The Design and Evaluation of a Defense System for 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

Many areas of society have become heavily dependent on services such as transportation facilities, utilities and so on that are implemented in part by large numbers of computers and communications links. Both past incidents and research studies show that a well-engineered Internet worm can disable such systems in a fairly simple way and, most notably, in a matter of a few minutes. This indicates the need for defenses against worms but their speed rules out the possibility of manually countering worm outbreaks. We present a platform that emulates the epidemic behavior of Internet active worms. For purposes of experimentation, the platform has been deployed on a cluster of computers to emulate worm outbreaks in very large networks. A wide variety of worm properties can be studied and network topologies of interest constructed. A reactive control system, based on the Willow architecture and the OOPS policy framework, operates on top of the platform 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 correlating sensor events. Details of our highly configurable platform, 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

* Contact author

1 INTRODUCTION

Many areas of society have become heavily dependent on services that are themselves dependent on networked information systems. We rely on financial services, transportation facilities, utilities and so on; systems that are implemented in part by large numbers of computers and communications links.

But what if these infrastructures were disabled very rapidly, perhaps in just a few minutes? This is neither a remote nor unrealistic threat; both past incidents [1] and more recent research studies [2] 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 [4]. The distinguishing feature of active worms, in contrast to email-born worms (a.k.a. mass mail viruses), is that malicious code takes control of machines autonomously.

In this paper, we present the design and evaluation of a *defense system* for Internet worms. The defense system is an instantiation of the Willow survivability architecture in which a networked information system is monitored for signs of worm activity, sensor data is analyzed to diagnose an attack, and changes made to limit the attack if necessary. By choosing this strategy, we pursued the idea that only an automated control mechanism can provide an effective defense against fast (or flash) worms. We also note that, for the most part, current defense solutions are tailored to specific detection technologies, but, since worm strategies can vary significantly, defense systems must be able to integrate different sensor types in order to detect and counter worm epidemics successfully. Thus, in this research we do not focus on detection techniques. Rather, our goal is to provide a framework that is independent of specific sensor types and that is capable of integrating different detection techniques. The defense system presented supplies effective tools to analyze sensor-generated information and to deploy countermeasures in a timely, fine-grained way without requiring modification to the underlying network.

Evaluation of the defense system was conducted by experimentation. In practice, such evaluation is problematic for two reasons. The first reason is the difficulty of replicating the target systems in a laboratory, and the second reason is the difficulty of replicating accurately the infection processes of real worms. The work reported here was conducted using a *platform* that emulates large networks and their infection by worms. We approached the first problem by emulating a large (although not Internet-scale) network, i.e., we built a real network with a large number of nodes although each individual node was not emulated by an

individual computer. We approached the second problem by developing a parameterized worm that was tailored to match the observed behavior of current worms such as Slammer. The effects of worms that are expected to occur or are otherwise of interest can be assessed by setting the parameters of the platform appropriately. Furthermore, the platform can be configured to study worm types that have not been seen in the wild yet, e.g., topology-driven worms.

The paper is organized as follows. Section 2 reviews the most relevant incidents that have happened in the past. Section 3 discusses related work about worm modeling and analyzes various techniques that have been developed to detect and counter worm attacks. Section 4 presents the emulation platform and the validation study we conducted, which compares the platform behavior with theoretical models. Section 5 describes the defense system and Section 6 provides the experimental results we obtained by operating the defense system with different worm scenarios. Finally, Section 7 concludes by discussing key results of this work.

2 WORMS AND THEIR EFFECTS

The first worm to propagate widely on the Internet was the Morris worm in 1988 [5]. The threat has become much more serious in recent years as the virulence of epidemics has increased. Inevitably, the cost to the community has become substantial. For instance, the CodeRed worm was able to take over more than 359,000 IIS web servers in less than 14 hours in July 2001. On September of the same year, the Nimda worm was released causing a severe epidemic because of 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. In January 2003, the Slammer worm 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. Finally, in August 2003 the Blaster worm infected more than 330,000 Windows machines in less than 5 days. Blaster was a relatively slow worm, but the high density of the vulnerable population (virtually any recent Windows system) allowed it to infect a large number of targets.

Clearly, state-of-the-art worms present a major threat to key information infrastructures, especially if the cost of disinfecting and restoring infected hosts is taken into account. Computer Economics estimated a worldwide damage of 1 billion dollars for Slammer, 635 million for Nimda, and 2.62 billion for CodeRed.

