Tools for Worm Experimentation on the DETER Testbed

Songjie Wei
University of Delaware
103 Smith Hall
Newark, DE 19711
Email: weis@cis.udel.edu

Calvin Ko
Sparta, Inc.
710 Lakeway Dr. Ste. 195
Sunnyvale, CA 94085
Email: Calvin.Ko@sparta.com

Jelena Mirkovic
USC/ISI
4676 Admiralty Way Ste 1001
Marina Del Rey, CA 90292
Email: sunshine@isi.edu

Alefiya Hussain
Sparta, Inc.
2401 E. El Segundo Blvd. Ste. 100
El Segundo, CA 90245
Email: Alefiya.Hussain@sparta.com

***Abstract*—Worm experimentation is challenging for researchers today because of the lack of standardized tools to simulate and emulate worm spreads in a realistic setting. We have developed two tools for the DETER testbed to aid in worm experimentation: the PAWS simulator for Internet-wide worm propagation studies and the WE emulator for analysis of worm spread and defense strategies in local area networks. We evaluate performance and fidelity of our tools by replicating results from recently published research. Both tools can be easily configured as per user specifications, facilitate comparison with past research and reduce the barrier to entry for worm research.**

I. Introduction and Motivation

DETER [1] is an open testbed for security experimentation, hosted at USC Information Sciences Institute and UC Berkeley. It runs Emulab software [2] for shared testbeds, and consists of more than 400 machines. Users receive exclusive access to a number of machines they need, and set up topologies, OS and applications of their choice. DETER and EMIST project participants — USC/ISI, UC Berkeley, SPARTA, Pennsylvania State University, Purdue University and UC Davis — have developed many tools to provide realistic, easy and fast experimentation for DETER users. Most of these tools focus on supporting performance tests, denial-of-service and routing experiments, which jointly comprise 49% of projects on DETER. In this paper we describe the work we performed to support the remaining large group of our projects — 16% that focus on Internet worms.

A worm is a program that self-propagates across the network using a variety of algorithms for target discovery and then exploits a software or security flaw to copy itself onto the target and activate the next cycle of propagation. Additionally, worms also have various malicious tasks to accomplish once they infect the target, such as file modification, password-sniffing, denial-of-service, etc. Our goal was to develop tools that help users set up and automate worm experiments, while providing conditions for realistic testing. Our first step was to understand needs of the community by surveying worm research literature from top security and networking conferences: ACM Computer and Communications Security Conference (CCS), ACM Workshop on Rapid Malcode (WORM), USENIX Networking System Design and Implementation Symposium (NSDI), USENIX Symposium on OS Design and Implementation (OSDI), USENIX Security Symposium and IEEE Symposium on Security and Privacy. We surveyed total of 33 papers on worms or worm defenses, published in 2006 and 2007, looking for information about evaluation approaches. About half of the papers used a custom simulator, written by authors. One third used trace-based measurement where full packet traces (header and contents) containing legitimate and attack traffic are replayed against the proposed research product. One fifth deployed their research product in a real, well-used network. The remaining papers described research that could be validated: (1) via theory, (2) via tests that involved a single machine and a repository of malicious code and (3) via tests that involved a small network of machines and a live worm spreading in this setting.

From this survey we converged to the following conclusion. There are two types of worm research. The first, which we will call *Internet-wide*, consists of examination of worm propagations, or a wide-area network worm defense, and usually requires that an entire worm spread and realistic environment conditions be replicated at a large scale. For example, paper [3] studies how quickly one must impose an Internet quarantine on infected hosts to stop wide-spread infections. To validate this research authors needed to replicate worm spread in the Internet, which included replicating a realistic Internet environment in terms of number of hosts, topology and routing. Replication of scale and environment in which the worm is spreading mandates simulation, since existing testbeds cannot meet these goals due to limited resources and limited diversity.

But the current practice of each author writing their own custom simulator reduces evaluation validity because the simulator's fidelity is not proven. It also disables comparison between related research since authors must rewrite others' simulators to compare own work with the existing approaches. Thus researchers would benefit from a single, customizable, realistic and versatile worm simulator. We have developed such a simulator, called PAWS, and verified that it exceeds current simulators in fidelity and scalability, via extensive tests [4]. PAWS is now a part of DETER, and we describe it in section II. Due to its modular design, PAWS can be easily customized. In Section III we demonstrate this by replicating results from selected, highly cited worm papers, using PAWS and achieving matching graphs.

The second type of worm research, which we will call *localized*, consists of proposing a worm detection or defense to be deployed on a single machine, or in a single network. Such a product must be tested by sending realistic traffic to it to examine if it correctly detects or defends from worms, and if it correctly recognizes legitimate traffic. Unfortunately, two approaches that were widely used in worm papers we surveyed — replay of a full-packet trace or deployment in a real network — are not accessible to a vast majority of researchers. Collecting a full-packet trace poses a lot of privacy concerns because all packets' contents must be preserved in original. Many organizations will not let their employees collect such traces to avoid violation of user privacy rights and liability. Needless to say, full-packet traces are never shared if they contain any legitimate traffic. Even full-packet traces with malicious traffic only are rarely shared and we are aware of only two such traces: CAIDA's trace of Witty worm spread and CAIDA's trace of DDoS backscatter, both available via `http://www.datcat.org`. Deployment of products in real networks is only possible if a researcher is employed by an organization which hosts a sufficiently large network, and is willing to deploy research-grade products in it. This is a condition many researchers do not meet.

