# An Automated Defense System to Counter Internet Worms

Riccardo Scandariato Dipartimento di Automatica e Informatica Politecnico di Torino Corso Duca degli Abruzzi, 24 10129 Torino, Italy Phone: +39 011 564 7048 Fax: +39 011 564 7099 Email: riccardo.scandariato@polito.it John C. Knight\* Department of Computer Science University of Virginia 151 Engineer's Way, P.O. Box 400740 Charlottesville, VA 22904-4740 Phone: +1 434 982 2216 Fax: +1 434 982 2214 Email: knight@cs.virginia.edu

## Abstract

Our society is highly dependent on network services such as the Web, email, and collaborative P2P enterprise applications. But what if such infrastructures were suddenly torn down? Both past incidents and research studies show that a well-engineered Internet worm can accomplish such a task in a fairly simple way and, most notably, in a matter of a few minutes. This clearly rules out the possibility of manually countering worm outbreaks. We present a testbed that operates on a cluster of computers and emulates very large networks for purposes of experimentation. A wide variety of worm properties can be studied and network topologies of interest constructed. A reactive control system, based on the Willow architecture, operates on top of the testbed and provides a monitor/analyze/respond approach to deal with infections automatically. The logic driving the control system is synthesized from a formal specification, which is based on control rules that correlate sensor events. Details of our highly configurable testbed, the theory of operation of the Willow architecture, the features of the specification language, and various experimental performance results are presented.

## **Index Terms**

Internet worm, emulation platform, defense system, Willow architecture, reactive control, policy rules

Submission category Regular paper Approximate word count 10510

The material included in this paper has been cleared through the authors' affiliations

#### I. INTRODUCTION

Many areas of society have become increasingly dependent on networked applications and services. We rely on services like the World Wide Web, email, instant messaging, and collaborative peer-to-peer enterprise applications for critical services, productivity, entertainment and so on. This high level of dependence on the network is confirmed by the widespread diffusion and popularity of such types of services. The most recent Netcraft Web Servers Survey reports that there are about 18.5 million active sites [1], and, according to Sharman Networks, the KaZaA peer-to-peer application for file-sharing reached 200 million downloads in March 2003 [2].

But what if these infrastructures were disabled in just a few minutes? This is neither a remote nor unrealistic threat; both past incidents [3] and more recent research studies [4] have shown that a well-engineered *Internet active worm* can accomplish this task in a fairly simple way. An active worm is a unit of self-replicating malicious code that spreads from host to host by exploiting network-accessible vulnerabilities of application software and operating system components [5]. The distinguishing feature of active worms, in contrast to email-born worms (a.k.a mass mail viruses), is that infections take control of machines autonomously. This paper is about a system for the emulation of a worm infection and its use in the evaluation of an automated defense system.

The first worm to appear on the Internet was the Morris worm in 1988 [6]. The threat has become much more serious in recent years and the virulence of infections has cost the community dearly as several recent incidents have shown. For instance, the CodeRed version 2 worm was able to take over more than 359,000 IIS web servers in less than 14 hours on July 2001 [7]. The CodeRedII worm followed a month later, with an improved and more aggressive scanning technique. The figures about the net effect of that worm are not available since the network traces of CodeRedV2 and CodRedII overlapped (they were both exploiting the same vulnerability). On September of the same year, the Nimbda worm took center stage [8]. Nimbda was very "successful" due to its propagation properties as a multi-vector worm. It spread by using a mix of contagion (browsers became infected by visiting infected web servers), active scanning, and mass mailing. On January 2003, the Slammer worm [3] presented a new perspective on the potential of worm infections. It was an amazingly fast outbreak that infected nearly the whole susceptible population (more than 75,000 Microsoft SQL servers) in less than 10 minutes, highlighting how ineffective human-based countermeasures are against this kind of malicious code. The last big outbreak was very recent. Many system administrator are still dealing with the tail of the Blaster worm [9] that infected more than 330,000 Windows machines in less than 5 days, as reported by Symantec analysts. Blaster is a relatively slow worm, but due to the high density of the vulnerable population (virtually any recent Windows system was susceptible to infection), it was still able to infect a large number of targets.

The above review gives a clear picture of the real threat that current, state-of-the-art worms represent to key information infrastructures, especially if we also take into account economic losses due to the process of disinfecting

and restoring infected hosts. The situation might even worsen in the near future, if worm programmers put into practice techniques such as permutation scanning, pre-computed hit-lists, and worm cooperation [4]. Such worms could spread over the complete population of vulnerable hosts in a matter of a very few minutes. This clearly rules out the possibility of manually countering future outbreaks.

This paper pursues the idea that only an automated control mechanism can present an effective defense against fast (or flash) worms. To experiment with such automated defenses, we have designed and implemented a *testbed* for the emulation of worm behavior. The testbed provides a target for evaluation of automated defense mechanisms. The testbed allows us to safely release a worm-like attack in a real network environment. It is highly configurable in order to capture a wide variety of possible worm types and scales to tens of thousands of emulated hosts. The testbed lets users collect fine-grained information about the evolution of an ongoing infection and offers a graphical console to visualize the state of the network in real-time.

We chose to emulate (i.e., use a live system), rather than simulate, because this allows us to directly test the *defense system* we have developed. The defense system is based on the monitor/analyze/respond paradigm. It creates a control loop and listens for alarms coming from sensors spread throughout the network. When an attack situation is recognized, the system reacts accordingly by enforcing protective actions. Moreover, a live system has the additional benefit of being ready-to-use for more advanced experimentation (see Section VI). For instance, the worm testbed can be coupled with a real application, so as to study the effect of worm immunization on the survivability requirements of the application itself [10].

The logic that drives the defense system is synthesized from a formal specification written in the OOPS (Object Oriented Policy Specification) policy language. The language provides users with a simple and intuitive way to define control rules, i.e., rules that associate a set of control actions to a given pattern of alerting events (and to a set of state conditions). An event correlation engine was implemented to evaluate and enforce the installed rules. The engine analyzes the history of alerts coming from sensors and matches these against rule patterns. This process of *correlation* is executed by code generated directly from the OOPS specification by a compiler. When the engine recognizes a pattern of events (supposing that all the conditions specified in the corresponding rule hold), the control system fires the rule, and the associated actions are actuated in the network to counter the worm. Event patterns can be as simple as sets of conjoined/disjointed events, or more sophisticated temporal expressions. The support infrastructure for the event delivery and the action actuation is provided by the Willow architecture [11], as described in Section IV-A.