The situation will worsen in the near future almost certainly as worm programmers develop yet more advanced techniques [2]. Sophisticated worms can infect all vulnerable hosts in a matter of a very few minutes, thereby ruling out the possibility of effective direct human intervention.

3 RELATED WORK

We begin this section by presenting theoretical models of epidemic behavior. These models are important to validate the fidelity of our platform in capturing the relevant aspects of worm attacks. Second, we analyze existing work dealing with worm detection and defense, by comparing them with the solution we propose.

Staniford [6] was the first to propose the adoption of epidemiological models to study the propagation of Internet worms analytically. His work relies on previous work by McKendrick [10] on homogeneous epidemiological model. The homogeneous model assumes that each infected node may 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. Under these assumptions, it is possible to derive a closed-form equation to represent the growth of infected nodes over time for a random-scanning worm as follows.

$$x(t) = \frac{Ke^{\frac{sN}{T}t}}{N + Ke^{\frac{sN}{T}t}} \quad with \quad K = \frac{NH}{N - H}$$
(1)

In Equation 1, x(t) is the total number of infected nodes, N is the size of the vulnerable population, s is the average scan rate (i.e., the number of attacked targets in a time unit, which also includes probes to invulnerable or non-existing targets), and T is the size of the address range used to randomly select candidate targets. K is a constant that depends on the initial conditions. In this case, an initial population of H infected nodes (hit-list) was imposed at the beginning of the epidemic. The trend represented by the equation is known as the logistic curve, which has a sigmoid shape.

The above model is quite simple and does not consider more sophisticated effects that can take place during an epidemic. Nonetheless, it models the infection growth precisely during the early stages of worm outbreaks. Furthermore, this model has proven accurate when compared to data from real observations of past epidemics [7]. For these reasons, it was selected as the reference model to validate our emulation platform.

More accurate models for random-scanning worms exist. Zou [11] studied the effects on the infection growth 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. That work shows clearly that these effects only become significant in the latest stage of an outbreak. We are interested in studying mitigation strategies that might help prevent the spread of worms into critical parts of enterprise networks, and we are interested in characterizing the behavior of a worm precisely during its *early phase* of diffusion. If the defense mechanism does not act at this time, it will be ineffective since the damage suffered by the network will be too high. Thus, removal is 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. In addition, although the decrease in the spread rate is not modeled explicitly in our platform (because effects are not observable during the infection phase in which we are interested), our experimentation exhibited such behavior as well. This effect is a positive outcome of using an emulation since it is naturally subject to network congestion conditions, as discussed below.

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. Kephart and White [12] have analyzed the effects of topology constraints over worm propagation. In particular, the authors applied the SIS (susceptible/infected/susceptible) model to random graphs. The SIS model assumes that there is both a birth rate for newly infected hosts and a cure rate for already infected hosts. The most relevant result is the demonstration that there is no epidemic if the product of the average connectivity and the birth rate is lower than the cure rate (*epidemic threshold*). However, the SIS infection model is of limited utility for the Internet worm infections. Indeed, this early work adopted the above-mentioned model because it is suitable for the propagation of old-style computer viruses (e.g. boot-sector viruses) that spread by means of floppy disks. For that type of virus, it is reasonable to assume that a host can be infected multiple times by the same virus. However, this is not the case for Internet worms. Indeed, Briesemeister [15] noted that the SI model is more appropriate for rapidly spreading malicious code. Moreover, the topological model the authors adopted is not suitable for the Internet case although it is a reasonable approximation for diskette-born viruses.

Wang [14] and Pastor-Satorras [13] investigated worm propagation behavior over realistic topologies. In particular, Wang presents the results of a simulation study to characterize epidemics in clustered and tree-

like networks, which are typical in hierarchical structures, e.g., bank networks. Pastor-Satorras proposed an epidemic model for scale-free networks, which are typical in social networks (e.g., connections among address books, and so on). We expect these results will be useful sources to evaluate the characteristic of our platform, when configured for topology-aware worms. Such evaluation will be the subject of future work.

For detection and defense systems, the majority of research is based on simulation experiments. Moore [16], for example, explores the effect of dynamic quarantine on infection trends. In particular, that work investigates 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 notify each node in a timely manner of either newly infected nodes (black-listing case), or newly discovered worm signatures (filtering case). The time needed to notify nodes is called the reaction time, and the authors investigate the acceptable values for the reaction time in order to stop an outbreak. An empirical Internet topology was used but the study only analyzed a slow moving worm, and a prototype of the system was not provided. It is not clear whether the proposed solution could be applied to fast-spreading worms, especially in consideration of the timing constrains that such a technique imposes.