To support evaluation of localized worm research in DETER testbed one would need realistic legitimate and worm traffic generators, with realistic payloads. We have developed legitimate traffic generator for Web traffic and a worm emulator, called WE, built upon the Metasploit tool [5]; we describe them in Section IV. The advantage of using Metasploit is that a user can combine various exploits and worm payloads, thus customizing the tool to her needs. In Section V we demonstrate the tool's performance and scalability through DETER experiments. We believe that these are the first necessary steps to support localized research in testbeds.

A. Contributions

The main contribution of our work is development of tools to aid worm experimentation in testbeds. Such tools are necessary to standardize worm research testing and to ensure realistic, comparable evaluation strategies. Our tools are described in this paper, along with convincing arguments, derived from tests, that they produce realistic, useful conditions for worm research testing. Tools, including their source code, are publicly available via DETER testbed. Researchers will benefit from our work by reducing their test setup time and by experimenting in a realistic setting, which will increase validity of their research.

II. Worm Simulator: PAWS

Popular network simulators, such as ns-2 [6], GTNetS [7] and SSFNet [8], have been extended to support Internet worm simulation. Worm researchers have also implemented their own worm simulators to examine worm dynamics and to validate various approaches for worm spread detection and defense [9] [3] [10]. Our *PArallel Worm Simulator* or PAWS [4] is an Internet-scale worm spread simulator, designed for scalability and with a realistic model of Internet environment.

There are three significant advantages of PAWS over other simulators. First, PAWS incorporates a detailed model of the Internet at the Autonomous System (AS) level. Internet topology and inter-AS routing are replicated according to the realistic global routing information obtained from Route Views [11]. Inter-AS links and their limited bandwidth are simulated to capture the congestion impact that is normally observed during the spreads of aggressive worms. Each vulnerable host's features are simulated separately, including physical resources that affect its scan generation rate. Second, PAWS is a distributed simulator that runs on multiple common PCs. Each physical machine simulates a portion of the Internet. Machines synchronize with each other at discrete time intervals, instead on per-packet basis, which improves scalability at a minimal cost to simulation fidelity. Higher resource demands such as those that arise in worm simulations with a large vulnerable population (e.g., 10 million) or a more sophisticated worm scanning mechanism (e.g. high scanning rate, non-uniform scanning strategy, multi-stage infection procedure), can be supported by engaging more simulation machines. We have implemented PAWS on the Emulab [2] testbed and the DETER [1] testbed. This choice of open, shared testbeds as implementation platforms makes PAWS accessible to any researcher. The third advantage lies in the fact that users can easily configure and customize PAWS to meet various research objectives of studying Internet worms. PAWS's Internet model can be customized with either the standard Route Views data source or a specific user input of routing and topology data. PAWS's vulnerable host model can be extended by adding user-implemented host behavior functions. And PAWS's worm spread model is configurable to mimic any past or predictable future worms with complex features. In the rest of this section, we briefly review PAWS's design and discuss how it can be customized.

A. Realistic Internet Model

PAWS models the Internet at the AS level. A realistic Internet model is necessary for a high-fidelity simulation of any interactions between a worm spread and the Internet environment.

PAWS uses data from the Route Views project [11] to reconstruct the Internet topology at the AS level. The Route Views data provides periodic snapshots of the BGP routing tables for the participating routers. Such routing information could be used to infer the inter-AS connectivity and intra-AS IP address allocation. The data is updated twice daily, which enables PAWS to reconstruct the Internet topology as it was at the specific time of an observed worm spread.

PAWS reconstructs inter-AS routing and forwarding in the following manner. Each AS is represented by a single router. Forwarding entries are first populated by inferring them from the AS-path variable in the Route Views data. Remaining vacant entries are then populated by calculating next hop ASes using the shortest path approach on the AS topology.

Propagation of aggressive worms such as Slammer [12] and Witty [13] produces huge scan volume that can quickly create severe congestion due to limited link bandwidths, interfering with legitimate traffic and with Internet routing. PAWS models the limited bandwidth of each inter-AS link by preassigning some bandwidth value to it. When the traffic demand (sum of legitimate and attack traffic) on a link exceeds its bandwidth, each packet will be dropped with a probability proportional to the ratio of their difference and the traffic demand.

We infer link bandwidths using the Pathneck [14] data to estimate a possible range of bandwidth values. More details about this inference process can be found in [4].

B. Distributed and Discrete Simulation

PAWS simulation tasks are shared by multiple physical machines, which exchange their local results over a network. Each machine simulates a portion of the whole Internet along with its vulnerable and infected population. Each machine hosts roughly the same number of vulnerable hosts, which equalizes the CPU cost among machines.

PAWS collects all the worm scans to the same destination machine and uses stream sockets to exchange information about this cross-machine traffic at the end of each simulation interval (one second). This discrete simulation approach improves scalability and does not compromise fidelity. Worm scans sent to non-routable addresses or non-vulnerable hosts are dropped or processed only on the sender-side machine and thus do not increase inter-machine communication.

C. Configuration and Customization

In this section, we explain how to configure and customize PAWS to simulate different worm spread events.

A worm spread event in PAWS is defined by specifying the worm's features and the network environment: (1) scanning rate, in scans per second, (2) scanning strategy (e.g., uniform or subnet), (3) worm's transport protocol (TCP or UDP), (4) vulnerable population size and distribution (uniform or log-normal), (5) size of a worm scan, in bytes, (6) infection delay, and (7) lifetime of an infectee.

Users can also configure the Internet environment model for each PAWS simulation. This entails the procedure of obtaining the Route Views data for a specific date of worm propagation, and processing it into the format required by PAWS.

For some simulation tasks it may be necessary for users to customize PAWS's Internet model and/or packet handling to simulate more sophisticated worm events or worm defenses. There are two possible approaches to customization: *overwriting input data files* and *overloading program functions*.