The use of a specification-driven approach, in general, and a policy-based specification language [12], in particular, was suggested by the continuous evolution observed in this particular application field (which is far from being mature). In the recent past, we observed a progressive evolution of strategies and techniques employed by worm

3

designers. Improvements in worm design lead to new, fast-changing scenarios that worm defense systems must face. Furthermore, techniques to effectively (i.e., both rapidly and accurately) detect worm attacks are evolving as well, necessitating the ability to seamlessly integrate new sensors. Under such conditions, a defense system must allow a high degree of flexibility and configurability. The automated generation of the control logic, together with the specification of such logic through control rules, is a step in this direction. Whenever new defense scenarios must be incorporated into the system (e.g., because of improvement in worm strategies), or new detection techniques are available, the defense system can be adapted or modified by simply changing a few rules, instead of requiring that the control system code be rewritten, a time-consuming and error-prone endeavor.

This paper presents the design and the evaluation of a platform incorporating the implementation of both the worm emulation testbed and the defense system. Accordingly, the paper is organized as follows. Section II discusses related work about worm modeling and analyzes various techniques that have been developed to detect and counter worm attacks. Section III presents the emulation testbed we developed in order to evaluate our defense system. It also presents the validation of the testbed, by comparing its behavior with different theoretical models. Section IV describes the defense system and Section V provides the experimental results we obtained by employing it in different worm scenarios. Finally, Section VI concludes by discussing key results and the directions for future work.

# II. RELATED WORK

A theoretical model to describe the infection trend of a random-scanning worm can be derived for the *homogeneous* epidemiological model in [13] and [14], by adapting it according to the parameters of our testbed. The homogeneous model assumes that each infected node can attack any other member of the population with the same probability. This assumption holds reasonably well in the case of the Internet. Since the only legal state transition for a node is from the susceptible state to the infected state (i.e., nodes are neither immunized, nor cured), this is also known as the *SI* model. By defining x(t) as the total number of infected nodes at time *t*, and  $\beta$  as the birth rate (i.e., the number of susceptible targets that a node can contact in a time unit), we have:

$$dx = x\beta(1 - \frac{x}{N})dt \tag{1}$$

The above equation can be interpreted as follows. In a time equal to dt, a node can contact  $\beta dt$  susceptible hosts. Hence, if x node are already active (i.e., infected), they will globally contact  $x\beta dt$  susceptible targets. A new infection will take place only if the target is not already infected, hence the number of targets must be scaled down to the percentage of the susceptible population (N) that is not yet infected (the term in parentheses).

Furthermore, the birth rate can be related to the scan rate (which also includes probes to invulnerable or non-

existing targets) as follows

$$\beta = \frac{N}{T}s$$

where T is the size of the address range. Hence, the law governing the spread in SI conditions is the solution of the following differential equation

$$\frac{dx}{dt} = \frac{sN}{T} \left(1 - \frac{x}{T}\right) x \tag{2}$$

By imposing that initially there are *H* infected nodes (i.e., with x(0) = H as the initial condition), we obtain the law of the infection trend of a random-scanning worm in a homogeneous environment as

$$x(t) = \frac{Ke^{\frac{sN}{T}t}}{N + Ke^{\frac{sN}{T}t}} \quad \text{with} \quad K = \frac{NH}{N - H}$$
(3)

We used this model as the main reference for the evaluation of our emulation testbed in Section III-B.

More accurate models for random-scanning worms exist. In [15], the authors show the effects on the infection trend caused by the (human-based) removal of susceptible/infected hosts (e.g., by patching or disinfecting), and by the decreasing spread rate due to the network congestion generated by the worm activity itself. The paper clearly shows that these effects only become significant at the latest stage of an outbreak. However, our aim is not to emulate worms per se. Indeed, we are interested in the study of mitigation strategies that avoid the spread of the worm into relevant parts of the network enterprises, for example. Toward this aim, it is important to precisely characterize the behavior of a worm during its *early phase* of diffusion. If the defense mechanism does not take place by that time, it will be useless, since the damage suffered by the to-be-protected systems will be too high. Removals are not relevant in our case since we are attempting to tackle the startup infection regime of extremely fast worms. Under such conditions, removals are neither in place, nor feasible. Additionally, although the decrease in the spread rate is not explicitly modeled in the testbed (because its effects are not observable during the infection phase we are interested in), our experimentations exhibited such behavior as well. This effect is a positive outcome of using a live system, which is naturally subject to network congestion conditions, as discussed later in this section.

It should be noted that the homogeneous model does not characterize topology-aware worms, nor does it capture the effect of topology constraints, e.g. physical connectivity, on the evolution of random-scanning worms. The works in [16], [17], and [18] provide theoretical foundations to mathematically analyze such effects. Furthermore, the work in [19] presents the results of a simulation study of the way in which infections propagate through certain types of network topologies (i.e., clustered and tree-like networks). A similar paper discusses the effect of immunization in scale-free networks [20]. We envision those works to be extremely valuable sources to evaluate the characteristic of our platform, when configured for topology-aware worms. Such evaluation will be the subject of future work.

A majority of research on worm defense systems is based on simulation experiments. For instance, in [21] the

authors explore the effect of dynamic quarantine on infection trends. In particular, they investigate two defense strategies: (1) black-listing of known infected nodes; and (2) filtering of connections based on worm signatures. The study assumes the existence of a notification service able to timely notify each node of either the newly infected nodes (black-listing case), or the newly discovered signature (filtering case). The time needed to notify the nodes is called the reaction time of the system, and the authors investigate the acceptable values for the reaction time in order to stop an outbreak. The work is interesting because an empirical Internet topology was used. However, the study only analyzes slow moving worm, and a prototype of the system is not provided.

The work in [22] shows the simulation result of an intrusion detection system that generates early warnings. The system is based on the monitoring of ingress/egress scans to unused addresses/ports at access points of networks. This information is gathered at a central point for analysis. The data is processed through a Kalman filter in order to estimate the spreading parameters of the worm dynamically, and consequently raise a warning. While a working implementation of the proposed system is not available, it would be interesting to employ similar techniques for our defense system, e.g. to prototype the intelligence input to the defense system (see Section IV-B and Figure 2).

The Netbait project [23] proposes a real system for the distributed detection of worms. The project uses forensics of web server logs to detect an infection, primarily through pattern matching. The information generated by the detectors on Netbait hosts is stored locally, but can be queried at the global level through a distributed query processing architecture using SQL statements. In principle, it is possible to query and then aggregate data from many nodes in order to have different viewpoints on the ongoing infection. Even though a working prototype was built, some concerns still remain about the real scalability of the system.