Zou [17] reports the simulation results of an intrusion detection system that generates early warnings. The study supposes that monitoring systems are spread in many places of the Internet, in particular at ingress points of edge networks. Monitors detect scan activity towards unused addresses and forward alerts to a central analysis station. Alerts are processed through a Kalman filter in order to estimate the spreading parameters of the worm dynamically and consequently raise a warning. Some concerns still remain about the scalability of the proposed system, since a single analysis point must collect all alerts.

Toth [19] proposes a system to detect and counter worm propagation in local-area networks. The author extends the idea (previously introduced by Staniford [20]) of analyzing connection graphs to infer causality relationships among connections in a network. The system only works in a single broadcast segment and for TCP-based worms. The network is monitored by an analysis station that listens to the traffic in search of new connection attempts to/from local hosts and local unused addresses. Each host is also equipped with a personal firewall, whose filtering rules can be updated dynamically by digitally signed remote messages. The monitor builds a graph of connections and analyzes it for a suspicious pattern that is likely to be an indicator of worm activity. The pattern recognition is based on the causality principle, i.e. that there is a chain of connections with similar characteristics.

Berk [21] presents a defense system in which 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 generates an alarm whenever a pre-configured pattern of activity is recognized. The system was tested on a worm emulation of 800 logical nodes running on a single machine. Recently, the authors extended the work in order to test the detection system with realistic worm-generated traffic patterns [22]. The authors used the worm traffic simulator proposed in [23], which can capture the propagation behavior of a worm in terms of flows crossing a coarse-grained network topology. With this setup, the authors show that the detection system is able to recognize a CodeRed-like infection with a set of instrumented routers (i.e. routers that forward a copy of ICMP messages to the collector) covering at least 2¹⁷ unused addresses.

4 THE WORM EMULATION PLATFORM

The platform that we built for emulating worm infections allows the construction of a network of software nodes, each emulating a real host running an insecure application element 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. The platform is highly configurable so as to allow a wide variety of possible types of worm to be modeled. The platform allows detailed information about the evolution of an ongoing infection to be collected and it provides a graphical console to visualize the state of the network in real-time.

For the sake of safety, worm propagation is not implemented as mobile code. Instead, the malicious code is embedded as a dormant thread on each node. The thread listens on a server UDP socket waiting for an activation message from an already infected peer. The platform was implemented mainly in Java, using the Sun Java Standard Edition 1.4.1_02. To gain better performance, the attack component of the worm was implemented in C++ (gcc 2.95.2). The platform was deployed 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 (*N*) can be organized according to the different vulnerabilities that each node exposes. For instance, the population can represent 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. Thus, for example, we could model the following three types of worm attacks:

- All systems running Apache are vulnerable, irrespective of the underlying operating system.
- Only systems running Apache on a Windows platform (all variants) are vulnerable.
- Systems are vulnerable only if they are running Apache on top of a specific version of an operating system, such as Windows NT.

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 platform 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 scans the address space randomly 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 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 e-mail worms commonly employ these techniques, 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 potential of those techniques). Our platform can be used to investigate such behaviors.

In random-scanning mode, each infected node indefinitely generates random targets out of a configurable range of addresses (T). The rate at which scans of possible targets are generated is called the *scan rate* (s). We used the Mersenne-Twister pseudo-random number generator [24] 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. In such cases, a network message is sent out to attack the target node. 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 partially reduces the amount of network traffic that the emulated worm generates 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. Moreover, one may object that real worm-generated traffic could affect the performance of the defense system as well (which works in-band). However, since the defense system operates in the early stage of epidemics, congestion is not a relevant factor.

Finally, nodes download from the repository their infection behavior. Nodes use this information to set the success rate and the activation time. The *success rate* (P_s) is the probability that an attack exploiting a vulnerability of the target 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 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 used conveniently to model similar behaviors. The *activation time* (t_A) is the period elapsing between the reception of a successful attack and the time when the infected 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 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.

The platform provides detailed information about the behavior and performance of the released worm. By analyzing the log files it creates, one can rebuild the infection history of a worm down to the level of a single node. For example, we can analyze the success ratio of individual nodes, i.e., how many targets a node infected, compared to the total number of attempts, 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.