PAWS reads in the configuration information for the Internet model from several data files: (1) *AS link file*, which defines all the inter-AS connections and the bandwidth of each link, (2) *IP ownership file*, which lists all the IP ranges and an AS that owns each one, and (3) *AS routing file*, which contains the forwarding tables for all the inter-AS routes. Users can replace or modify the existing files to achieve their specific simulation goals.

Users can further overload the functions called in PAWS every simulation second, to create specific worm propagation events or to implement user-specific worm detection and defense systems.

- *worm_infectee_scan()* simulates each infectee sending out scans. It can be modified to support limited or rate-variable scanning.
- *generate_ip_target()* is called by a worm infectee to select the next target to scan. It can be modified to implement novel worm scanning strategies.
- *calculate_routing_path()* is executed to calculate the routing path of each worm scan. It can be modified to simulate novel detection or defense mechanisms between ASes.
- *process_worm_scan()* is called to transfer a worm scan from the source to the destination. It can be modified to control the worm scan delay, drop and retransmission.
- *infect_vulnerable_host()* is called when a worm scan reaches a vulnerable host. It can be modified to simulate novel infection procedures.
- *update_infectee_status()* is called to update the status of each infected host. It can be modified to simulate a worm infectee's behaviors other than sending out scans, and to simulate human countermeasures.

The function *determine_vulnerable_host()* is executed once during the initialization stage of each worm simulation. This function creates vulnerable hosts in the simulated Internet environment. Each vulnerable host can be configured separately for its location, scanning rate, and scanning lifetime. By overloading this function, users can define heterogeneous vulnerable hosts with features and locations following user-specified distributions.

III. Worm Simulator Experiments

We demonstrate the usability of our PAWS simulator for different worm experiments, by replicating selected experiments conducted by other researchers using custom simulators. These experiments appear in highly visible publications: [9], [3] and [10]. Our replication also illustrates how to configure and customize PAWS for user-specific simulations.

Zou et al proposed a host-based dynamic quarantine system to contain Internet worm propagation [9]. In their system, a suspicious host is blocked from sending traffic from a specific port for a short period of time. The authors simulated the Slammer worm [12] propagation and enforced their quarantine policy on the simulated worm infectees. Replication of their experiment helps us illustrate how one can implement a host-based worm defense in PAWS. Table I lists the original experiment settings in [9] and our configuration and customization of PAWS to match these settings. Figure 1 compares published results (line without markers) with the our results (line with markers). There is a close match between theirs and our results.

Moore et al propose another worm quarantine system, called Internet quarantine [3]. The paper examines the impact an Internet-wide deployment would have on containing fast worms using two containment approaches: *address blacklisting* and *content filtering*. Address blacklisting drops scans from

TABLE I
SIMULATION OF SLAMMER WORM PROPAGATION UNDER DYNAMIC QUARANTINE DEFENSE

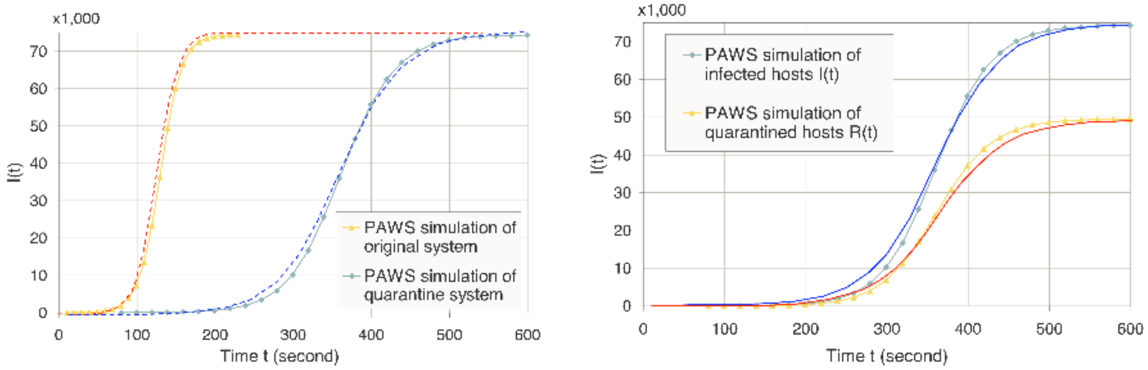
Original simulation	**PAWS simulation**
IPv4 address space with 2^{32} addresses	Default Internet model includes entire IPv4 address space
Vulnerable population is $N = 75,000$	Vulnerable_population = $75,000$
Average scan rate is $\eta = 4,000$ per second	Scanning_rate = $4,000$ per second
10 initial infectees at the beginning	In function *determine_vulnerable_host()*, randomly mark 10 vulnerable hosts as infected
Simulation time unit is 0.05 second	Set the simulation interval as 0.05 second
Dynamic quarantine, with quarantine rate $\lambda_1 = 0.2$ per second and quarantine time $T = 10$ seconds	In function *update_infectee_status()*, if an infectee is currently active, mark it as quarantined with a probability of 0.2 per second. In function *update_infectee_status()*, if an infectee has been quarantined for 10 seconds, mark it as active. In function *worm_infectee_scan()*, if an infectee is currently quarantined, skip its scanning activity for the current simulation interval.