The work in [24] proposes a system to detect and counter worms propagation in local area networks. Each host logs its connection history, which is then transmitted to a monitoring station. The monitor builds a graph of connections and analyzes it for suspicious patterns that are likely to be indicators of a worm activity. If a worm is detected, the proper firewall rules are installed at the edge of the network, and a notification is broadcast to the nodes. In contrast to the system we will present, this work is highly customized to a specific detection mechanism. Furthermore, it is aimed at small networks, and hence will not scale to the large size we are interested in. Finally, the system is not flexible in terms of defense mechanisms: only the predetermined set of actions (hard-coded in the nodes and the firewall) can be executed to respond to an attack scenario.

Another interesting defense system is presented in [25]. In this case ICMP-T3 (destination unreachable) messages are monitored for detection. Those messages reveal connection attempts to non-existing hosts and are common for random-scanning worms. Routers are configured to copy and forward such messages to an analysis station. The analysis system looks for repeated probes to the same address (with configurable thresholds), and in such cases performs a static set of pattern matchings before generating an alarm. The system was tested on a worm emulation

of 800 logical nodes running on a single machine (in contrast, our platform scales up to 20,000 emulated nodes on 100 machines). A drawback of this work, besides its limited scale, is the lack of flexibility in the analysis engine. The system can only perform a fixed set of correlations on incoming ICMP messages. Furthermore, the system is not able to work with different detection techniques.

## III. THE WORM EMULATION TESTBED

In order to experiment with defense mechanisms against Internet worms, we designed and developed a testbed that emulates the epidemiological properties of different types of worms. The testbed allows for the construction of a network of software nodes where each node represents a possible target of infection. A node in the testbed represents the emulation of a real host running an insecure application or a vulnerable operating system. To achieve a large-scale emulation (more than 20,000 nodes), multiple nodes can be instantiated on a single physical machine.

For the sake of safety, the worm propagation is not implemented as mobile code. Instead, the malicious code is embedded as a dormant thread in the node itself. The thread listens on a server UDP socket, waiting for an activation message from an already infected peer. The testbed was implemented mainly in Java, using the Sun Java Standard Edition 1.4.1\_02. To gain better performance, the attack routine of the worm was implemented in C++ (gcc 2.95.2). The testbed was tested on a cluster of about 100 homogeneous dual 400 MHz processors PCs running Red Hat Linux version 6.2 (kernel 2.2.19), with 100 Mbps Ethernet connections.

The *susceptible* population of emulated nodes can be organized according to the different vulnerabilities that each node exposes. For instance, the population can represent all Internet hosts running the Apache web server on Windows and Linux platforms. Furthermore, a vulnerability can be specific to a version of the operating system, e.g., Windows XP, rather than Windows NT. Accordingly, in the above scenario we could model different types of worm attacks, as follows:

- all systems running Apache are vulnerable, irrespective of the underlying operating system;
- only systems running Apache on a Windows system (all variants) are vulnerable;
- systems can be infected only if they are running Apache on top of a specific version of an operating system, such as Windows NT.

# A. Testbed configuration

Nodes download their configuration at startup time from a central repository. This allows a great deal of flexibility in experimenting with different types of worm since the deployed testbed need not be modified for different experiments. All changes can be effected via a single configuration file in the repository machine. The relevant parameters that each node fetches from the repository are the *target selection strategy*, the *peer list*, and the *infection behavior*.

The target selection strategy indicates whether a node randomly scans the address space in search of new vulnerable nodes or uses on-board topological information to attack peers. In the latter case, the peer list is used to select new targets, the node continues to extract an address randomly from the list until the list is empty. This behavior is particularly useful for emulating worms that spread by gathering information from the host node, e.g., worms that spread over applications like KaZaA [26] by following the peer-to-peer topology, or worms that inspect host machine resources (e.g., browser cache, hosts file, address book, instant messenger contacts, etc.) to find vulnerable targets. Even though these techniques are commonly exployed by e-mail worms, they have not (yet) been seen in active worms. Nonetheless, it is highly probable that topology-driven attacks will be used in active worms in the near future (a recent worm used peer-to-peer technologies to install a Distributed Denial of Service network, clearly demonstrating the interest of the black-hat community in the potential of those techniques [27]). Our platform can be used to investigate such behaviors.

In random-scanning mode, each infected node indefinitely generates random addresses. The rate at which scans of possible targets are generated is called the *scan rate*. We used the Marsenne-Twister pseudo-random number generator [28] because of its large period since the number of generated addresses can be very high for fast worms. After generating an address, the node probes it to verify that it is valid, i.e., that there actually is an active node responding to that address. For random-scanning nodes, the peer list contains the addresses of all nodes in the network and is used to verify the generated addresses. Note that the replacement of a real network probe with a lookup in the peer list does not invalidate the infection model. Indeed, the time needed by a node to execute the probe of a generated address is aggregated into the scan rate. However, we are aware that this simplification reduces the amount of network traffic that the emulated worm generates as compared to a real worm. This implementation choice was necessary in order to accommodate our emulation environment without crashing the network infrastructure with extraordinary large amounts of ARP traffic [29].

The third piece of information that nodes download from the repository is the infection behavior. Nodes use this information to set the success rate and the activation time. The *success rate* is the probability that an attack exploiting a vulnerability of the node will be successful, i.e., the attack will actually lead to an infection. The same vulnerability can necessitate that different malicious payloads be used for an exploit. For instance, the Blaster worm [9] exploited a buffer overflow vulnerability in the DCOM RPC interface of Windows NT and Windows XP. In order to exploit the vulnerability, a payload specific to the OS variant must be crafted. Once it found a vulnerable target, the Blaster worm selected Windows XP 80% of the time. The success rate parameter can be conveniently used to model similar behaviors. The *activation time* is the period elapsing between the reception of a successful attack and the time when the node, in turn, starts its attack phase. This parameter can be used to model several aspects of worm's behavior. First, the actual payload of a real worm can be large enough that transmission times

Parameter	Definition				
N	Size of the susceptible population				
Vulnerabilities	The kinds of possible vulnerabilities and their stochastic distribution among the				
	population				
m	Boolean parameter denoting whether the node is randomly scanning for new				
	targets, or using the topological information				
Т	Size of the address space used for the random scans				
Topology	Adjacency matrix used by the topology-aware scans				
S	Scan rate				
Н	Hit-list size, i.e., the number of initially compromised nodes				
$P_S$	Success rate, i.e., the probability that an attack will successfully exploit a				
	vulnerability				
t <sub>A</sub>	Activation time, i.e., the time needed by an attack to activate the malicious code				
Structure	Distribution of the nodes into sites, domains, and the external world (as described				
	in Section IV)				

 TABLE I

 TUNABLE PARAMETERS OF THE EMULATION TESTBED

are relevant; in this case that time can be taken into account at the receiver end by slowing down packet processing. Second, once the malicious code is received, it takes some time to gain control of the node. Finally, worms such as CodeRedII [7], have a programmed delay that imposes a dormant period before starting the attack phase.

