HoneyStat: Local Worm Detection Using Honeypots

David Dagon, Xinzhou Qin, Guofei Gu, Wenke Lee
Julian Grizzard, John Levine, and Henry Owen
{dagon, xinzhou, guofei, wenke}@cc.gatech.edu,
{grizzard, levine, henry.owen}@ece.gatech.edu

Georgia Institute of Technology

Abstract. Worm detection systems have traditionally used global strategies and focused on scan rates. The noise associated with this approach requires statistical techniques and large data sets (e.g., 2^{20} monitored machines) to avoid false positives. Worm detection techniques for smaller local networks have not been fully explored.

We consider how local networks can provide early detection and compliment global monitoring strategies. We describe HoneyStat, which uses modified honeypots to generate a highly accurate alert stream with low false positive rates. Unlike traditional honeypots, HoneyStat nodes are minimal, script-driven and cover a large IP space.

The HoneyStat nodes generate three classes of alerts: memory alerts (based on buffer overflow detection and process management), disk write alerts (such as writes to registry keys and critical files) and network alerts. Data collection is automated, and once an alert is issued, a time segment of previous traffic to the node is analyzed. A logit analysis determines what previous network activity explains the current honeypot alert. The result can indicate whether an automated or worm attack is present.

We demonstrate HoneyStat's improvements over previous worm detection techniques. First, using trace files from worm attacks on small networks, we demonstrate how it detects zero day worms. Second, we show how it detects multi vector worms that use combinations of ports to attack. Third, the alerts from HoneyStat provide more information than traditional IDS alerts, such as binary signatures, attack vectors, and attack rates. We also use extensive (year long) trace files to show how the logit analysis produces very low false positive rates.

1 Introduction

Worm detection strategies have traditionally relied on artifacts incidental to the worm infection. For example, many researchers measure incoming scan rates (often using darknets) to indirectly detect worm outbreaks, e.g., [ZGGT03]. But since these techniques measure noise as well as attacks, they often use costly algorithms to identify worms. For example, [ZGGT03] suggests using a Kalman filter [Kal60] to detect worm attacks. In [QDG+], this approach was found to work with a large data set but proved inappropriate for smaller networks.

To improve detection time and decrease errors caused by noise, the solution so far has been to increase monitoring efforts, and gather more data. The intuition is that with more data, statistical models do a better job of identifying attacks. Thus, researchers have suggested the creation of global monitoring centers [MSVS03], and collecting information from distributed sensors. Distributed intrusion detection efforts are already yielding interesting results [YBJ04,Par04].

Although the need for global monitoring is obvious, the value this has for local networks is not entirely clear. For example, some local networks might have enough information to conclude a worm is active, based on additional information they are unwilling to share with other monitoring sites. Likewise, since global detection strategies require large amounts of sensor data before detecting worm outbreaks, some local networks might learn about a worm outbreak only after it becomes too late. That is, global detection strategies depend on enough local networks falling victim to a worm, before sufficient worm traffic becomes detectable. Also, we see significant problems in gaining consensus among different networks, which frequently have competing and inconsistent policies regarding privacy, notification, and information sharing.

Without doubt, aggregating information from distributed sensors makes good sense. However, our emphasis is on local networks and requires a complimentary approach. In addition to improving the *quantity* of monitoring data, researchers should work to improve the *quality* of the alert stream. Thus, in [QDG$^+$,GSQ$^+$04], it was recommended that researchers track worm *behavior* instead of just the scanning artifacts.

In this paper, we propose the use of honeypots to improve the accuracy of alerts generated for local intrusion detection systems. To motivate the discussion, we describe in Section 3 the worm infection cycle we observed in honeypots that led to the creation of HoneyStat. Since honeypots usually require labor-intensive management and review, we describe in Section 4 a deployment mechanism used to automate data collection.

HoneyStat nodes collect three types of events: memory, disk write and network events. In Section 4, we describe these in detail, and discuss a way to compare and correlate intrusion events detected by honeypots. Using logistic regression, we analyze previous network traffic to the honeypot to see what network traffic most explains the intrusion events. Intuitively, the logit analysis asks if there is a common set of network inputs that precede honeypot intrusions. Finding a pattern can determine the presence of an automated attack or worm.

To demonstrate HoneyStat's effectiveness, in Section 6, we describe our experience deploying HoneyStat nodes, and a retrospective analysis of network captures. We also use lengthy (year long) network trace files to analyze the false positive rate associated with the algorithm. The false positive rate is found to be low, due to two influences: (a) the use of honeypots, which only produce alerts when there are successful attacks, and (b) the use of user-selected confidence intervals, which let one define a threshold for alerts.

Finally, in Section 7, we analyze whether a local detection strategy with a low false positive rate (like HoneyStat) can make an effective worm detection tool. We consider the advantages this approach has for local networks.

2 Related Work

Honeypots A honeypot is a vulnerable network decoy used for several purposes: (a) distracting attackers, (b) early warnings about new attack techniques, (c) in-depth analysis of an adversary's strategies [Spi03,Sko02]. By design, a honeypot should not receive any network traffic, nor will it run any legitimate production services. This greatly reduces the problem of false positives and false negatives that are an issue with other types of IDS systems.

Traditionally, honeypots have been used to gather intelligence about how human attackers operate [Spi03]. The labor-intensive log review required of traditional honeypots makes them unsuitable for a real-time IDS. In our experience, data capture and log analysis time can require a 1:40 ratio, meaning that a single hour of activity can require a week to fully decipher [LLO$^+$03].

The closest work to our own is [LLO$^+$03], which uses honeypots in an intrusion detection system. We have had great success here at the Georgia Institute of Technology utilizing a Honeynet as an IDS tool, and have identified a large number of compromised systems on campus. The majority of these systems have been compromised by worm type exploits.

Researchers have also considered using virtual honeypots, particularly with honeyd [Pro03]. Initially used to help prevent OS fingerprinting, honeyd is a network daemon that exhibits the TCP/IP stack behavior of different operating systems. It has since been extended to emulate some services (e.g., NetBIOS).

Conceptually, honeyd is a daemon written using libpcap and libdnet. To emulate a service, honeyd requires researchers to write programs that completely copy the service's network behavior. In other words, instead of running an OS in emulation, honeyd uses a network daemon that behaves like a particular service. Assuming one can write enough modules to emulate all aspects of an OS, there are many benefits: (a) an entire subnet can be emulated by a single machine, confusing attackers, and (b) scanning worms experience latency delays, if they scan to local emulated networks.

Recently, honeyd was offered as a way to detect and disable worms [Pro03]. We believe this approach has promise, but must overcome a few significant hurdles before it is used as an early warning IDS. First, it is not clear how a daemon emulating a network service can catch zero day worms. If one knows a worm's attack pattern, it is possible to write modules that will behave like a vulnerable service. But before this is known, catching zero day worms requires emulating even the *presumably unknown* bugs in a network service. Worms that do anything complex (such as downloading a trojan) easily evade honeyd, until a module emulating the bug is created.