TABLE II
SIMULATION OF CODE RED V2 WORM PROPAGATION UNDER INTERNET QUARANTINE DEFENSE

Original simulation	**PAWS simulation**
IPv4 address space with 2^{32} addresses	Default Internet model includes entire IPv4 address space
Vulnerable population is $360,000$	Vulnerable_population = $360,000$
Average scan rate is 10 per second	Scanning_rate = 10 per second
Address blacklisting, reaction time R	In function *check_routing_path()*, if the sender of a worm scan has been known as infected (thus its IP is on the blacklist) by any hop on the routing path, drop this scan. The IP of each infectee is added to the blacklists R seconds after its infection.
Content filtering, reaction time R	In function *check_routing_path()*, if any hop in the routing path participates in the containment system and knows the worm signature (only after R seconds have elapsed from the infection), drop this scan.

TABLE III
SIMULATION OF SLAMMER WORM PROPAGATION WITH A HETEROGENEOUS CLUSTER MODEL

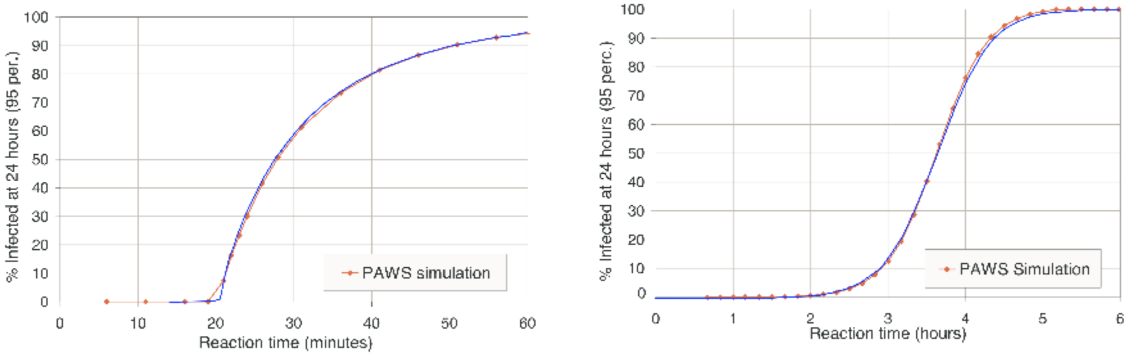
Original simulation	**PAWS simulation**
Vulnerable population is $75,000$	Vulnerable_population = $75,000$
Use BGP information from Route Views to map each Slammer infectee into the most precise prefix	Since we did not have the original IP addresses of Slammer infectees, we used the Slammer's geographical distribution presented in [12]. We obtain the AS information from the Route Views data, and retrieve the IP prefixes inside each AS. We then look up the geographical location of each AS and thus determine the number of infectees inside this AS, proportionally to the AS size and the percentage shown in [12]. Infectees inside each AS follow uniform distribution among multiple IP prefixes.
Each routed prefix has an access link, with bandwidth of $4,300$ scans per link per second	Add a worm-scan counter for each IP prefix in an AS. In function *check_routing_path()*, when checking the first AS hop for a worm scan, increase the counter for the scan sender's IP prefix. Drop the scan if the counter exceeds $4,300$. In function *worm_infectee_scan*, reset all the counters at the beginning of each simulated second.
Scale down the experiment to 1/64, by randomly selecting access links	Randomly select ASes until the number of infectees in the selected ASes reaches 1/64 of the total vulnerable population.



(a) Infected hosts I(t)

(b) Infected I(t) and quarantined R(t) hosts

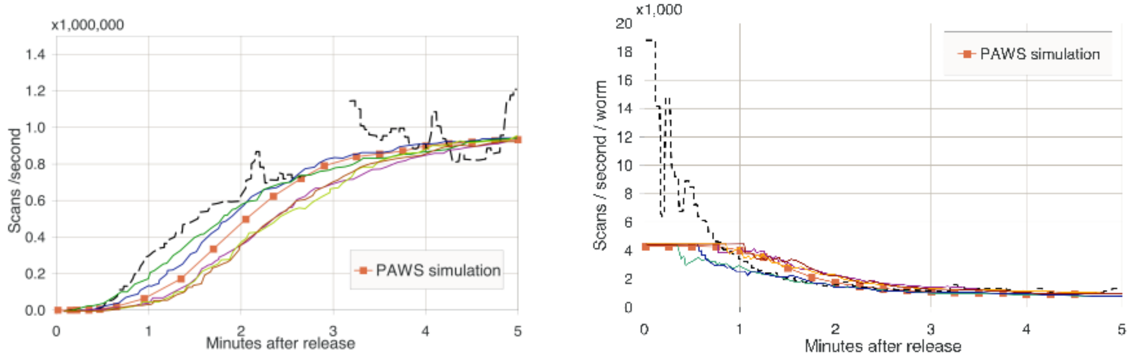
Fig. 1. Worm spread under dynamic quarantine [9]



(a) Address blacklisting

(b) Content filtering

Fig. 2. Worm spread under Internet quarantine [3]



(a) Total worm scans

(b) Scans per worm

Fig. 3. 1/64 scale simulation of Slammer worm [10] [12]

any blacklisted senders before they reach their targets. Content filtering drops all the scans that match a specific worm signature. Due to the limited space, we only present a replication of authors' 24-hour simulation of Code Red v2 worm [15] with an idealized deployment. We list the original experiment settings and describe our PAWS simulation parameters in Table II. We show our results (line with markers) in Figure 2, comparing them with the original results (line without markers) from [3]. Again, there is the close match between our and published results.