# B. Testbed evaluation

The testbed provides detailed information about the behavior and performance of the released worm. By analyzing the log files, it is possible to rebuild the infection history of a worm down to the level of a single node. It is possible to extract both macroscopic information, such as the total infection time or the propagation behavior (i.e., the evolution of the infected hosts along time), and microscopic information. For instance, we can analyze the success ratio of single nodes (i.e., how many targets a node infected, compared to the total number of scans), or the re-infection rate of a node. This analysis can be extended to a cluster of nodes in a region of interest (e.g., a site) as well.

We validated the testbed propagation behavior against the theoretical model of Equation 3, the results are shown in Figure 1. For the validation we deployed a set-up of 98 physical machines, each running 205 virtual nodes, for a total of 20,090 nodes (N, in Table I). We executed 10 runs under the following conditions:

- all nodes share a single vulnerability, there is a 100% probability of success during attacks ( $P_s = 1.0$ ), and infection is instantaneous ( $t_A = 0 \text{ sec}$ );
- nodes are configured to work in random-scanning mode, with a scan rate (s) of 1000 probes per second per node;
- possible targets are randomly extracted out of  $2^{28}$  address range (*T*);
- an initial hit-list (H) of 100 nodes is used.

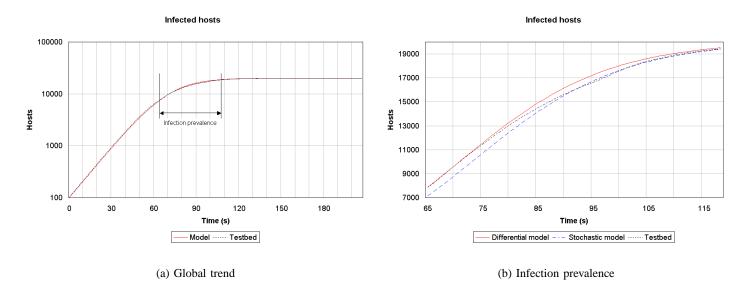


Fig. 1. Results of evaluation

With the above parameters, the infection takes about three and a half minutes to hit the total population.

These tests led to very similar data. We obtained an R-squared equals to 99.97%, which was calculated using the experimental average trend. Using the theoretical model, the R-squared value is 99.91%. The above values show that statistical significance was achieved even with a relatively small number of experiments.

Figure 1 compares the experimental data with the theoretical model. The solid line in Figure 1(a) represents the theoretical model (note the logrithmic scale), derived from Equation 3, where the involved parameters have been customized to our scenario. The dashed line represents the average values obtained from the 10 runs. The testbed tracks closely the analytical model, especially during the early stages of the infection, i.e., in the interval between 0 and 60 seconds (the exponential growth phase). This regime of the infection behavior is of particular interest as it represents the working area where the control system is supposed to operate. The experimental data yields confidence that the testbed renders an accurate picture of a real worm infection, and hence that the results of the control system (once implemented on the testbed) are realistic.

As mentioned in Section II, in the "infection prevalence" zone, i.e., the interval between 80 and 110 msec where most of the population is already infected, the testbed performs with an effective scan rate which is lower than the nominal value of 1000 probes per second per node. The slow-down is due to the high traffic load present in the network under infection prevalence conditions, as also observed in [15] and in the experimental data of past outbreaks [30]. While nodes still continue to generate attacks at the nominal scan rate (by construction), the reception at the receiver side is not instantaneous, since receiving machines are overwhelmed by UDP packets. Under infection prevalence, almost all the nodes on each machine are active (recall that multiple logical nodes are mapped to a single physical machine) and they are receiving re-infection attacks. Under such conditions, the

delivery of attacks to newly to-be-infected nodes takes more time. This behavior was unplanned but is highly beneficial because it makes the testbed more realistic.

Figure 1(b) is a close-up of the infection prevalence zone and leads to another interesting observation. In the chart, we compared the testbed with both the differential model of Equation 3, and the trend derived using the discrete-time stochastic model of Equation 4 (similarly to [31]):

$$x_{i+1} = x_i + [N - x_i][1 - (1 - \frac{1}{T})^{sx_i}] \quad \text{with} \quad x_0 = H$$
(4)

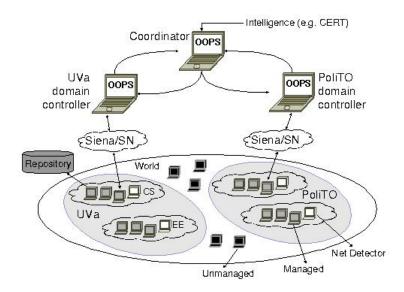
This formula can be interpreted as follows. Let us suppose that, at time tick *i*, there are  $x_i$  infected nodes. Since each infected node probes *s* addresses in a time tick, there will be  $sx_i$  scans before the time tick i+1. The events that a node will *not* be hit by any of those probes are mutually independent, and the corresponding probability is expressed by the term between round parentheses. Thus, the number of newly infected nodes at the next time tick can be obtained by multiplying the number of non-infected nodes (N-x) by the probability that each node will actually *be* hit by a scan.

As shown in Figure 1(b), the testbed behavior (dashed line in the middle) deviates from the differential model of Equation 3 (solid line in the upper part of the chart) and approaches the stochastic model of Equation 4 (dotted line in the lower part), in the time interval between 80 and 95 seconds. Afterwards, the testbed steadily tracks the latter model. This suggests that the stochastic model is more accurate for the late stage of the infection since it better represents the testbed behavior<sup>1</sup>.