Second, honeyd's ability to delay worms does come at a cost. By emulating a large virtual network, a machine essentially invites a worm to attack the local network. Depending on the topology, this may cause the worm to consume large amounts of local bandwidth. In any event, a worm can overcome efforts to distract it with decoy networks simply by adding more threads, or by counting the number of local failed connections to infer the operation of LaBrae-style defenses [Lis01]. We were unable to find solutions to these limitations, and so do not consider virtual networks as a means of providing an improved alert stream. Instead, we used full honeypots.

More closely related to our work, [Kre03] suggested automatic binary signature extraction using honeypots. This work used honeyd, flow reconstruction, and pattern detection to generate IDS rules. The honeycomb approach also has promise, but uses a very simple algorithm (longest common substring) to correlate payloads. This makes it diffi cult to identify polymorphic worms and worms that use multiple attack vectors. Honeycomb was more than adequate for its stated purpose, however: extracting string signatures for automated updates to a fi rewall.

Worm Detection Worm propagation and early detection have been active research topics in the security community. In worm propagation, researchers have proposed an epidemic model to study worm spreading, e.g., [Sta01,ZGT02,CGK03]. In worm early detection, researchers have also proposed different techniques to identify worm activities at an early phase, e.g., Kalman Filter [ZGGT03], ICMP message collected at border routers to infer worm activity [BGB03] and victim counter-based detection algorithm [WVGK04]. All these approaches require a large deployment of sensors or a large monitoring IP space (e.g., 2^{20} IP addresses) in order to collect distributed data for analysis. The data collection and analysis architecture is coordinated by "cyber Center for Disease Control" [SPN02]. Researchers have also proposed various data collection and monitoring architectures, e.g., "network telescopes" [Moo02] and "Internet Storm Center" [Ins].

Our objective is also to conduct early worm detection. However, considering the current diffi culty and challenges in large space monitoring system (e.g., diffi culty in data sharing, privacy, diffi culty in coordination), instead of counting on a large monitoring system and IP space for data collection, our detection mechanism is based on local networks, in particular, local honeynet or honeypots for worm detection. In our prior work [QDG$^+$] we analyzed the current worm early detection algorithms, i.e., [ZGGT03] and [WVGK04], and found the instability and high false positives when applying these techniques to local monitoring networks. Therefore, we develop a new worm detection algorithm that can identify worm activities with a low false positive rate and retrieve worm infection procedures using our correlation mechanism.

Event Correlation Several techniques have been proposed for the alert/event correlation, e.g., pre-/post-condition-based pattern matching [NCR02,CM02,CLF03], chronicles formalism [MMDD02], clustering technique [DW01] and probabilistic-based correlation technique [VS01,GHH01]. All these techniques count on a certain form of prior knowledge of attack step relationship. Our approach is different from these techniques in that our approach aims to detect *zero-day* worm attack and our correlation mechanism does not depend on prior knowledge of attack step relationship but statistical pattern analysis. Statistical alert correlation was presented in [QL03]. Our work is different in that our correlation analysis is based on variables collected over *short* observations. Time series-based analysis proposed in [QL03] is good for relatively long observation variables and requires a series of statistical tests in order to accurately correlate variables.

3 Worm Infection Cycles

If local networks do not have access to the volume of data used by global monitoring systems, what local resources can they use instead? Studying worm infections gives some insights, and identifi es what data can be collected for use in a local IDS.

Consider the simple scenario where a honeypot receives network traffi c from a remote host, and then begins to send out traffi c to different machines. One might suppose that the traffi c sent to the honeypot was a worm. Or, it could represent the actions of a live human hacker. Our suspicions become stronger if there are similarities in the traffi c sent to and generated by the honeypot. For example, if the destination ports are always the same, or if the payload is substantially similar, one can suspect automated malware.

Further proof comes when one observation follows another, i.e., when multiple honeypots are compromised in a pattern. This is not unlike how security researchers currently spot fast breaking worms and viruses. Using e-mail, IRC or instant messaging, researchers compare local observations, and quickly spot emerging patterns. We propose the use of specially modifi ed honeypot sensors to automate this process, and help detect worm outbreaks.

3.1 Model of Infection

A key assumption in our monitoring system is that the worm infection can be described in a systematic way. We fi rst note that worms may take three types of actions during an infection phase. The Blaster worm is instructive, but we do not limit our model to this example.

Blaster consists of a series of modules designed to infect a host [LUR03]. The fi rst module was based on a widely available RPC DCOM exploit that spawns a system shell on a victim host. The “egg” payload of the worm is a separate program (usually “msblast.exe”) that has undergone many revisions.

Memory Events The infection process, illustrated in Figure 1(a), begins with a probe for a victim providing port 135 RPC services. The service is overflowed, and the victim spawns a shell listening on a port, usually 4444. (Later generations of the worm use different or even random ports.) This portion of the infection phase is characterized by memory events. No disk writes have taken place, and network activity cannot (yet) be characterized as abnormal, since the honeypot merely ACKs incoming packets and appears to run basic services. Still, a buffer overflow has taken place, and the infection has begun by corrupting a process.

Network Events The Blaster shell remains open for only one connection and closes after the infection is completed. The shell is used to instruct the victim to download (often via tftp) an “egg” program. The egg can be obtained from the attacker, or a third party (such as a free website, or other compromised host.) The time delay between the initial exploit and the download of the egg is usually small in Blaster, but this may not always be the case. Exploits that wait a long period to download the egg risk having the service restarted, canceled, or infected by competing worms (e.g., Nachi). Nonetheless, some delay may occur between the overflow and the “egg” transfer. All during this time, other harmless network traffi c may arrive. This portion of the infection phase is often characterized by network traffic. Downloading the egg for example requires the honeypot to initiate TCP (SYN) or UDP traffic.

In some cases, however, the entire worm payload can be included in the initial attack packet. In such a case, network events may not be seen until much later.

Disk Events Once the Blaster egg is downloaded, it is written to a directory so it may be activated upon reboot. Some worms (e.g., Witty [LUR04]) do not store the payload to disk, but do have other destructive disk operations. Not every worm creates disk operations.

These general categories of events, although present in Blaster, do not limit our analysis to just the August 2003 DCOM worm. A local detection strategy must anticipate future worms lacking some of these events.

3.2 Improved Data Capture

Traditional worm detection models deal with worm infection at either the start or end of the cycle shown in Figure 1(a). For example, models based on darknets consider only the rate and sometimes the origin of incoming scans, the traffic at the top of the diagram. The Destination Source Correlation (DSC) model [GSQ+04] also considers scans, but also tracks outgoing probes from the victim (traffic from the bottom of the diagram). The activity in the middle (including memory and disk events) can be tracked.

Even if no buffer overflow is involved, as in the case of mail-based worms and LANMAN weak password guessing worms (e.g., pubstro worms), the infection still follows a general pattern: a small set of attack packets obtain initial results, and further network traffic follows, either from the egg deployment, or from subsequent scans.