We illustrate how users can customize PAWS for any chosen address space size, network topology and worm distribution by replicating results from [10], where Weaver et al. investigated the scale-down techniques to simulate worm dynamics in a smaller address space without losing fidelity. A scaled-down worm simulation has both the vulnerable population and the address space reduced by the same factor. Among other approaches, the authors used a 1/64 scale to faithfully simulate the Slammer worm [12] within a heterogeneous network model, which is the case we replicated with PAWS. They used the Route Views data to find all the routable IP prefixes and assumed each prefix had an access link with the same limited bandwidth. They mapped each Slammer infectee into its prefix and thus achieved an empirical distribution of the vulnerable hosts. We customize PAWS to replicate their 1/64 scale experiment on the heterogeneous cluster model, and details are given in Table III. Figure 3 shows that our results (line with markers) lie between published results (straight lines without markers) showing multiple runs of Weaver et al's simulator with different random seeds. The dashed line shows the observed Slammer's dynamics in the Internet that [10] tried to match with a scaled-down model, sometimes imperfectly.

IV. Worm Emulator: WE

Worm emulation refers to the ability to introduce a worm into a network to test propagation and defense strategies in a more realistic environment than possible with a simulator. During emulation worm traffic can interact with live, legitimate traffic, and experience congestion and outages as in the real network. If a researcher is testing a defense, emulation enables her to evaluate defense's overhead, efficiency in detecting worms and collateral damage to legitimate traffic. It provides insight into the system dynamics in a controllable, predictable, and repeatable environment.

The goal for supporting worm emulation on DETER was to facilitate worm experiments that evaluate defense systems, and discover behavior that emerges from the inherent complexity of such systems when they are deployed on larger networks.

We have developed *Worm Emulator* or WE, for DETER, based upon the Metasploit [5] framework, which allows an experimenter to combine various exploit, propagation and payload modules into a single worm. We have also developed a virtualization approach for WE — the ability to emulate multiple vulnerable and infected hosts on a single physical machine.

In addition to worm generation, researchers must generate realistic legitimate cross traffic for emulation experiments. There are two techniques that can be used for this: (i) traffic replay along with content, and (ii) traffic modeling. The DETER SEER tool [16] allows the experimenter to replay full– or header–only tcpdump traces in a congestion-responsive manner, analogous to the Swing tool [17]. Unfortunately, we lacked realistic full-packet traces to input to this tool for experiments presented in this paper. SEER also has support for replaying Web request traffic from publicly available Web server logs, and can reproduce original request's contents along with the client IP address diversity. We use this replay tool in our experiments.

Traffic replay permits evaluation with realistic and diverse packets so that the experimenter can test her systems with content and address mixes typically seen on the Internet. If content realism is not critical for evaluation, SEER provides the ability to create, plan, and iterate through a large range of legitimate traffic scenarios that are realistic at the application, network and transport layers. Realism is achieved by using real client and server applications as traffic generators and driving them with request and reply features (e.g., size, interarrival times) drawn from public traffic traces. A combination of these traffic generators can be used to model an application mix on the network.

Jointly, the advantages of our worm emulation approach are: (i) validation of simulated worm propagation models with real traffic, (ii) exposing congestion-reactive cross traffic and worm defenses to realistic worm traffic, and (iii) scaling to larger topologies by multiplexing virtual hosts onto the same physical host, thus exposing worm dynamics that may not be visible in small-scale topologies.

A. Propagation Routine

Scanning refers to probing a set of IP addresses to identify vulnerable targets. There is a range of algorithms by which a worm can discover a new target to exploit, e.g., random or localized scanning, pre-generated target lists, and passive monitoring. The worm could use a combination of these algorithms for increased virulence.

Two popular forms of scanning are (a) sequential, where a worm targets addresses in a block in some predetermined order, and (b) random, where a worm targets addresses in an address block in a pseudo-random fashion. WE employs both scanning techniques. It can randomly scan the full IPv4 address range, or a specified CIDR-based address block. It can also sequentially scan addresses on the same local subnet or within a specified address range. It also supports definition of a hit-list at the start of the emulation. Worm instances first probe the addresses from this hit-list sequentially. Once they are completely infected, the worm converts to sequential or random scanning on the specified address block.

In addition to the scanning algorithm, the experimenter can select the scanning rate (the number of scans per second), which allows her to control the stealth of the emulated worm. While some worms, such as Slammer [12], scan at maximum possible rate, more sophisticated ones may employ a low scanning rate to avoid detection.

B. Exploit and Activation Routine

Once a worm finds a vulnerable target, it needs to exploits a vulnerability to penetrate onto the target and activate the next cycle of propagation. The WE emulator leverages the Metasploit framework for these mechanisms [5]. Metasploit is a platform developed for testing and launching various exploits. It allows administrators to rapidly perform penetration testing and vulnerability research by providing a collection of reusable tools, libraries, and user interfaces to configure an exploit and launch it at a target system. If the exploit succeeds, a chosen payload can be executed on the target and the user is provided with a shell to interact with the payload. The framework contains exploits for various Windows and Unix vulnerabilities, and payloads for a variety of malicious actions.

The Metasploit framework provides several user interfaces. The one we use — `msfcli` — is a command-line interface. We have programmed it to emulate a self-propagating worm which targets a given vulnerability. Our current implementation triggers the `xmlrpc` exploit, which allows injection of arbitrary PHP code into XML documents, or the `distcc` exploit, which allows execution of an arbitrary command on the target system to gain access. Our payload is the code that fetches a copy of the worm using `wget` from the host that infected this target and then activates the worm on the target. The WE emulator is implemented in Perl and the propagation and exploit activation are performed from two different threads. Thus the worm can parallelize scanning and propagation phases to maximize its efficiency.

C. Requirements for Realism

We now briefly discuss the aspects of worm emulation that have to retain high fidelity to correctly engage research defense systems. Many defenses analyze worm content and/or spread dynamics and develop techniques to separate worms from legitimate traffic. Emulated worm behavior should at the minimum reproduce realistically two worm features: (a) Rapid spread on the network from a variety of sources to a variety of destinations, with numbers of infected hosts and scans growing exponentially, [18], (b) High similarity of worm packet contents (very high for isomorphic worms and lower, but still significant for polymorphic worms) when compared to legitimate traffic's contents. We believe that our emulation approach meets these goals and we validate this in the following section.