# IV. THE DEFENSE SYSTEM

The defense system is designed to protect a portion of interest of the whole vulnerable population, such as an enterprise network or a critical information infrastructure. Such a portion is later referred to as a *managed domain*. As shown in Figure 2, the testbed described in Section III can be configured in order to start up two kinds of nodes: (1) managed nodes (gray nodes in the lower part), which represent hosts inside a managed domain; and (2) unmanaged nodes (black nodes), which represent hosts in the "outside" world. Nodes of the first type can be organized into *sites* (clouds, in the picture), and sites aggregated to create a domain (shaded circles). The dimension of each site, and thus of domains, can be specified as well. For each domain, a dedicated control system can be instantiated. For instance, in Figure 2, two domains are depicted: one representing the University of Virginia campus (on the left), with two department sites; the other representing the Politecnico di Torino network (on the right). In the figure, two control systems (domain controllers) are guarding their respective domains. These controllers are organized hierarchically to share information with each other through the mediation of a top level coordinator.

 $<sup>^{1}</sup>$ It is an interesting area for further work to investigate whether this property holds in general. Some preliminary experiments show encouraging results in that respect. Such study could be particularly valuable, because the stochastic model could be a simpler substitute for elaborate models proposed to model the infection prevalence regime.



#### Fig. 2. The hierarchical defense system

The communication infrastructure used by the control system is presented in Section IV-A, while Section IV-B gives the detailed description of the control system behavior.

#### A. The Willow architecture

The Willow architecture [11] is a comprehensive framework to deal with the *survivability* requirements of largescale networked application in case of complex non-local faults. Survivability [10] is ensured by both proactively eliminating faults when they are identified or suspected (but before they become manifest) and reactively tolerating the effects of faults during operation. In this work we exploit the reactive features of the Willow architecture. In particular, we exploit the Willow concept of (multiple) reactive monitor/analyze/respond loops, where the state of the networked application is continuously monitored and analyzed. If the analysis detects a fault of some kind (including physical damage, software failures, a security attack, etc.), the control system responds by issuing the necessary commands to reconfigure the network, until the system returns to a desired (possibly different) state. In a largescale scenario, especially when dealing with complex faults and multiple control loops, it is extremely important to be prepared to deal with conflict situations, for example, simultaneous and mutually conflicting actuations (e.g., a command to shutdown would conflict with a request of backup). The Willow architecture offers the proper mechanisms to deal with such conflicts.

The state of the application is gathered through notifications sent by the application itself. To this aim, the Willow architecture employs the Siena publish/subscribe event service [32]. A modified version of the Siena communication infrastructure provides the *selective notification* (SN) layer [33], and the *workflow projection* service. The latter allows the creation of logical sets of actions, with the intuitive formalisms of tasks and workflows. In particular, it

is possible to specify conflicts and precedence relationships among tasks and workflows. The service is responsible for the deployment of the tasks (actions) in the network and for the gathering of workflow completion information. Furthermore, each application node subjected to control (subscriber end) is equipped with the client side of the workflow service where conflicts are resolved by means of an intention mechanism (intentions are a high level strategy to assign priorities).

The selective notification layer lets the workflow projection service target the execution of tasks to a subset of the subscribers. Each subscriber can expose a set of attribute values, in conformance with an attribute model, and each task can be associated with a pattern of attribute values. In that case, a subscriber receives a task only if the associated pattern is compliant with its own exposed attributes. This is a key technique for the management of large-scale systems. For example, in the worm platform each managed node exposes a number of attributes including those representing the node address (for tasks directed to a specific node), the site and the domain a node is in, and the node state (susceptible, infected, or removed).

## B. Correlation of alarm events

The kernel of the defense system is the correlation engine. The defense policy is coded by means of rules, which are enforced by analyzing incoming alarms, and then reacting accordingly. In our platform, three types of alarms can be generated.

Local alarm. This type of event is generated by a local detector within each managed node each time the local node is suspected of being infected. Various local intrusion detection techniques can be employed for this function, e.g., the analysis of system log files [23] or the monitoring of outgoing connections [34]. Note that in our platform, the unmanaged nodes do not contribute to the detection activity. Since local detectors are sometimes unable to properly detect an ongoing infection, we model the *accuracy* of the detector by encompassing a configurable success rate in the detector model itself. The success rate represents the false negative rate of the local detection system, that is, the probability that an infected node will not be reported to the control system. False positives are supported, but not explicitly modeled in the worm emulation testbed. However, they may be injected in the network by directly stimulating the local detectors through an external application. Such an application could follow a statistical model or replicate the observed pattern obtained from the forensic analysis of real networks. Finally, since detection activity might be lengthy and hence could not be considered instantaneous, we have embedded a tunable *delay* parameter in the detector model. The delay represents the interval between the time the node is infected and the time the alarm is sent out by a detector.

*Site alarm.* This event is generated by network detectors, represented by white nodes inside each site in Figure 2. In our platform we modeled a honey pot intrusion detection system [36] by assigning a monitor to each site.

Note that the defense system does not rely on a particular network detection technique. The choice we made was driven by the appeal (in terms of performance) of the above mentioned technique, but different intrusion detection techniques can be easily incorporated into our platform [35]. The monitor we modeled is configured with a set of unused addresses (honey set) and listens for probes directed to that set. Each probe is obviously suspicious, since the address is not in use, and is hence reported to the control system. The performance of each site monitor is proportional to the size (U) of its honey set.

*Control alarm.* This event is generated by a domain under attack and is intended as a mean to inform the coordination controller at the upper-level. The coordinator may forward this type of alarms to other domains that federated with the originating one, in order to share worm alerts. Control alarms can also be generated by an external authority (e.g., CERT [37]), as the result of intelligence activity, as showed in Figure 2.

To guide the selection of the *U* parameter of network detectors, the following rule can be used. Let us suppose that we want our network detector to detect an outbreak before the infection has plagued the fraction *r* of the whole susceptible population *N*. That means that we want to receive a probe in the *U* address space before the critical time  $t_c$ , which represents the time the infection has reached the threshold *r* according to the analytical model of Equation 3. The critical time can be calculated by imposing x(t) = rN in Equation 3, which yields:

$$t_c = \frac{T}{sN} \log \frac{rN(N-H)}{NH(1-r)}$$

Following the same reasoning that led to Equation 4 in Section III-B, we can express the probability of having *at least* one probe in the honey set *U*, before the critical time, as follows.

$$P(t_c) = 1 - \prod_{0 \le t < t_c} (1 - \frac{U}{T})^{sx(t)}$$
(5)

Note that in the above formula, we assume a discrete time for the infection model x(t). For instance, Equation 3 can be sampled (this is the approach we followed, since that model is more accurate in the early stages of infection); otherwise, Equation 5 can be employed directly. In conclusion, by imposing the values of both r and  $P(t_c)$ , we can solve Equation 5 numerically in order to obtain a lower-bound for U.