Intrusion detection based only on incoming scan rates must address the potentially high rate of noise associated with darknets. As noted in Figure 1(a), every phase of the infection cycle may experience non-malicious network traffic. Statistical models that filter the noise (e.g., Kalman) require large data sets for input. It is no wonder, then, that scan-based worm detection algorithms have recently focused on distributed data collection. As noted above, these efforts are starting to have good results.

But what about networks that are unwilling or unable to participate in global detection strategies? The needs of local network detection motivated our initial inquiry. Our first hypothesis was that instead of increasing the quantity of data, early worm detection may be possible with a higher quality input stream. As noted above, there are many aspects of the worm infection cycle that are ignored by simple scan rate detection algorithms.

4 HoneyStat Configuration and Deployment

The foregoing analysis of the worm infection cycle generally identified three classes of events that one might track in an IDS: memory, disk and network events. As noted above in Section 2, it is difficult to track all of these events in virtual honeypots or even in stateful firewalls. Networks focused on darknets, of course, have little chance of getting even complete network events, since they generally blackhole SYN packets, and never see the full TCP payload.

A complete system is needed to gather the worm cycle events and improve the data stream for an IDS. We therefore use a HoneyStat node, a minimal honeypot created in an emulator, multihomed to cover a large address space. The deployment would typically not be interesting to attackers, because of its minimal resources (limited memory, limited drive size, etc.) Worms, however, are indiscriminating and use this configuration.

In practice, we can use VMware GSX Server as our honeypot platform. Currently, VMware GSX Server V3 can support up to 64 isolated virtual machines on a single hardware system [VMW04]. Mainstream operating systems (e.g. windows, linux, Unix, etc.) all support multihoming. For example, Windows NT allows up to 32 IP addresses per interface. So if we use a GSX server with 64 virtual machines running windows and each windows having 32 IP addresses, then a single GSX machine can have $64 * 32 = 2^{11}$ IP addresses.

In practice, we found nodes with as little as 32MB RAM and 700MB virtual drives were more than adequate for capturing worms. Since the emulators were idle for the vast majority of time, many instances could be started on a single machine. Although slow and unusable from a user perspective, these virtual honeypots were able to respond to worms before any timeouts occur.

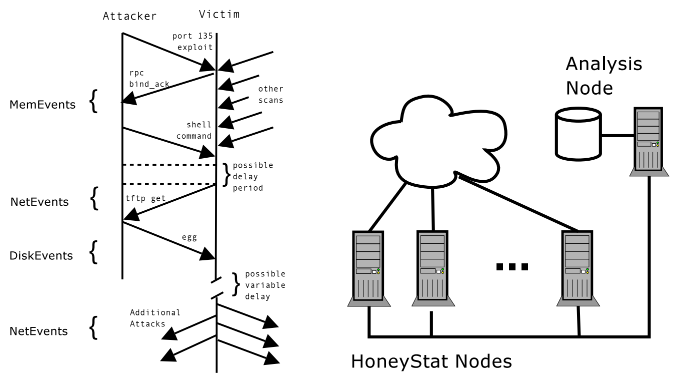
The honeypots remain idle until a *HoneyStat event* occurs. We define three types of events, corresponding to the worm infection cycle discussed in Section 3.

1. `MemoryEvent`. A honeypot can be configured to run buffer overflow protection software, such as a StackGuard [Inc03], or similar process-based monitoring tools. Any alert from these tools constitutes a HoneyStat event. Because there are no users, we found that one can use very simple anomaly detection techniques that would otherwise trigger false positives on live systems.
2. `NetworkEvents`. The honeypots are configured to generate no outgoing traffic. If a honeypot generates SYN or UDP traffic, we consider it to be an event.
3. `DiskEvents`. Within the limits of the host system, we can also monitor honeypot disk activities and trap writes to key file areas. For example, writes to systems logs are expected, while writes to `C:\WINNT\SYSTEM32` are clearly events. In practice, we found that kqueue [Lem01] monitoring of flat virtual disks was reasonably efficient. One has to enumerate all directories and files of interest, however.

Data recorded during a HoneyStat event includes:

1. The OS/patch level of the host.
2. The type of event (memory, net, disk), and relevant capture data. For memory events, this includes stack state or any core, for network events this is the outgoing packet, and for disk events this includes a delta of the file changes, up to a size limit.
3. A trace file of all prior network activity, within a bound t_p. This provides a record of traffic to the honeypot, for use in analysis. The size of t_p is discussed in Section 5.

Once events are recorded, they are forwarded to an analysis node. This may be on the same machine hosting the honeypots, or (more likely) a central server that performs logging and propagates the events to other interested nodes. Figure 1(b) shows a conceptual view of one possible HoneyStat deployment. In general, the analysis node has a secure channel connecting it with the HoneyStat servers. Its primary job is to correlate alert events, perform statistical analysis, and issue alerts.

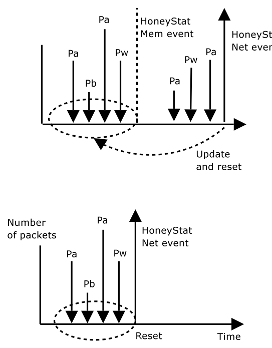


(a) Infection Cycle (b) Deployment

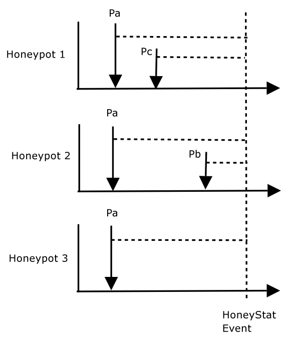
Fig. 1. a) General pattern of Blaster worm attack. Because of modular worm architectures, victims are first overflowed with a simple RPC exploit, and made to obtain a separate worm "egg", which contains the full worm. The network activity between the initial overflow and download of the "egg" constitutes a single observation. Multiple observations allow one to filter out other scans arriving at the same time. b) HoneyStat nodes interact with malware on the Internet. Alerts are forwarded through a secure channel to an analysis node for correlation.

Several actions are taken when a HoneyStat event is reported.

1. First, we check if the event corresponds to a honeypot that has already been recorded as "awake" or active. If the event is a continuation of an ongoing infection, we simply annotate the previous event with the current event type. For example, if we first witness a `MemoryEvent`, and then see a `DiskEvent` for the same honeypot, we update the `MemoryEvent` to include additional information, such as the `DiskEvent` and all subsequent network work activity. The intuition here is that `MemoryEvent`s are usually followed by something interesting, and it is worth keeping the honeypot active to track this.
2. Second, if the event involved `NetworkEvents` (e.g., either downloading an egg or initiating outgoing scans), the honeypot reporting the event is reset. The idea here is two-fold. In keeping with the principle of honeypot Data Control [LLO$^+$03], we need to prevent the node from attacking other machines. Also, once network activity is initiated, we have enough attack behavior recorded to infer that the worm is now infective. If only `DiskEvents` or `MemoryEvents` are observed, the node is not reset.



