. In *Proceedings of the 2006 Workshop on Visualization for Computer Security*, November 2006.
26. X. Yin, W. Yurcik, and M. Treaster. VisFlowConnect: NetFlow visualizations of link relationships for security situational awareness. In *Proceedings of the 2004 Workshop on Visualization for Computer Security*, October 2006.
27. C. Zou, L. Gao, W. Gong, and D. Towsley. Monitoring and early warning for Internet worms. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, pages 190–199, New York, NY, USA, 2003.