All the alarms generated inside a domain are collected by the corresponding domain controller through the Siena network. Alarms are stored to form the current event history, which is used by the correlation engine to fire the rules. When an event pattern is matched against the event history, the rule is executed and actions sent to the domain. The controller can target the actions to a subset of domain nodes selectively using the selective notification mechanism. Once the action is received and enacted by a node, a feedback event is sent back to the controller. The same mechanism is used for the communication between controllers at different levels in the hierarchy.

Control rules are specified by means of the OOPS language. The language accommodates the specification of

sets of events (with and/or/not logical relationships), repeated events (e.g., a cluster of events of the same type), temporal sequences, and combinations of the above. Furthermore, each event pattern can be associated with an absolute time interval, e.g., "in the last 10 minutes", or a relative one, e.g., "every Monday morning". In such cases, the pattern will trigger the rule only if it happens in the proper time interval. Additionally, constraining conditions can be imposed both on the value of event attributes (e.g., the source of the event) and on state variables (e.g., the state of the domain). Finally, such values can be dynamically passed to the actions as argument, e.g., to selectively target the domain nodes on the basis of the event attributes.

## V. EVALUATION OF THE DEFENSE SYSTEM

In this Section we present the experimental results obtained from the evaluation of the control system. We defined three main metrics in order to evaluate the effectiveness of the defense system. They are:

- *Penetration ratio*: percentage of managed nodes being hit before the defense system reacts. This metric is extremely important for evaluating the effectiveness of the defense system as a whole, since it is influenced by both the accuracy of the detection system (because of false negatives), and the reactiveness of the control system (if the reaction is slow in enforcing the protective actions, new nodes will be compromised, even after the infection has been detected).
- *Infection size*: percentage of nodes that are globally infected (even in the outside world) at the time the control system reacts. This metric evaluates the ability of the defense system to detect an outbreak early on. The expected behavior is that the system will be able to detect and recover from an ongoing infection, before such infection assumes epidemic proportions.
- *Reaction delay*: time between the detection of an ongoing infection and the actual enforcement of the corresponding protective actions. This value is measured as the interval between the time a rule is fired and the time of the last feedback from the nodes involved by the rule action. This metrics is valuable in evaluating the reactiveness of the control system, independently from the accuracy of the detection system.

The above metrics are measured in three scenarios of increasing complexity. We deployed three instantiations of the worm platform as described in detail below.

#### A. Scenario 1: local detection without coordination

In this experiment, we released a very fast random scanning worm, with the following characteristics:

- N = 10,290 nodes;
- $T = 2^{24}$  addresses;
- $P_s = 1.0$ , and  $t_A = 0$  seconds;

- s = 100 probes per second per node;
- H = 10 nodes.

According to the analytical model, the infection completes in 4.4 minutes.

We created a single domain with two sites in it. The first site has 300 nodes and the other only 100. This configuration models a company network with a headquarters and a smaller branch. In this experiment network detectors are not included in order to measure the effect of local detectors on the performance of the defense system. We tested two cases: (1a) first, we used a set of local detectors with perfect accuracy, i.e., with no false positives; (1b) then, we repeated the experiments after lowering the detectors' accuracy to 80%.

For both cases, the correlation engine enforces two simple rules: (1) for each infection alarm, the corresponding node must be immunized, and (2) if a cluster of 3 nodes in the same site reported a local infection, the corresponding site must be isolated.

As observed during our experimentation, the *penetration ratio* (calculated as the number of nodes being infected in the managed domain, compared to the size of the domain itself) has a value<sup>2</sup> of 1.75%. That means that, on average, only 7 out of 400 nodes are compromised before the defenses are deployed. This is good performance, especially since at least 6 nodes must be compromised (because of Rule 2) before an ongoing infection is detected. As expected, if the detectors are faulty, i.e., with a false negative rate of 20%, the penetration ratio increases, due to the missed detection of some infected nodes. In such case, we observed an average penetration of 2.25%. This means that, on average, 3 extra nodes are hit before the worm is successfully detected inside the domain.

The *infection size* estimates the ability of the defense system to detect the worm during the outbreak's early phase. In case (1a), the worm was detected in the headquarter site before 0.78% of the population (78 nodes out of 10,000) were globally infected. For the branch site, the infection size was 2.93% (293 nodes). The higher percentage value is an obvious consequence of the branch site's smaller size. Because of the random nature of probes, it takes more time before 3 nodes in the smaller site are compromised, hence the infection has more time to replicate itself before being detected. In case (1b) the infection size for the headquarter and the branch was 1.6% (160 nodes) and 5.2% (520 nodes), respectively.

The *reaction time* (i.e., the time elapsed between the triggering of a rule in the engine and the reception of the associated action by all involved nodes) was separately calculated for Rule 1 and Rule 2. The reason is the operational difference between the two rules. The first requires a command to be transmitted to a single node (the infected one). The second rule takes more time to be effective because the corresponding action (i.e., site isolation) is selectively transmitted to all the nodes in the pertaining site. Furthermore, the larger the site, the longer the transmission time. Hence, the reaction times of Rule 2 are separately presented for the two sites. In case (1a) we

<sup>2</sup>All the reported numerical results are calculated as the average behavior we observed.

	Headquarter	Branch	
Penetration ratio	0.08%		
Infection size	0.22%	0.45%	
Reaction time	551 msec	280 msec	

 TABLE II

 Experimental results for Scenario 2

observed a reaction time of 226 milliseconds for Rule 1. The enforcement of Rule 2 performed as well as 494 msec for the headquarter site, and as 425 msec for the branch site. In case (1b) the reaction times were 258 milliseconds (Rule 1), 539 msec (Rule 2, headquarter), and 434 msec (Rule 2, branch site). The better values obtained in case (1a) compared to case (1b) are due to the later detection observed in the second case (as shown above during the discussion of the infection size metric). Indeed, because the detection was later, the network load is higher (more nodes are actively scanning), and this influences the performance in delivering commands to nodes, since commands are transmitted in-band. Nonetheless, the defense system proved to be very fast, being able to react within half a second (on average) in order to protect the domain.