(a) Sampling Events



(b) Event Aggregation

Fig. 2. a) In the top diagram, a HoneyStat `MemEvent` occurs, and the honeypot is allowed to continue, in hopes of capturing an egg or payload. (The event is immediately analyzed, however, without delay.) If a subsequent `NetEvent` occurs, we update the previous `MemoryEvent` event and reset. In the bottom diagram, we see a `NetEvent` without any prior `MemoryEvent`, indicating our host-based IDS did not spot any anomaly. To prevent the worm from using the honeypot for attacks, we immediately reset, and analyze the circled traffic segment to find what caused the honeypot activity. b) Aggregating multiple honeypot events can help spot a pattern, and identify which port activity is common to all events.

Since the honeypot is deployed using an emulator, resets are fast in practice. One merely has to kill and restart the emulator, pointing the program in round-robin style to a fresh copy of the virtual disk. The disk image is kept in a suspended state, and no reboot of the guest OS is required. The reset delay is slight, and its effect on detection time is considered in the analysis in Section 6.

3. Third, the analysis node examines basic properties of the event, and determines whether it needs to redeploy other honeypots to match the affected OS. The intuition here is that HoneyStat nodes are often deployed to cover a variety of operating systems: Linux, Windows, and with different patch levels. If one of the systems falls victim to a worm, it makes sense to redeploy most of the other nodes to run the vulnerable OS. This improves the probability that an array of HoneyStat nodes will generate events. Again, the delay this causes for detection is discussed in Section 7.
4. Finally, the HoneyStat event is correlated with other observed events. If a pattern emerges, this can indicate the presence of a worm or other automated attack. Any reasonable correlation of events can be done. In the next section, we present a candidate analysis based on logistic regression.

As an example, in Figure 2(b), we see three different honeypots generating events. Prior input to the honeypots includes a variety of sources. For simplicity, the example in Figure 2(b) merely has three different active ports, P_a, P_b, P_c. Intuitively, we can use the time difference between the honeypot event and the individual port activity to infer what caused the honeypot to become active. But if all these events are from the same worm, then one would expect to see the same inputs to all three honeypots. In this case, only P_a is common to all three. A logistic regression presents a more flexible way of discovering the intersection of all inputs and provides a better explanation why a honeypot has become active.

5 Logistic Analysis of HoneyStat Events

Our key objective is the detection of *zero-day* worms, or those without a known signature. Without the ability to perform pattern matching, our task is analogous to *anomaly detection*. We therefore use a statistical analysis of the events to identify worm behavior. Statistical techniques, e.g., [MHL94,AFV95,PN97,QVWW98], have been widely applied in anomaly detection, . In our prior work, we applied time series-based statistical analysis to alert correlation [QL03].

Our preference was for a technique that can effectively correlate variables collected in a *short* observation window with a short computation time. Time series-based analysis is good for a relatively long observation and requires a series of statistical tests in order to accurately correlate variables. It is also often not suitable for real-time analysis because of its computationally intensive nature. Therefore, in this work, we instead apply logistic analysis [HL00] to analyze port correlation.

Logistic regression is a non-linear transformation of the traditional linear regression model. Instead of correlating two continuous variables, logistic regression considers (in the simplest case) a dichotomous variable and continuous variables. That is, the dependent variable is a boolean "dummy" variable coded as 0 or 1, which corresponds to a state or category we wish to explain. In our case, we treat the honeypot event as a dichotomous variable, i.e., the honeypot is either awake (1) or quiescent (0). Logit analysis then seeks to explain what continuous variables explain the changes in the honeypot state, from asleep to awake.

We settled on using a logit analysis only after considering other, more restrictive analysis techniques. A simple logistic regression, for example, would compare continuous to continuous variables. In the case of honeypots, this would require either measuring rates of outgoing packets, or identifying some other continuous measurement in the memory, network and disk events. Since it only takes one packet to be infected or cause an infection to spread, a simple linear regression approach would not clearly identify "sleeper worms" and worms on busy networks. Additionally, measuring outgoing packet rates would also include a signifi cant amount of noise, since honeypots routinely complete TCP handshakes for the services they offer (e.g., normal, non-harmful webservice, mail service, ftp connections without successful login, etc.). Using continuous variables based on outgoing rates may only be slightly better than using incoming scan rates.

The basic form of the model expresses a binary expectation of the honeypot state, $E(Y)$ (asleep or awake) for k events, as seen in Eq. (1).

$$E(Y) = \frac{1}{1+e^{-Z}}, \text{ where } Z = \beta_0 + \epsilon + \sum_{j=1}^{k} \sum_{i=1}^{n_j} (\beta_{i,j} X_{i,j}) \tag{1}$$

In Eq. (1), j is a counter for each individual honeypot event, and i is a counter for each individual port traffic observation for a specific honeypot. Each $\beta_{i,j}$ is the regression coefficient corresponding to the $X_{i,j}$ variable, a continuous variable representing each individual port observation. We have one error term ϵ and one constant β_0 for the equation. To set values of $X_{i,j}$, we use the inverse of time between an event and the port activity. Thus, if a `MemoryEvent` (or honeypot event j) occurs at time t, and just prior to this, port i on that same honeypot experienced traffic at time $t - \delta_t$, the variable $X_{i,j}$ would represent the port in the equation, and would have the value of $\frac{1}{\delta_t}$. This biases towards events closer in time, consistent with our infection model discussed in Section 3.

An example shows how honeypot events are aggregated. Suppose one honeypot event is observed, with activity to ports $\{P_1, P_2, \ldots, P_n\}$. We calculate the inverse time difference between the port activity and the honeypot event, and store the values for $X_{1,1}, X_{2,1}, \ldots X_{n,1}$ in a table that solves for Y. Suppose then a second event is recorded, in the same class as the first. We add the second event's values of $X_{1,2}, X_{2,2}, \ldots, X_{n,2}$ to the equation. This process continues. After each new event is added, we resolve for Y, and calculate new values of β. After sufficient observations, the logit analysis can identify candidate ports that explain why the honeypots are becoming active. (The number of observations required to have confidence in the logit results is discussed below.)

The inverse time relation between event and prior traffic allows one to record arbitrary periods of traffic. Traffic that occurred too long ago will, in practice, have such a low value for $X_{i,j}$ that it cannot affect the outcome. As a convenience, we cut off prior traffic t_p at 5 minutes, but even this arbitrary limit is generous. Future work will explore use of other time treatments, such as $\frac{1}{\delta_t^2}$, and $\frac{1}{\sqrt{\delta_t}}$, as a means of further biasing toward more recent network events. Note that this assumption prevents HoneyStat from tracking worms that sleep for a lengthy period of time before spreading. These worms are presumably self-crippling, and have a slow enough spread rate to allow for human intervention.