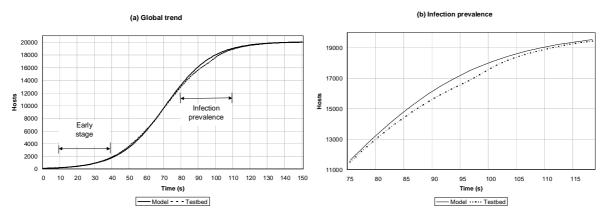


Figure 1 Results of validation

4.1 Platform validation

We assessed the propagation behavior generated by the platform for a specific sample worm by comparing it with the theoretical model of Equation 1 and the results are shown in Figure 1. For the assessment we deployed a set-up of 98 physical machines, each running 205 virtual nodes, for a total of 20,090 nodes (N). We executed 10 runs under the following four conditions:

- All nodes shared a single vulnerability, there was a 100% probability of success during attacks ($P_s = 1.0$), and infection was instantaneous ($t_A = 0$ sec).
- Nodes were configured to work in random-scanning mode, with a scan rate (*s*) of 1000 probes per second per node.
- Possible targets were randomly extracted out of 2^{28} address range (*T*).
- The hit-list (*H*) contained 100 nodes.

With the above parameters, the worm took about three and a half minutes to infect the total population. These tests led to very similar data. We obtained an R-squared equal to 99.97%, which was calculated using the experimental average trend. Using the theoretical model, the R-squared value is 99.91%. These values show that statistical significance was achieved even with a relatively small number of experiments.

Figure 1 compares the experimental data with the analytical model of Equation 1. The solid line in Figure 1(a) represents the model, where the involved parameters have been customized to our scenario. The dashed line represents the average values obtained from the 10 emulation runs. The platform tracked the analytical model closely, especially during the early stages of the infection, i.e., in the interval between 0 and 40 seconds (the early stage). This regime of the infection behavior is of particular interest as it represents the working area where the control system is supposed to operate.

Figure 1(b) shows the "infection prevalence" zone, i.e., the interval between 80 and 110 seconds where the platform performs with an effective scan rate that is lower than the nominal value. The slow-down is due to the high traffic load present in the network under infection prevalence conditions, as also observed in [11] and in the experimental data of past outbreaks [7]. While nodes still continue to generate attacks at the nominal scan rate (by construction), the reception at the receiver side is not instantaneous because UDP packets overwhelm receiving machines. Under infection prevalence, almost all the nodes in the modeled network on each physical 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 platform more realistic.

5 THE DEFENSE SYSTEM

The defense system creates a monitor/analyze/respond loop that supplements the networked application. The system monitors sensors spread throughout the emulated network and correlates sensor data to detect attacks. It then enforces defensive actions. The analytic element of the defense system is synthesized by a compiler from a formal specification written in the Object Oriented Policy Specification (OOPS) language [3]. The language provides users with an intuitive way to model defensive decisions that must be undertaken in case of attack. Scenarios of interest (e.g., an ongoing infection within the protected network) are identified by means of event patterns and decisions are modeled as control rules associating defensive actions to each pattern of alerting events. The support infrastructure for both event delivery and action deployment is provided by the Willow architecture [9].

The use of a specification-driven approach, in general, and a policy-based specification language, in particular, was suggested by the continuous evolution observed in this particular application field (which is far from being mature). There has been a progressive evolution of strategies and techniques employed by worm designers, and improvements in worm design lead to new, fast-changing scenarios that worm defense systems must face. Furthermore, techniques to detect worm attacks are evolving, necessitating the ability to integrate new sensors. The automated generation of control logic from a specification provides the flexibility necessary to permit rapid response to change. Whenever new techniques have to be incorporated into the system, the defense system can be adapted by simply changing rules in the specification rather than requiring that the control system be rewritten.

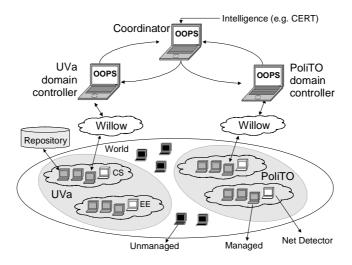


Figure 2 The hierarchical defense system

The defense system is designed to protect a portion of the vulnerable population that is deemed critical. An example of such a portion might be an enterprise network or a critical information infrastructure. We refer to such a portion as a *managed domain*. As shown in Figure 2, the emulation platform can be configured to provide two types of node: (1) managed nodes (gray nodes in the lower part of the figure) that represent hosts inside a managed domain; and (2) unmanaged nodes (black nodes) that represent hosts in the "outside" world. Managed nodes receive and execute commands from the defense system; unmanaged nodes can be organized into *sites* (clouds, in the figure), and sites aggregated to create a domain (shaded circles). The dimension of each site, and thus of domains, can be specified.