V. Worm Emulator Experiments

To demonstrate the WE's capabilities, we first evaluate its performance limits. We then reproduce selected results from the EarlyBird worm detection system [19] to demonstrate WE's capabilities for defense testing.

WE supports physical and virtual experimentation modes. In the physical mode, each end host represents only one IP address and can harbor at most one instance of the worm. The virtual mode allows end hosts to simulate networks by hosting alias IP addresses, and multiple independent worm copies. Each alias can be independently infected and can then independently propagate the worm further.

A. Test Environment

We tested the WE emulator on a 26-node topology on DETER, with 18 nodes clustered into six networks, and connected to each other via 8 routers. The links in the network are all 100 Mbps with no added delay. All the nodes and routers in the network are dual 3.0 GHz Pentium 4 Xeon processors with with 2 GB RAM running Fedora Core 4. The operating system on 15 end hosts had vulnerable software installed on it. The worm in the experiment used the `xmlrpc` exploit that allows injection of arbitrary PHP code into XML documents.

B. WE's Performance

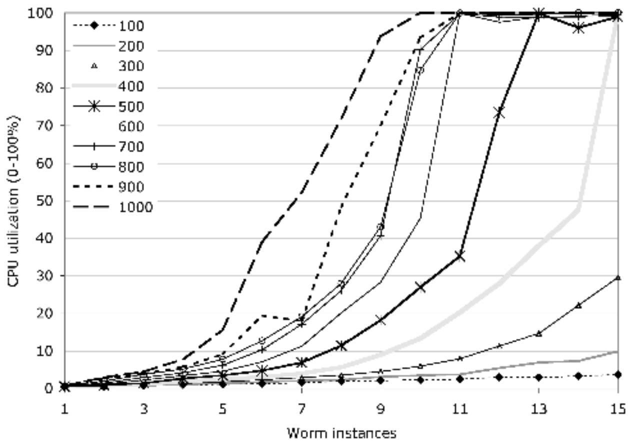


Fig. 4. CPU utilization vs worm instances for different scanning rates

The goal of worm emulation on DETER is to provide support for replication of worm spread dynamics in moderate-scale networks. Scalability is important for some worm behaviors that may not be observable at a small scale, such as the exponential shape of the propagation curves. We achieve scalability by running WE emulator in its virtual mode. Depending on the specifics of the worm behavior, such as its computation overhead and scanning rate, the physical hardware poses limits on the number of concurrent worm instances that can run on any end host, and thus the limit on our virtualization. Figure 4 shows average CPU utilization at an end host (y-axis) as the function of the scanning rate (different lines), and the number of worm instances virtualized on each host (x-axis). As the CPU utilization approaches 100%, it indicates the end host overload. In the physical mode (worm instances=1), WE can support very high scan rates (close to 1,000 scans/sec), and is bandwidth-limited at the end hosts while CPU utilization is low (< 0.64%). In the virtual mode, at low scan rates of 100 scans/second we can support about 50 worm instances on each end host before utilization goes above 60% (not shown on the graph due to scale). As the scan rate increases, the number of worm instances that can be supported reduces sharply. Thus a worm with a scan rate of 300 scans/sec, creates 30% CPU utilization when there are 15 virtualized instances on an end host.

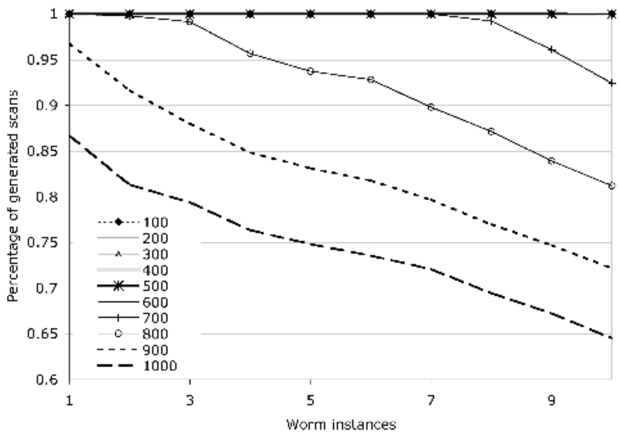


Fig. 5. Generated scans vs worm instances for different scanning rates

As additional worm instances are invoked on the end host, the desired and the actual scan rate may not match due to multiple worm copies competing for limited network resources. Figure 5 shows the ratio of the measured to the desired scan rate (y-axis) as a function of the scanning rate (different lines), and the number of worm instances multiplexed on each host (x-axis). A ratio of 100% means that all the worm instances were able to meet the desired scan rate, thus the graph indicates how many worm instances can be supported for a given scan rate. For example, a single worm instance can generate up to 900 scans/sec, but two worm instances can only send 800 scans/sec each.

C. Using WE for Defense Evaluation

This section describes experiments we performed with WE to recreate selected results from the paper describing the Earlybird worm fingerprinting algorithm [19]. The paper proposes an automatic worm defense system, which derives detection signatures for zero-day worms. The system has been evaluated in the paper using replayed, full-packet traces, as well as via deployment in a real environment. As we discussed in the Introduction none of these approaches is susceptible to reproduction by other researchers, which was our main motivation for development of WE.