A key variable in this analysis includes the *Wald statistic*, which lets us test whether a variable's coefficient is zero. The Wald statistic is merely the ratio of the coefficient to its standard error, with a single degree of freedom [HL00]. The Wald statistic can be used to reject certain variables, and exclude them from a model. For example, if ports $P_0, P_1, \ldots P_n$ were observed prior to a honeypot event, we might exclude some of these ports based on the ratio of their coefficient $\beta_{i,j}$, and their standard error. Thus, the Wald statistic essentially poses a null hypothesis for each variable, and lets us exclude variables with zero coefficients. (After all, a variable with a zero β value does not contribute to solving Eq. 1). This analysis is helpful since it reduces noise in our model. However, since it uses a simple ratio, when the standard error is large, it can lead one to not reject certain variables. Thus, the Wald statistic can be used to remove unlikely variables, but might not always remove variables that have no affect.

Applying logistic analysis involves the following steps. First, for a particular honeypot event j, we estimate the coefficients, i.e., $\beta_{0,j}, \beta_{1,j} \ldots \beta_{n,j}$, using *maximum likelihood evaluation* [HL00] (MLE). In this step, we try to find a set of coefficients that minimize the prediction error. Stated another way, MLE assigns values that will maximize the probability of obtaining the observed set of data. (This is similar to the least squares method under simple regression analysis.)

Second, we use the Wald statistic to evaluate each variable, and remove those below a user-selected threshold of signifi cance level, say, 5%. The intuition of this step is that we try to evaluate whether the "causal" variable in the model is *significantly* related to the outcome. In other words we essentially ask the question: Is activity on port x signifi cantly related to the honeypot activity or was it merely random?

If the analysis results in a single variable explaining changes in the honeypot, then we report the result as an alert. If the results are conclusive, the event data is stored until additional events are observed, triggering a renewed analysis. Of course, since the events involve breakins to honeypots, users may also wish to receive informational alerts about these events, regardless of whether a worm has been detected. (As noted in Section 3, manual attacks can sometimes preceded automated attacks, such as with Blaster.)

6 HoneyStat in Practice

To evaluate HoneyStat's potential as a local worm detection system, we tested two key aspects of the algorithm: (a) does it properly identify worm outbreaks, and (b) what false positive rate does it produce? Testing showed that HoneyStat could identify worm outbreaks, with a low false positive rate. Our testing with available data showed the false positive rate of zero.

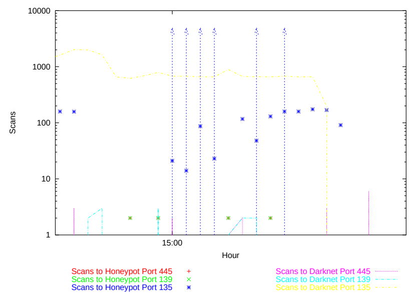
6.1 Worm Detection

In [QDG$^+$], we used data from six honeypots that became active during the Blaster worm outbreak in August 2003. The trace data used for the analysis also included network traffi c from some 100 /24 darknet IPs. Figure 3 shows an aggregate view of traffi c to all the honeypots on August 9 and 11, as well as background traffi c to the darknets.

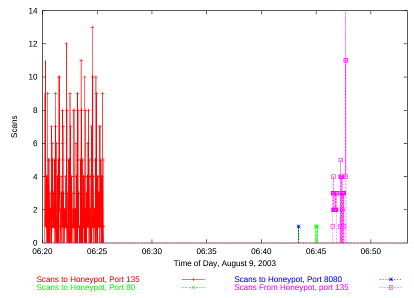
If we mark the honeypot activity as `NetEvent`s, we can examine the previous network activity to fi nd whether a worm is present. As shown in Table 1, a logit analysis of the honeypot data shows that of all the variables, port 135 explains the tendency of honeypots to become active. (In our particular example, one can even visually confi rm in Figure 3(a) that honeypot activity took place right after port 135 traffi c arrived.) The standard error reports the error for the estimated β, and the signifi cance column reports the chance that the variable's influence was merely chance. The Wald statistic indicates whether the β statistic signifi cantly differs from zero.

The signifi cance column is the most critical for our analysis, since it indicates whether the variable's estimated β is zero. The lower the score, the less chance the variable had no influence on the value of Y. Thus, we eliminate any variable with a signifi cance above a threshold (5%). From this, the observations for ports 80, 8080, and 3128 can be discounted as not a signifi cant explanation for changes in Y.

In this case, the logit analysis performs two useful tasks. First, we use the significance column to rule out variables above a certain threshold, leaving only ports 135, 139 and 445. Second, the analysis lets us rank the remaining variables by signifi cance.



(a) HoneyStat Worm Detection



(b) HoneyStat Observance of Non-Worm Events

Fig. 3. a) *HoneyStat worm detection for Blaster*. The Blaster attack on August 11 is detected by the honeypots. Upward arrows, not drawn to scale, indicate the presence of outgoing traffic from the HoneyStat nodes. Traffic prior to the honeypot activity is analyzed, using the inverse of time difference, so that more recent activities more likely explain the change in the honeypot. A logit analysis shows that prior scans to port 135 explains these episodes–effectively identifying the blaster worm. b) *Avoiding false positives*. Here, we see a trojaned honeypot node becoming active days prior to the Blaster worm outbreak. However, since this event is seen only in isolation (one honeypot), it does not trigger a worm alert. Traffic to ports 80 and 8080 does not bias the later analysis.

The logit analysis did not pick one individual port as explaining the value of Y. The alert that issues therefore identifi es three possible causes of the honeypot activity. As it turns out, this was a very accurate diagnosis of the Blaster outbreak. Recall that just prior to Blaster's outbreak on port 135, there were numerous scans being directed at ports 139 and 445. The port 135 exploit eventually became more popular, since only a few machines were vulnerable on 445 and 139. We are aware of no statistical test that could focus on port 135 alone, given the high rate of probing being conducted on ports 139 and 445. This required human insight and domain knowledge to sort out.

The number of observations required for logistic regression appears to be a matter of some recent investigation. In [HL00], the authors (eminent in the fi eld) note "there has been surprisingly little work on sample size for logistic regression". Some rough estimates have been supplied. They note that at least one study show a minimum of 10 events per parameter are needed to avoid over/under estimations of variables [HL00]. Since each honeypot activity observation is paired with a corresponding inactivity observation, HoneyStat would need to generate 5 HoneyStat events to meet this requirement. Section 7 notes how waiting for this many observations potentially affects worm detection time.

Since each event involves an actual compromise of a system, one could also report alerts with a lower confi dence level. While we might want more samples and certainty, we can at the very least rank likely ports in an alert.