For each domain, a dedicated control system is instantiated. In Figure 2, two domains are depicted: one representing the University of Virginia (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 may be organized hierarchically to share information with each other through the mediation of a top-level coordinator.

The Willow architecture (middle of Figure 2) is a comprehensive framework to deal with *survivability* requirements of large-scale networked application in case of complex non-local faults [9]. Survivability 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 [8]. 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 by each loop. If the analysis detects a fault of some kind (including physical damage, software failures, security attacks, 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 our defense system, the state of the network is gathered through notifications sent by sensors to controllers using a simple peer-to-peer connection mechanism. A substantially modified version of the Siena publish/subscribe event service [26] implements *selective notification* (SN) [27] and *workflow projection* services.

Selective notification is a technique to manage large-scale systems, and it is used by the workflow projection service to send tasks to a subset of subscribers. Each subscriber (network node) exposes a set of attributes that describe its characteristics, and this set constitutes its "address". In turn, each task contains a set of characteristics of nodes that should receive the task. In the worm scenario, each managed node exposes its identifier, the domain and site it is in, the node configuration (e.g., the type of host it represents), and the current infection state (healthy or allegedly infected). Accordingly, the defense system can direct a task to a subset of managed nodes, e.g., to "all infected web servers in the CS site".

The workflow system allows the creation of logical sets of commands, with the intuitive formalisms of tasks and workflows. In particular, it is possible to specify both conflict and precedence relationships among tasks and workflows. The service is responsible for deploying tasks (commands) in the network and then gathering workflow completion information. Furthermore, each network node subjected to control is equipped with the client side of the workflow service, which resolves conflicts by means of the "intention" mechanism. Intentions are a high-level strategy to assign priorities. In a large-scale scenario, especially when dealing with complex faults and multiple control loops, it is extremely important to be prepared to deal with conflict situations due to simultaneous and mutually exclusive commands.

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, four types of alarms can be generated. These alarms are treated as general entities, and detectors that generate them are modeled probabilistically. Thus, the defense system abstracts from specific detection techniques employed at the lower level.

Local alarms are generated directly by managed nodes that have an on-board sensor emulating a local intrusion detector. Various local intrusion detection techniques can be employed for this function, e.g., the

analysis of system log files [18] or the monitoring of outgoing connections [28]. 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, i.e., 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 platform. 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 alarms are generated by network intrusion sensors, represented by white nodes inside each site in Figure 2. In our platform we modeled a honey pot intrusion detection system [29] 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 the platform. 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. Note that real networks are not clean environments. Thus, similarly to local sensors, network intrusion sensors can be directly stimulated, e.g. to emulate routine scanning traffic (noise).

Domain alarms are generated by domains under attack to inform the coordination level about a high level of threat. The policy followed to generate this alarm is determined by the domains themselves. For instance, a domain alarm can be forwarded to the coordinator whenever multiple sites of the same domain are under attack. In turn, the coordinator forwards alarms to other domains that federated with the originating domain.

External alarms can be produced by a third-party agency (e.g., SEI/CERT) based on evidence such as the result of intelligence activity. Intelligence can be directly injected at the controller level, or propagated through the mediation of the coordination level (as shown in the upper part of Figure 2).

To guide the selection of the *U* parameter of network detectors, the following approach can be used. Suppose that we would like network detectors to detect an outbreak before the infection has plagued a 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 1. The critical time can be calculated by imposing $x(t_c) = rN$ in Equation 1 and we can express the probability of having *at least* one probe in the honey set *U*, before the critical time, as in the following.

$$\mathbf{P}(t_{C}) = 1 - \prod_{0 \le t < t_{C}} \left(1 - \frac{U}{T}\right)^{sx(t)} \tag{2}$$

To interpret the formula, suppose that, at time *t*, there are x(t) infected nodes. Since each infected node probes *s* addresses per unit time, there will be sx(t) scans before the next time tick. The events that an unused address will *not* be hit by any of those probes for all time ticks preceding the critical time are mutually independent, and the corresponding probability is expressed by the term between parentheses. Note that in the above formula, we assume a discrete time for the infection model x(t). For instance, Equation 1 can be sampled. In conclusion, by imposing the values of both *r* and $P(t_c)$, we can solve Equation 2 numerically in order to obtain a lower-bound for *U*.