We have implemented the fingerprinting algorithm from Earlybird for this experiment. The algorithm identifies packets or substrings in the network trace: (1) that have occurred many times in the payload, indicating high *content prevalence*, and (2) that have been observed in packets with many different source and destination IP addresses, indicating high *address dispersion* property. The hashes of packets or substrings that have high content prevalence and address dispersion are considered content signatures of the worm. Our current implementation omits some optimizations from [19] for speed and memory footprint, while preserving all the characteristics needed for detection.

Our defense experiments were performed on the same network as the performance experiments. Each of the 15 vulnerable end nodes hosts multiple IP addresses. We collect traffic traces for Earlybird's analysis on the central router in the topology. Since we did not have an access to realistic full-packet traces we have decided to replay Web request traffic as legitimate traffic. There are publicly available Web server logs that can be used to reproduce original request contents and client IP address diversity. Thus our experiments can be observed as replaying a subset of a legitimate traffic that would be present in a network hosting one Web server. We have developed a Web traffic generator that runs on each leaf node, reads the Web log data, and for each client IP that is hosted on this machine issues a `wget` request for the given file to the server. Our server hosts the entire file tree that appears in the Web log, with actual file sizes from the log, but with random file contents within each given file. Thus requests for the same file will result in the identical replies but there will be no content similarity across different files. In our experiment we used a Web log for one recent month of traffic, from a large academic Web server.

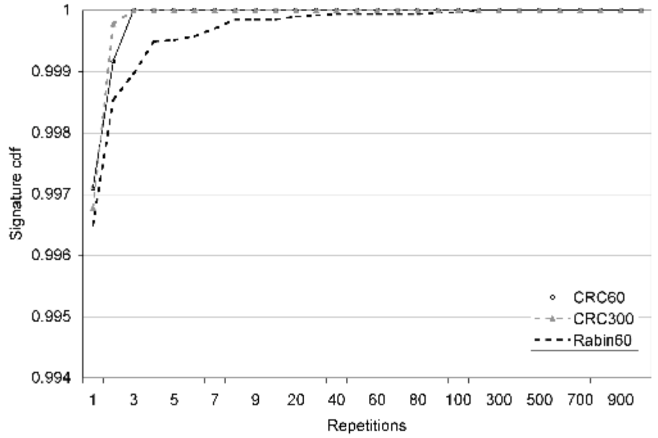


Fig. 6. CDF of content signatures

Figure 6 shows the cumulative distribution function (CDF) of content signatures for three types of hash functions. An Earlybird content signature is either a hash of the whole packet or a hash of a substring of the whole packet. The CDF is computed from the set of repetitions found in each measurement interval over a period of 10 minutes. The hash functions being investigated in [19] include the whole packet CRC hash over a 60-second and over a 300-second measurement interval, and the Rabin, 40-bytes long substring fingerprint function, over a 60-second interval. For the substring Rabin hashes, each packet of length k will produce k - 40 substring Rabin hashes. Using a 60-second measurement interval and a whole packet CRC, over 98% of all signatures repeat two or fewer times and 95% are only observed once. The shapes of all the curves are similar to those observed in the original paper, but the y-axis scale differs. The original paper had values between 0.94 and 1, while our values are between 0.996 and 1. This is to be expected since we used a different legitimate traffic trace: ours was a Web log from a small academic server while theirs was a full-packet trace from a large academic network. Our result still supports the conclusion from the original paper that a prevalence threshold of 3 is sufficient to filter over 90% of unlikely signatures.

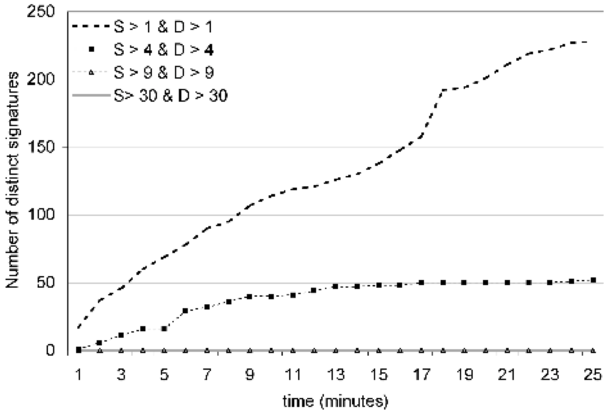


Fig. 7. Distinct signatures for different dispersion thresholds

Figure 7 shows the number of distinct signatures found over time in the legitimate traffic, for different source (S) and destination (D) dispersion thresholds. Again we observed the similar trend as in the original paper, but there was a significant scale difference resulting from the limit on our virtualization approach given the number of physical machines in the experiment. We also had 12 virtualized destinations — we hosted the Web server's files on three hosts with each having 4 IP aliases. Thus we had no Earlybird detections for S > 30 and D > 30, while [19] had a few.

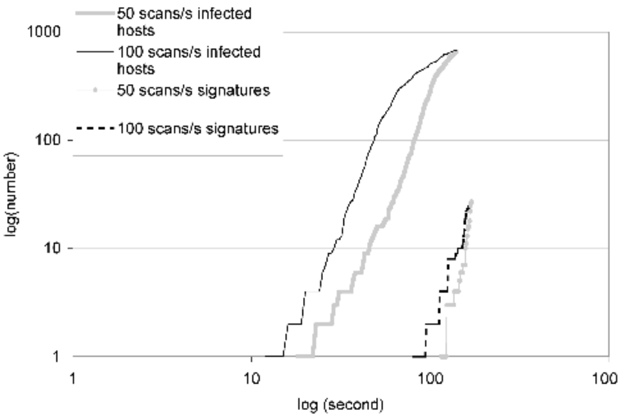


Fig. 8. Infected hosts and detected signatures during worm propagation

We next tested the Earlybird in presence of worm traffic. After running the background traffic for a while, we engaged WE to release the worm on one emulated host, and let it spread with random scanning. Figure 8 shows the number of infected hosts and detected signatures on log/log scale. We tested two scanning rates: 50 scans/s and 100 scans/s. The 15 victim hosts on the network are configured to virtualize 50 active worms each, thus the maximum number of infected hosts is 750. From the graph, one can observe the exponential grow of the infection rate, and the number of detected signatures, as expected. In this experiment, the Earlybird was configured with the prevalence threshold PV=3, and dispersion setting S $>$ 15 and D $>$ 15. Using the Rabin-40 substring fingerprint it detects the first signature of the worm after 143 (or 92) seconds when the worm scans 50 (or 100) addresses per second.