Table 1. Logit Analysis of Multiple HoneyStat Events

Variable	β	Standard Error	Wald	Significance
port_80	-17463.185	2696276.445	.000	.995
port_135	3.114	.967	10.377	.001
port_139	1869.151	303.517	37.925	.000
port_445	-1495.040	281.165	28.274	.000
port_3128	-18727.568	9859594.820	.000	.998
port_8080	10907.922	10907.922	6919861.448	.999
constant	.068	1.568	.210	1.089

6.2 Benefits of HoneyStat

HoneyStat provides the following benefi ts to local networks:

1. It provides a very accurate data stream for analysis. Every event is the result of a successful breakin. This signifi cantly reduces the amount of data that must be processed, compared to Kalman fi lter, and other traditional scan-based algorithms.
2. Since HoneyStat uses complete operating systems, it detects zero day worms, for which there is no known signature.
3. HoneyStat is agnostic about the incoming and outgoing ports for attack packets, as well as their origin. In this way, it can detect worms that enter on port P_a, and exit on port P_b.

Thus, HoneyStat reports *an explanation* of worm activation, and not merely the *presence* of a worm. Other information, such as rate of scans, can be obtained from the traffi c logs captured for the logit analysis. [Kre03] has already suggested a simple method of quickly extracting a binary signature, in a manner compatible with Honey-Stat.

6.3 False Positive Analysis

Analyzing the false positive rate for HoneyStat is subtle. Since honeypot events always involve breakins and successful exploits, it might seem that honeypot-based alert systems would produce no false positives. This is not the case. Although the underlying data stream consists of serious alerts (successful attacks on honeypots), we still need to analyze the potential for the logit analysis to generate a false positive. Two types of errors could occur. First, normal network traffi c could be misidentifi ed as the source of an attack. That is, a worm could be present, but the analysis may identify other, normal traffi c as the cause. Second, repeated human breakins could be identifi ed as a worm. We do not consider this second failure scenario, since in such a case, the manual breakins are robotic in nature, and (for all practical purposes) indistinguishable from, and *potentially just as dangerous* as any worm.

Model Failure It is not feasible to test HoneyStat on the Internet. This would require waiting for the outbreak of worms, and dedicating a large IP space to a test project. We can instead perform an retrospective analysis of a tracefi le to estimate the chance of a false positive.

Using a honeypot activity log, dating from July 2002 to March 2004, we used uniform random sampling to collect background traffi c samples, and injected a worm attack. The intuition is this: we wish to see if a HoneyStat logit analysis were to cause a false positive. This could occur if normal non-malicious background traffi c occurs in such a pattern that random sampling produces a candidate solution to the logistic regression.

The data we use for the background sampling came from the Georgia Tech Honeynet project. We have almost two years of network data captured from the Honeynet. The fi rst year of data was captured on a Generation I Honeynet, which is distinguishable by the use of a reverse fi rewall serving as the gateway for all the Honeypots. The second year of data was captured from a Generation II Honeynet, which is distinguishable by the use of a packet fi ltering bridge between all of the Honeypots and their gateway. The data is available to other researchers in a sanitized form.

A random sampling of over 250 synthetic honeypot events did not produce a false positive. This certainly does not prove that HoneyStat is incapable of producing a false positive. Rather, this may reflect the limited range of the data. A much larger data set is required to fully explore the potential of logistic regression to misidentify variables.

Even if false positives are found, it should be noted that these are not the usual false positives, or type I errors found in IDS. Instead, a false positive with a HoneyStat node is half right: there are breakins to honeypots, even if the algorithm were to misidentify the cause.

7 HoneyStat as an IDS Tool

The previous sections have shown that HoneyStat can detect worm attacks with a low false positive rate. This shows that it could be incorporated into a local intrusion detection system. A more important question is whether this strategy can detect worm outbreaks early. In this section, we use an analytical model to evaluate HoneyStat's effectiveness, in particluar , the worm detection time with various HoneyStat's confi gurations.

7.1 Early Local Detection

A HoneyStat deployment can effectively detect worms that use random scan techniques. As noted in [ZGGT03], a random scan is a common scan strategy used by many worms, e.g., Code Red, Slammer, to generate a random IPv4 address to attack. Realistically we assume the vulnerable hosts are uniformly distributed in the real assigned IPv4 space (all potential victims are located in this space, denoted as $T = 10^9$), not the whole IPv4 space (denoted as $\Omega = 2^{32}$). Assume N is the total number of vulnerable machines on the Internet, n_i is the number of whole Internet victims at time tick i and s is the scan rate of worm (per time tick). So the scans entering space T at time tick $i+1$ should be $k_{i+1} = sn_i\frac{T}{\Omega}$. Within this space, the chance of one host being hit is $1 - (1 - \frac{1}{T})^{k_{i+1}}$. Then we have worm propagation equation Eq. (2).

$$n_{i+1} = n_i + [N - n_i]\left(1 - (1 - \frac{1}{T})^{sn_i \frac{T}{\Omega}}\right) \tag{2}$$

In fact because T and Ω are very big, $(1 - (1 - \frac{1}{T})^{sn_i \frac{T}{\Omega}} \doteq \frac{1}{T} sn_i \frac{T}{\Omega} = \frac{sn_i}{\Omega}$. So the spread rate is almost the same as seen in previous models (e.g., Analytical Active Worm Propagation (AAWP) model [CGK03], epidemic model [KW93,KCW93,SPN02] etc.)

Now suppose we have a honeypot network with size D ($D \subseteq T$). The initial number of vulnerable hosts is u_0. Generally a network with size D has DN/T vulnerable hosts on average. But with HoneyStat, each network has its own mix of vulnerable OS distributions. Since most worms target Windows, we can intentionally let most of our honeypots run Windows so that we present a higher number of initially vulnerable hosts to the worm. Without loss of generality we suppose $u_0 = D\alpha$. We let α be the minimum ratio for vulnerable hosts. The number of local victims at time tick i is v_i and $v_0 = 0$ which means initially there is no victim in our honeypot network. The time for the first HoneyStat node to become active is t_1 (clearly t_1 is the first time tick i when $v_i \geq 1$). We have

$$v_{i+1} = u_0 \left(1 - (1 - \frac{1}{T})^{\sum_{j=0}^{i} sn_j \frac{T}{\Omega}}\right) \text{ when } i + 1 < t_1 + t_r \tag{3}$$

Here $u_0 = D\alpha$. We let t_r represent the time required to reconfigure most of the non-vulnerable honeypots to run the same OS and patch level as the first victim. (In other words, this is the time required, say, to convert most of the Linux honeypots to Windows, if Windows is first attacked.) Since we need more observations for the logit analysis to work, as noted in Section 5, we shift some of our honeypots to match the vulnerable OS. After this response time t_r, we suppose the number of new vulnerable hosts becomes $u_1 = D\gamma$. We let γ represent the maximum ratio for vulnerable hosts. Normally $\gamma < 1$ because we may not convert *all* of our HoneyStat nodes to the operating system that is attacked. We might keep a few to detect other worms. Now we have

$$v_{i+1} = 1 + u_1 \left(1 - (1 - \frac{1}{T})^{\sum_{j=t_1+t_r}^{i} sn_j \frac{T}{\Omega}}\right) \text{ when } i + 1 >= t_1 + t_r \tag{4}$$

Here $u_1 = D\gamma$. We can calculate the time (in terms of the whole Internet infection percentage) when our first HoneyStat node is infected. Table 2 and Figure 4 use the effect of different α and D. For example, we can see that using $D = 2^{10}$ and $\alpha = 10\%$, the first victim is found when only 0.9786% Internet vulnerable hosts are infected.