Alarms are stored to form the domain event history, and this is used by the correlation engines of controllers to activate rules. When an event pattern is matched against the event history, the rule is executed and associated commands sent to the domain. Once the command is received and enacted by a node, a feedback event is sent back to the controller. Control rules are specified by means of the OOPS language, which accommodates the specification of complex event patterns involving both logical and temporal relationships. 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., to select target nodes dynamically on the basis of the event attributes.

6 EVALUATION OF THE DEFENSE SYSTEM

Evaluation of a system such as this raises two critical questions: what are the important evaluation metrics and how can their values be determined? Different perceptions yield different desired metrics and resource limitations restrict their determination severely. In this work we defined three metrics that we consider to be generally useful and we measured their values for a set of three scenarios. Each scenario includes a worm with specific characteristics and a particular instantiation of the defense system. However, the results of the experimentation are indicative only. Indeed, the metrics that we measured are dependent on many implementation details and the size of the target network available to us. In considering the results presented here, it is essential therefore to keep this in mind. We measured times, for example, and those times are meaningful for the specific implementation on the specific target that we were using.

We defined the following three metrics to assess the effectiveness of the defense system:

- *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 responsiveness 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 it assumes epidemic proportions.
- *Reaction time*. Time between the detection of an ongoing infection and the actual enforcement of the corresponding protective actions. This value is the interval between a rule firing and receipt of the last feedback from the nodes involved in the action defined by the rule. This metric is useful in evaluating the responsiveness of the control system independent of the accuracy of the detection system.

First two metrics assess the performance and effectiveness of the defense system. The third metric is used to show that worm network activity does not impact system capabilities, even though defenses are deployed by means of in-band communication between managed nodes and the controllers. For purposes of evaluation, these three metrics were measured in the following three increasingly complex scenarios.

6.1 Scenario 1. Local detection without coordination

In this experiment, we released a medium-speed random scanning worm, with the following characteristics:

- N = 10,290 nodes (on a cluster of 49 PCs); $T = 2^{24}$ addresses;
- s = 100 probes per second per node; H = 10 nodes;
- $P_S = 1.0; t_A = 0$ seconds.

According to the analytic model, the infection should last about 4.4 minutes. We created a single domain with two sites in it. The first had 300 nodes and the other 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 detectors accuracy to 80%. This value has no special importance and was selected to test the defense system when detectors are *significantly* less dependable than the ideal case of (1a).

For both cases, the correlation engine enforces two simple rules: (1) for each local 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.

The observed *penetration ratio* (calculated as the number of nodes being infected in the managed domain, compared to the domain size) was 1.75%¹. This means that, on average, only 7 out of 400 nodes are compromised before defenses are deployed. This is good performance given that at least 6 nodes must be compromised (because of Rule 2) before an ongoing infection is detected. Repeating the experiment with faulty detectors, we observed an average penetration of 2.25%. This means that, on average, 3 extra nodes were hit before the worm was successfully detected inside the domain.

The *infection size* estimates the ability of the defense system to detect the worm during the early phase of an infection. In case (1a), the worm was detected in the headquarters site when 0.78% of the population (i.e., 78 nodes out of 10,000) had been infected. For the branch site, the corresponding infection size was 2.93%. The higher percentage value is an obvious consequence of the branch site's smaller size. Because of

¹ All the reported numerical results are calculated as the average behavior we observed.

the random nature of probes, it takes more time for 3 nodes in the smaller site to be compromised, and hence the infection has more time to replicate before being detected. In case (1b) the infection sizes for the headquarters and the branch were 1.6% and 5.2%, respectively.

The *reaction time* was separately calculated for Rule 1 and Rule 2 because of the operational difference between the two rules. The first requires a command be transmitted to a single node, namely the infected one. The second rule takes more time to be effective because the corresponding action is transmitted selectively to all nodes in the site, with the further issue that the transmission time increases with the size of the site.