# B. Scenario 2: effect of network detection

In this set of experiments, we released the same worm of Section V-A on the same network structure. However, we turned on local detectors at both sites. For the headquarter, we used a honey set of size U = 470 addresses. The above value was estimated by imposing r=1% and P = 99% in Equation 5, that is we expect to detect the outbreak before it reaches a spread of 1%. The second site has a less effective network detector because the honey set has a size of 350 addresses. Furthermore, in order to isolate the effect of the network sensors on the defense system performance, we used 100% accuracy for the local detectors.

In addition to the control rules of Section V-A, we added a third rule: (3) if a probe is sensed by a network detector, the corresponding site must be isolated. This rule is similar to Rule 2, but implies higher confidence in the network detector system over local detectors; a single probe is sufficient to infer an attack condition while in Rule 2 three alarms are necessary (e.g., because of false positives in the local detectors).

The experimental data we gathered are summarized in Table II. The only rule that fired during the experiments was Rule 3 and the *reaction times* (calculated for Rule 3) were in line with the data in Section V-A.

The results for the penetration ratio and the infection size are extremely interesting. Indeed, network detection is an extremely powerful technique that leads to a very early detection of an outbreak, even before the infection starts to spread in the managed domain. The network detectors signaled the presence of the worm in the outside world when it had infected as few as the 0.22% (headquarter) and the 0.45% (branch) of the global population (see the *infection size* entry in Table II). The lower performance of the detector in the branch site is obviously due to

	Nodes	Accuracy (%)	Delay (msec)	U (addresses)
UVa.CS	300	90	100	200
UVa.EE	100	90	100	200
PoliTO.CS	150	100	0	0
PoliTO.EE	50	100	0	0

TABLE III Configuration of Scenario 3

the smaller size of the honey set. However, the defense system proved to be very effective even with a honey set smaller than the theoretical optimal value. Moreover, as an effect of early detection, less than a node (on average) was compromised during the experiments, as reported by the *penetration ratio* entry in Table II.

## C. Scenario 3: effect of coordination

In this experiment, we released a different version of a random-scanning worm. In particular, we emulated a Slammer-like worm with the following parameters:

- N = 20,000 nodes (on a cluster of 100 PCs);
- $T = 2^{30}$  addresses;
- s = 1000 probes per second per node;
- H = 100 node.

In the real Slammer incident, the scan rate was on the same order of magnitude of our experiment, but the size N of the vulnerable population was 4 times larger. We scaled down the address range T by the same factor to maintain the proper density. With the above configuration, the infection completes within 13.6 minutes, according to the theoretical model of Equation 3.

We also deployed a hierarchical defense system. Similar to Figure 2, this network was arranged into two domains. The first domain, representing the University of Virginia (UVa), was divided into the CS, and the EE sites (e.g., to model two departments). The second domain (PoliTO) was organized into two sites as well. The configuration parameters for the two domains are summarized in Table III. The UVa domain had less accurate local detectors, but was equipped with a network detection system, while the PoliTO local detectors had perfect accuracy, but the domain had no network detectors. The expected behavior was that the PoliTO domain would benefit from the better detection capabilities of the UVa domain.

Furthermore, we supplemented the rule set with the following: (4) if both sites in a domain are under attack, notify the coordinator, and (5) if an alert is received from the coordinator, then seal the domain. The rule engine at the coordinator level had a simple rule that forwards alarms received by a domain (as a consequence of Rule 4) to the other domain(s).

In this scenario we obtained a *penetration ratio* of 1% for the UVa domain and 1.5% for the PoliTO domain. It is worth noting the positive effect of cooperation on the second domain. Indeed, since none of the sites in the PoliTO managed network had a network detector, the only rule that could protect the sites is Rule 2. This means that, in isolation, we could not have had a performance better than 3% (at least, 6 nodes out of 200 should be infected for the infection to be detected). Cooperation doubled the performance of the defense system in the PoliTO domain.

To measure the *reaction time*, we evaluated the performance of Rule 4 chained to Rule 5. We measured the time between the firing of Rule 4 (notification to the coordinator) in a domain, and the consequent completion of Rule 5 (protection of the domain after receiving a notification from the controller) in the opposite domain. This measure estimates the performance of the hierarchical control systems, since it also considers the time needed by coordinated alerts to travel up an down the controller hierarchy. The reaction time roughly doubled compared to the above scenarios, with an average value of 915 milliseconds.

Finally, the *infection size* was calculated with respect to the time when the coordinated defense took place, leading to a value of 0.38%. This figure is comparable with the one obtained in scenario 2, which is a success if we consider that only half of the sites contained network detectors.

# VI. CONCLUSIONS AND FUTURE WORK

The key contributions of this paper are the development of the first (to our knowledge) testbed for large-scale worm emulation and the design of an effective defense system that is not bound to specific detection techniques, making it general and adaptable to evolving needs. In particular, we presented a defense system that can automatically counter the spread of very fast Internet active worms, i.e., of zero-day outbreaks that can completely infect a population within a few minutes. The system is designed to protect parts of the Internet, such as corporate and institutional networks, but can also be effectively deployed on even larger portions as a result of its hierarchical structure. The rule-based approach we adopted confers great flexibility to the system, which can be tailored to a vast number of scenarios. This means that the system can be effortlessly evolved to accommodate both new worm strategies and detection techniques. We also developed a highly configurable testbed for the controlled experimentation of different types of worm outbreaks in a live network. We presented the results of the evaluation of the emulation testbed, which was able to give a realistic picture of worm behaviors.

The performance of the defense system has been extensively evaluated on top of the worm emulation testbed, demonstrating that it is able to react in about half of a second in isolation, or in less than a second in the case of inter-domain cooperation. These values were obtained in a very large scale environment, i.e., having the system running on top of up to 20,000 emulated nodes (with a cluster of 100 machines). These key results demonstrate the scalability of our approach. Furthermore, the system not only performs well in terms of scale, but also in terms of reaction time by guaranteeing recovery during the very early stages of the worm propagation and, most notably,

with a penetration ratio in the managed domain as low as 0.08% of the protected population (depending on the effectiveness of the detection techniques).

There are many directions for further extension and future work. For instance, the platform needs to be further evaluated for topology-driven worms, as outlined in Section II. The platform should also be supplemented with the implementation of an injector of false positives for the local detector network, as sketched out in Section IV-B. We are also interested in the development of more advanced defense techniques (besides the isolation of networks at risk), e.g., dynamic filtering of suspicious traffic patterns. To experiment with these techniques directly, the present implementation needs some extensions. For instance, dynamic filtering requires the platform, to a certain extent, encapsulate the concept of lower-level topology. In particular we need to detail the structure of sites (at least) in term of access points since filtering primarily takes place at those points. Finally, we envision the adoption of the emulation platform to test the combined effects of survivability and security requirements of distributed, large-scale systems. The platform can be paired with an application network in order to experiment with conflicts arising when different policies are in place. For instance, the survivability policy can require the application to perform a given set of actions to preserve functionality, while at the same time the security policy can mandate a set of conflicting actions to preserve the integrity of the application infrastructure itself.