When the first HoneyStat node becomes infected, it informs the other nodes what OS and patch level they should switch to. This takes time t_r (including communication time and re-deploy time), after which there will be $u_1 = D\gamma$ vulnerable hosts. After redeployment, the chance of getting the next victim improves. This is shown in Eq. (4). The effect of D, γ and t_r is shown in Figure 5.

From Figure 5 we can see that after redeployment we will quickly get enough victims when the whole Internet has a low infection percentage. The reason is that we have

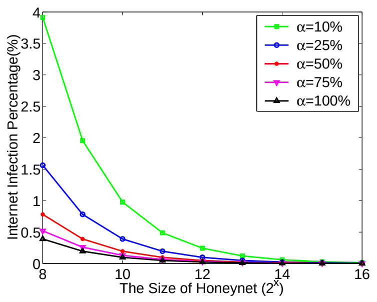
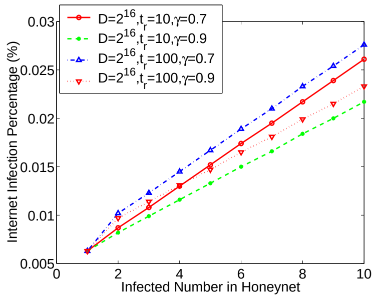
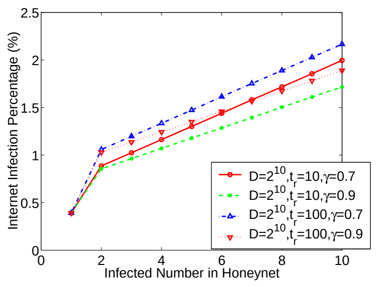


Fig. 4. Effect of α and D on Time (Infection Percentage) when HoneyStat network has a first victim. N=500,000, Scan rate=20 per time tick



(a) $D = 2^{16}$



(b) $D = 2^{10}$

Fig. 5. Effect of HoneyStat network size, D, maximum percentage of vulnerable hosts,γ, and time to redeploy after first victim, t_r, on the victim count. These graphs, drawn with $\alpha = 0.25$; N=500,000; scanrate=20 per time tick; Hitlist=1, show that with a larger IP space monitored by HoneyStat, D, the detection time (as a precent of the infected Internet) improves greatly. Even with only 2^{10} IPs monitored, detection time is quick, requiring only a little more than 1% of the Internet to be infected.

Table 2. Time (infection percentage) when HoneyStat network has a first victim

α	$D = 2^8$	$D = 2^9$	$D = 2^{10}$	$D = 2^{11}$	$D = 2^{12}$	$D = 2^{13}$	$D = 2^{14}$	$D = 2^{15}$	$D = 2^{16}$
10%	3.9141%	1.9558%	0.9786%	0.4895%	0.2448%	0.1223%	0.0613%	0.0307%	0.0155%
25%	1.5634%	0.7825%	0.3910%	0.1959%	0.0981%	0.0491%	0.0247%	0.0124%	0.0063%
50%	0.7825%	0.3910%	0.1959%	0.0981%	0.0491%	0.0247%	0.0124%	0.0063%	0.0033%
75%	0.5210%	0.2606%	0.1305%	0.0655%	0.0328%	0.0165%	0.0083%	0.0043%	0.0022%
100%	0.3910%	0.1959%	0.0981%	0.0491%	0.0247%	0.0124%	0.0063%	0.0033%	0.0017%

got more vulnerable honeypots by reconfi guring the OS and patches to the same as the fi rst victim's. Therefore, we get higher chances of being hit by the worm. For example, if $\alpha = 0.25, D = 2^{16}, \gamma = 0.7, t_r = 10$, it is still very early to have 4 victims in the HoneyStat network, when only 0.013% Internet vulnerable hosts are infected. To have 10 victims, still only 0.0261% Internet vulnerable hosts are infected. And we can see that $t_r = 10$ or $t_r = 100$, $\gamma = 0.7$ or $\gamma = 0.9$ do not affect the outcome very much. Instead, the size of honeynet D is the most important factor. Thus, the delay in switching HoneyStat nodes does not play a critical role in overall worm detection time.

In section 4, we noted that machines can be massively multihomed, so that one piece of hardware can effectively handle hundreds of IP addresses in multiple virtual machines. From the discussion of D above, 2^{11} is already a reasonable number of IP addresses that can be used in our local early worm detection. Assuming we had a few computers suffi cient to allow $D = 2^{11}$ and $\alpha = 0.25$, we can see from Table 2 that the fi rst victim appears when on average 0.1959% of Internet vulnerable hosts are infected. Suppose $\gamma = 0.75, t_r = 10$, then to have 5 victims in our honeynet (or enough to have a minimal number of data points suggested in Section 6), it is still very early when only 0.4794% of the Internet's vulnerable hosts are infected. When one IP is infected, we'll reset the OS so that it can be infected again. This kind of "replacement" policy makes the whole honeynet work as we have discussed above although there are only 64 virtual machines running on every GSX server.

Thus, HoneyStat provides an effective detection capability to local networks, assuming they have the modest collection of hardware needed to host multiple multihomed images, and (more importantly) assuming they have the extra IP space. Since most universities or businesses might have this many spare addresses in a /16 space, HoneyStat can serve as an effective local detection strategy. The general insight is that by improving the quality of the alerts, one can do early detection without having to obtain large data sets usually associated with distributed monitoring efforts. We note that this detection strategy is complimentary to other intrusion detection measures. The local network may benefi t from participating in global detection efforts as well.

8 Conclusion

Worm detection systems have traditionally used scan rates. To overcome noise in this approach, statistical models have required large data sets. In turn, this means networks must participate in global detection systems in order to detect new worm outbreaks.

This approach has merit. But local detection systems need further exploration. We have suggested that in addition to increasing the *quantity* of data used by alert systems, the *quality* can be improved as well.

It has been said that if intrusion detection is like finding a needle in a haystack, then a honeypot is like a stack of needles. Literally every event from a honeypot is a noteworthy event. Honeypots are therefore used to create a highly accurate alert stream.

Using logistic regression, we have shown how a honeypot alert stream can detect worm outbreaks. We define three classes of events, to capture memory, disk and network activities of worms. The logit analysis can eliminate noise sampled during these events, and identify a likely list of causes. Using extensive data traces of previous worm events, we have demonstrated that HoneyStat can identify worm activity. An analytical model suggests that, with enough multihomed honeypots, this provides an effective way to detect worms early.