In case (1a) we observed a reaction time of 226 milliseconds for Rule 1. The enforcement of Rule 2 took 494 msec for the headquarters site and 425 msec for the branch site. In case (1b) the reaction times were 258 milliseconds (Rule 1), 539 msec (Rule 2, headquarters), 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. Indeed, because of later detection, the network load is higher (more nodes are actively scanning) and this influences the performance of command delivery to nodes since commands are transmitted in-band. Nonetheless, the defense system was able to react within half a second, on average.

6.2 Scenario 2. Effect of network detection

In this set of experiments, we released the same worm as in Scenario 1 and on the same network structure. However, we turned on local detectors at both sites. For the headquarters, we used a honey set of size U = 470 addresses, which is roughly equivalent to 2 class C networks and is not unrealistic for medium to large institutions. Smaller companies may still benefit of the defense system by federating. The above value was estimated by imposing r = 1% and $P_S = 99\%$ in Equation 2, i.e., we confidently expect to detect the outbreak before it reached a penetration 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 local detectors.

In addition to the control rules of Scenario 1, we added a third rule: (3) if a network detector senses a probe, the corresponding site must be isolated. This rule is similar to Rule 2, but implies higher confidence in network detectors over local ones. Indeed, 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 local detectors.

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

Table 1 Configuration of Scenario 3

In this experiment, the only rule that fired was Rule 3 and the *reaction times* were similar to the data in Scenario 1 (551 msec for the headquarters, 280 msec for the branch site). Indeed, network detection is an extremely powerful technique that leads to very early detection of outbreaks, even before an infection starts spreading within the managed domain. The network detectors signaled the worm presence in the outside world when it had infected as few as the 0.22% (headquarters) and the 0.45% (branch) of the global population (*infection size*). The lower performance of the detector in the branch site is due to 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 one node (on average) was compromised during the experiments, as indicated by a *penetration ratio* of 0.08%.

6.3 Scenario 3. Effect of coordination

In this experiment, we released a much faster 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 of the same order of magnitude as our experiment, but the vulnerable population (*N*) 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 is estimated to last 13.6 minutes. In this scenario, we deployed a hierarchical defense system. Similar to Figure 2, the network was arranged into two domains. The first domain, representing the University of Virginia (UVa), was divided into the CS and the EE sites, i.e., 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 1. The UVa domain had less accurate local detectors, but was equipped with a network detection system (see the *U* column in Table 1), 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 alarm is received from the coordinator, seal the domain. The rule engine at the coordinator level had a simple rule that forwards domain alarms between domains.

In this scenario we measured 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 includes the time needed by domain alarms to travel up and down the controller hierarchy. The reaction time roughly doubled compared with the earlier 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. The value obtained was 0.38%. This figure is comparable to the one obtained in Scenario 2, which is a encouraging since only half of the sites contained network detectors.

7 CONCLUSIONS

We have presented details of the first (to our knowledge) platform for large-scale, high fidelity 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 counter automatically the spread of very fast Internet worms, i.e., outbreaks that can completely infect a population within a few minutes. The system is designed to protect parts of the Internet, such as corporate networks, but it can also be deployed effectively on even larger portions because of its hierarchical structure. The rule-based approach we adopted confers significant flexibility on the system because it can be tailored to a large number of scenarios. Thus the system can evolve to accommodate new detection techniques with very little effort. We also presented a highly configurable platform for controlled experimentation with different types of worm outbreaks in a live environment along with the validation results for the platform. The performance of the defense system was evaluated extensively using the worm emulation platform.

This evaluation demonstrated that the defense system can react in about half a second in isolation, or in less than a second in the case of inter-domain cooperation. These values were obtained in a large-scale, emulated environment, i.e., having the system operating on a network of up to 20,000 emulated nodes (with a cluster of 100 real 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 low penetration ratio in the managed domain.