## **ACKNOWLEDGMENTS**

This work was supported in part by the Defense Advanced Research Projects Agency under grant N66001-00-8945 (SPAWAR) and the Air Force Research Laboratory under grant F30602-01-1-0503. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, the Air Force, or the U.S. Government.

#### REFERENCES

- [1] Netcraft, Web Server Survey, Nov. 2003, on-line at http://news.netcraft.com/archives/web\_server\_survey.html
- [2] Sharman Networks, News web page, Mar. 2003, on line at http://www.kazaa.com/us/news/
- [3] D. Moore, et al., Inside the Slammer worm, IEEE Security & Privacy, Vol. 1(4) pp. 33-39, Jul. 2003
- [4] Stuart Staniford, et al., *How to own the Internet in your spare time*, Proc. of the 11th USENIX Security Symposium, San Francisco, CA, Aug. 2002
- [5] N. Weaver, et al., A taxonomy of computer worms, ACM Workshop on Rapid Malcode (WORM), Washington, DC, Oct. 2003
- [6] E. Spafford, The Internet worm: crisis and aftermath, Communications of the ACM, Vol. 32(6), pp. 678-687, Jun. 1989
- [7] D. Moore, et. al, *Code-Red: a case study on the spread and victims of an Internet worm*, Proc. of the ACM Internet Measurement Workshop, Marseille, France, Nov. 2002
- [8] CERT Security Advisory, Nimda worm, Sep. 2001, on-line at http://www.cert.org/advisories/CA-2001-26.html
- [9] CERT Security Advisory, W32/Blaster worm, Aug. 2003, on-line at http://www.cert.org/advisories/CA-2001-26.html
- [10] J.C. Knight, et al., Towards a rigorous definition of information system survivability, 3rd DARPA Information Survivability Conference and Exposition (DISCEX 2003), Washington, DC, Apr. 2003
- [11] J.C. Knight, et al., *The Willow architecture: comprehensive survivability for large-scale distributed applications*, Intrusion Tolerance Workshop, The International Conference on Dependable Systems and Networks, Washington, DC, Jun. 2002
- [12] E. Lupu, et al., Reconciling role based management and role based access control, ACM Workshop on Role Based Access Control, Fairfax, Virginia, Nov. 1997
- [13] N. Bailey, *The mathematical theory of infectious diseases and its applications, second edition*, Oxford University Press, New York, 1975
- [14] A.G. McKendrick, Applications of mathematics to medical problems, Proc. of Edin. Math. Society, Vol. 14, pp. 98-130, 1926

- [15] C.C. Zou, et al., CodeRed worm. Propagation modeling and analysis ACM Symposium on Computer and Communication Security, Washington, DC, Nov. 2002
- [16] J.O. Kephart, D.M. Chess, S.R. White, Computers and epidemiology, IEEE Spectrum, May 1993
- [17] Y. Wang, et al., *Epidemic spreading in real networks: an eigenvalue viewpoint*, IEEE Symposium on Reliable Distributed Systems, Florence, Italy, Oct. 2003
- [18] R. Pastor-Satorras and A. Vespignani, Epidemic spreading in scale-free networks, Physical Review Letters, Vol. 86(14), Apr. 2001
- [19] C. Wang, J. C. Knight, M. C. Elder, On computer viral infection and the effect of immunization, 16th Annual Computer Security Applications Conference, New Orleans, Louisiana, Dec. 2000
- [20] L. Briesemeister, et al., Epidemic profiles and defense of enterprise networks, ACM Workshop on Rapid Malcode (WORM), Washington, DC, Oct. 2003
- [21] D. Moore, et al., *Internet quarantine: requirements for containing self-propagating code*, The 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003), San Francisco, CA, Mar. 2003
- [22] C.C. Zou, et al., Monitoring and Early Warning for Internet Worms, 10th ACM Conference on Computer and Communications Security, Washington, DC, Oct. 2003
- [23] N. C. Brent, et al., Netbait: a distributed worm detection service, Intel Research Berkeley Technical Report IRB-TR-03-033, Sep. 2003
- [24] T. Toth, et al., Connection history based anomaly detection, IEEE Workshop on Information Assurance and Security, West Point, NY, Jun. 2002
- [25] V. Berk, et al, *Designing a framework for active worm detection on global networks*, IEEE International Workshop on Information Assurance, Darmstadt, Germany, Mar. 2003
- [26] Sharman Networks, KaZaA Media Desktop, On line at http://www.kazaa.com
- [27] Slapper worm, CERT Advisory CA-2002-27 Online at http://www.cert.org/advisories/CA-2002-27.html, Sept. 2002
- [28] M. Matsumoto, Y. Kurita, Twisted GFSR generator II, ACM Trans. on Modeling and Computer Simulation, Vol. 4(3), pp. 254-266, Jul. 1994
- [29] D.C. Plummer, Ethernet Address Resolution Protocol IETF Standard, RFC 0826, Nov. 1982
- [30] Cooperative Association for Internet Data Analysis (CAIDA), *The Spread of the Code-red worm (CRv2)* Online at http://www.caida.org/analysis/security/code-red/coderedv2\_analysis.xml
- [31] Z. Chen, et al., Modeling the spread of active worms The 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003), San Francisco, CA, Mar. 2003
- [32] A. Carzaniga, et al., Forwarding in a content-based network, In proc. of SIGCOMM, Karlsruhe, Germany, Aug. 2003
- [33] J. C. Hill, et al., Selective notification: combining forms of decoupled addressing for Internet-scale command and alert dissemination, Technical Report CS-2003-14, University of Virginia, Department of Computer Science, Apr. 2003
- [34] M. W. Williamson, *Throttling viruses: restricting propagation to defeat malicious mobile code*, HP Labs Bristol Technical Report HPL 2002-172R1, Dec. 2002
- [35] R. Rehman, Intrusion detection with SNORT, Prentice Hall, ISBN-0131407333, May 2003
- [36] D. Moore, Network Telescopes: observing small or distant security events, 11th USENIX Security Symposium, San Francisco, CA, Aug. 2002
- [37] CERT Coordination Center, http://www.cert.org