While participation in global monitoring efforts has value, we believe local network strategies require exploration as well. Further work could include identification of additional logistic models to sort through large sets of data, coordination of shared honeypot events, integration with other intrusion detection techniques, and response.

References

[AFV95] D. Anderson, T. Frivold, and A. Valdes. Next-generation intrusion detection expert system (NIDES): A summary. Technical Report SRI-CSL-95-07, Computer Science Laboratory, SRI International, Menlo Park, California, May 1995.

[BGB03] V.H. Berk, R.S. Gray, and G. Bakos. Using sensor networks and data fusion for early detection of active worms. In *Proceedings of the SPIE AeroSense*, 2003.

[CGK03] Z. Chen, L. Gao, and K. Kwiat. Modeling the spread of active worms. In *Proceedings of the IEEE INFOCOM 2003*, March 2003.

[CLF03] S. Cheung, U. Lindqvist, and M. W. Fong. Modeling multistep cyber attacks for scenario recognition. In *Proceedings of the Third DARPA Information Survivability Conference and Exposition (DISCEX III)*, Washington, D.C., April 2003.

[CM02] F. Cuppens and A. Miège. Alert correlation in a cooperative intrusion detection framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 202–215, Oakland, CA, May 2002.

[DW01] H. Debar and A. Wespi. The intrusion-detection console correlation mechanism. In *4th International Symposium on Recent Advances in Intrusion Detection (RAID)*, October 2001.

[GHH01] R.P. Goldman, W. Heimerdinger, and S. A. Harp. Information modleing for intrusion report aggregation. In *DARPA Information Survivability Conference and Exposition (DISCEX II)*, June 2001.

[GSQ$^+$04] Guofei Gu, Monirul Sharif, Xinzhou Qin, David Dagon, Wenke Lee, and George Riley. Worm detection, early warning and response based on local victim information. Submitted for review, 2004.

[HL00] D.W. Hosmer and S. Lemeshow. *Applied Logistic Regression*. Wiley-Interscience, 2000.

[Inc03] Immunix Inc. Stackguard. http://www.immunix.org/stackguard.html, 2003.

[Ins] SANS Institute. http://www.sans.org.

[Kal60] R.E. Kalman. A new approach to linear filtering and prediction problems. *Transaction of the ASME–Journal of Basic Engineering*, March, 1960.

[KCW93] J.O. Kephart, D.M. Chess, and S.R. White. Computers and epidemiology. 1993.

[Kre03] Christian Kreibich. Honeycomb automated ids signature creation using honeypots. http://www.cl.cam.ac.uk/ cpk25/honeycomb/, 2003.

[KW93] J.O. Kephart and S.R. White. Measuring and modeling computer virus prevalence. In *Proceedings of IEEE Symposium on Security and Privacy*, 1993.

[Lem01] Jonathan Lemon. Kqueue: A generic and scalable event notification facility. pages 141–154, 2001.

[Lis01] T. Liston. Welcome to my tarpit - the tactical and strategic use of labrea. http://www.hackbusters.net/LaBrea/LaBrea.txt, 2001.

[LLO$^+$03] John Levine, Richard LaBella, Henry Owen, Didier Contis, and Brian Culver. The use of honeynets to detect exploited systems across large enterprise networks. In *Proceedings of the 2003 IEEE Workshop on Information Assurance*, 2003.

[LUR03] LURHQ. Msblast case study. http://www.lurhq.com/blaster.html, 2003.

[LUR04] LURHQ. Witty worm analysis. http://www.lurhq.com/witty.html, 2004.

[MHL94] B. Mukherjee, L. T. Heberlein, and K. N. Levitt. Network intrusion detection. *IEEE Network*, May/June 1994.

[MMDD02] B. Morin, L. Mé, H. Debar, and M. Ducassé. M2d2: A formal data model for ids alert correlation. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID)*, October 2002.

[Moo02] D. Moore. Network telescopes: Observing small or distant security events. http://www.caida.org/outreach/presentations/2002/usenix_sec/, 2002.

[MSVS03] D. Moore, C. Shannon, G. M. Voelker, and S. Savage. Internet quarantine: Requirements for containing self-propagating code. In *Proceedings of the IEEE INFOCOM 2003*, March 2003.

[NCR02] P. Ning, Y. Cui, and D.S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *9th ACM Conference on Computer and Communications Security*, November 2002.

[Par04] Janak J Parekh. Columbia ids worminator project. http://worminator.cs.columbia.edu/, 2004.

[PN97] P. A. Porras and P. G. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *National Information Systems Security Conference*, Baltimore MD, October 1997.

[Pro03] Niels Provos. A virtual honeypot framework. http://www.citi.umich.edu/techreports/reports/citi-tr-03-1.pdf, 2003.

[QDG$^+$] X. Qin, D. Dagon, G. Gu, W. Lee, M. Warfield, and P. Allor. Technical report.

[QL03] X. Qin and W. Lee. Statistical causality analysis of infosec alert data. In *Proceedings of the 6th International Symposium on Recent Advances in Intrusion Detection (RAID 2003)*, Pittsburgh, PA, September 2003.

[QVWW98] D. Qu, B. Vetter, F. Wang, and S.F. Wu. Statistical-based intrusion detection for OSPF routing protocol. In *Proceedings of the 6th IEEE International Conference on Network Protocols*, Austin, TX, October 1998.

[Sko02] E. Skoudis. *Counter Hack*. Upper Saddle River, NJ: Prentice Hall PTR, 2002.

[Spi03] Lance Spitzner. *Honeypots: Tracking Hackers*. Addison Wesley, 2003.

[SPN02] S. Staniford, V. Paxson, and N.Weaver. How to Own the Internet in Your Spare Time. In *Proceedings of 2002 Usenix Security Symposium*, 2002.

[Sta01] S. Staniford. Code red analysis pages: July infestation analysis. http://www.silicondefense.com/cr/july.html, 2001.

[VMW04] Inc. VMWare. Gsx server 3. http://www.vmware.com/products/server, 2004.

[VS01] A. Valdes and K. Skinner. Probabilistic alert correlation. In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID)*, October 2001.

[WVGK04] J. Wu, S. Vangala, L. Gao, and K. Kwiat. An efficient architecture and algorithm for detecting worms with various scan techniques. In *Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS'04)*, February 2004. to appear.

[YBJ04] Vinod Yegneswaran, Paul Barford, and Somesh Jha. Global intrusion detection in the domino overlay system. In *Proceedings of NDSS*, 2004.

[ZGGT03] C. C. Zou, L. Gao, W. Gong, and D. Towsley. Monitoring and early warning for internet worms. In *Proceedings of 10th ACM Conference on Computer and Communications Security (CCS'03)*, October 2003.

[ZGT02] C. C. Zou, W. Gong, and D. Towsley. Code red worm propagation modeling and analysis. In *Proceedings of 9th ACM Conference on Computer and Communications Security (CCS'02)*, October 2002.