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] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, N. Weaver, *Inside the Slammer worm*, IEEE Security & Privacy, 1(4):33-39, July 2003
- [2] S. Staniford, V. Paxson, N. Weaver, S. Staniford, *How to own the Internet in your spare time*, USENIX Security Symposium, San Francisco, CA, August 2002
- [3] R. Scandariato, Policy-based solutions for distributed systems, PhD thesis, Politecnico di Torino, April 2004
- [4] N. Weaver, V. Paxson, S. Staniford, R. Cunningham, A taxonomy of computer worms, ACM Workshop on Rapid Malcode, Washington, DC, October 2003
- [5] E. Spafford, *The Internet worm: crisis and aftermath*, Communications of the ACM, 32(6):678-687, June 1989
- [6] S. Staniford, *Analysis of spread of July infestation of the Code Red worm*, online at http://www.silicondefense.com/cr/july.html
- [7] D. Moore, C. Shannon, J. Brown, *Code-Red: a case study on the spread and victims of an Internet worm*, ACM Internet Measurement Workshop, Marseille, France, November 2002
- [8] J.C. Knight, E. Strunk, K. Sullivan, *Towards a rigorous definition of information system survivability*, DARPA Information Survivability Conference and Exposition, Washington, DC, April 2003
- [9] J.C. Knight, D. Heimbigner, A. Wolf, A. Carzaniga, J. Hill, P. Devanbu, M. Gertz, *The Willow architecture:* comprehensive survivability for large-scale distributed applications, Intrusion Tolerance Workshop, The International Conference on Dependable Systems and Networks, Washington, DC, June 2002
- [10] B. Keyfitz, N. Keyfitz, *The McKendrick partial differential equation and its uses in epidemiology and population study*, Mathematical and Computer Modeling, 26:1-9, 1997
- [11] C. Zou, W. Gong, D. Towsley, *CodeRed worm. Propagation modeling and analysis*, ACM Symposium on Computer and Communication Security, Washington, DC, November 2002
- [12] J. Kephart, S. White, Computers and epidemiology, IEEE Spectrum, May 1993
- [13] R. Pastor-Satorras, A. Vespignani, *Epidemic spreading in scale-free networks*, Physical Review Letters, 86(14), April 2001
- [14] C. Wang, J.C. Knight, M. Elder, *On computer viral infection and the effect of immunization*, IEEE Annual Computer Security Applications Conference, New Orleans, LA, December 2000
- [15] L. Briesemeister, P. Lincoln, P. Porras, *Epidemic profiles and defense of enterprise networks*, ACM Workshop on Rapid Malcode, Washington, DC, October 2003
- [16] D. Moore, C. Shannon, G. Voelker, S. Savage, *Internet quarantine: requirements for containing self-propagating code*, Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), San Francisco, CA, March 2003
- [17] C. Zou, L. Gao, W. Gong, D. Towsley, *Monitoring and early warning for Internet worms*, ACM Conference on Computer and Communications Security, Washington, DC, October 2003
- [18] B. Chun, J. Lee, H. Weatherspoon, *Netbait: a distributed worm detection service*, Intel Research Berkeley Technical Report IRB-TR-03-033, September 2003

- [19] T. Toth, C. Krugel, *Connection history based anomaly detection*, IEEE Workshop on Information Assurance, West Point, NY, June 2002
- [20] S. Staniford, S. Cheung, R. Crawford, M. Dilger, J.Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, D. Zerkle, *GrIDS. A graph-based intrusion detection system for large networks*, National Information Systems Security Conference, Baltimore, MD, October 1996
- [21] V. Berk, G. Bakos, R. Morris, *Designing a framework for active worm detection on global networks*, IEEE International Workshop on Information Assurance, Darmstadt, Germany, March 2003
- [22] M. Liljenstam, D. Nicol, V. Berk, R. Gray, Simulating realistic network worm traffic for worm warning system design and testing, ACM Workshop on Rapid Malcode, Washington, DC, October 2003
- [23] M. Liljenstam, Y. Yuan, B. Premore, D. Nicol, A mixed abstraction level simulation model of large-scale Internet worm infestations, IEEE/ACM Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Fort Worth, TX, October 2002
- [24] M. Matsumoto, Y. Kurita, *Twisted GFSR generator II*, ACM Transactions on Modeling and Computer Simulation, 4(3):254-266, July 1994
- [25] Z. Chen, L. Gao, K. Kwiat, *Modeling the spread of active worms*, Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), San Francisco, CA, March 2003
- [26] A. Carzaniga, D. Rosenblum, A. Wolf, *Design and evaluation of a wide-area event notification service*, ACM Transactions on Computer Systems, 19(3):332-383, August 2001
- [27] J. Rowanhill, P. Varner, J.C. Knight, *Efficient hierarchic management for reconfiguration of networked information systems*, The International Conference on Dependable Systems and Networks (DSN), Florence, Italy, June 2004
- [28] M.W. Williamson, *Throttling viruses: restricting propagation to defeat malicious mobile code*, HP Labs Bristol Technical Report HPL 2002-172R1, December 2002
- [29] D. Moore, *Network telescopes: observing small or distant security events*, USENIX Security Symposium, San Francisco, CA, August 2002