We acknowledge that our experiments with WE and Earlybird were limited because of the lack of realistic full-packet traces to replay and because of the topology size. Still, we observed the same trends as in [19]. We believe this offers a small but substantial proof that WE can be useful for moderate-scale, realistic testing of worm defenses.

VI. Related Work

A. Worm simulation

Savage et al. [20] model Internet topology at the AS level, along with the epidemiological worm spread model. Wagner et al. [21] model delay and bandwidth differences between vulnerable hosts by grouping Internet hosts into four categories and modeling the interactions both between and within categories. Liljenstam et al. [22], [23] design a worm model integrated with the SSFNet [8] simulator. This model adds simulation of countermeasures at a router level to the simple epidemiological model, and simulates those actions assuming a single router per AS. A realistic AS topology from the Route Views project [11] is used in a scaled-down form, due to single-machine memory constraints. Riley et al. [24] develop a packet-level worm simulator on GTNetS [7] to observe connection-level behaviors of TCP worm propagation. They approximate the Internet topology as a set of limited access links that connect vulnerable hosts to the "core" with unlimited bandwidth. Perumalla et al. [25] propose a large-scale packet-level simulation of worm propagation on PDNS [26]. All the above approaches deploy an overly simplified model of the Internet topology and do not simulate the interaction of the worm traffic with the background traffic, both of which lower simulation fidelity. In addition to this, all but [25] are single-node simulators, which limits their scalability to several thousand vulnerable hosts. In [25] authors simulate worm propagation among 1.28 million vulnerable nodes at a dedicated 128-CPU cluster, using PDNS and GTNetS. In another paper on distributed simulation of Internet events [27] the authors note that PDNS and GTNetS can simulate about 95K packet transmissions per wall-clock second (PTS) at a single machine and 5.5M packet transmissions on a dedicated 136-CPU cluster. This is an excellent result, given that both PDNS and GTNetS simulate traffic at a flow and packet level, and must maintain connection state for each packet transmission. We note two main problems with this approach for large-scale simulation: (1) To achieve reasonable simulation speed (5.5M PTS) one needs a powerful 100+ node cluster. Not every researcher has access to such a cluster, yet many researchers need a high-fidelity Internet simulator that can run in reasonable time on multiple common PCs, to validate their ideas. PAWS almost doubles this speed (10 PTS) with 8 PCs [4]. (2) Packet-level simulation in PDNS and GTNetS generates a network message for each packet sent to a different simulation node which incurs huge overhead when simulating high-rate worm scans. Instead, these transmissions can be aggregated and sent in a single network message at the end of a simulated time unit, like it is done in PAWS.

B. Legitimate traffic generation

Legitimate traffic generation for simulation and experiments is a well studied problem. Due to space constraints, we refer the interested reader to our previous publication [16] for a detailed discussion on the subject.

The simplest form of background traffic generation is using packet trace replay along with packet payload, e.g., tcpreplay [28]. The main drawback of a simple replay is that it is not congestion reactive and could result in different dynamics than those seen in a real network. Swing tool [17] performs congestion-responsive generation of TCP traffic according to models derived from traces. This results in realistic traffic at network and transport level, and in realistic application behavior (but not application headers). It does not result in a realistic content mix, which is often needed for testing worm defenses.

C. Malware Emulation

Sommers et al propose MACE toolkit [29] for malicious workload generation. MACE is more of an extensible framework than a fully functional tool. It provides a high-level language and a modular attack composition framework where different exploit, obfuscation, propagation and background traffic models can be specified. It remains for the user to populate these models and provide their implementation. Our WE emulator focuses only on reproducing worm traffic but the user needs to simply select propagation models, exploits and payloads from an existing implementation.

Vigna et al propose a generator of exploit mutations, which imposes network, application and content-level changes on exploit code to evade detection [30]. Our WE emulator uses existing exploits, but replicates worm payload and propagation strategy.

The Metasploit framework is the leading freeware for penetration testing on networks. The other available frameworks include Inguma [31], SecurityForest [32] and ATK [33]. Inguma and SecurityForest have an extensive exploit database, But Inguma does not have support for a graphical user interface or reporting while Metasploit does. SecurityForest tools are not as easy to use as the Metasploit framework. ATK toolkit is a small and handy tool for Windows to realize fast checks for specific vulnerabilities without extensive user interaction. Additionally, several commercial penetration testing frameworks are available but since our goal was to produce an open-source tool we could not use them within WE.

VII. Conclusions and Future Work

A worm researcher today must test her hypotheses either by writing a complex simulator from scratch, deploying a proposed system in a real network or replaying a full-packet trace captured from a real network. Writing simulators from scratch is a thankless task. It either takes a very long time to be done right or it results in naive approximations that do not match real worm spread conditions and potentially produce invalid results. On the other hand, few researchers can initiate deployment of research-grade systems in real networks, or obtain full-packet traces with privacy sensitive data.

In this paper we described our work on building worm experimentation support in the DETER testbed. We concluded from surveying current worm research that Internet-wide solutions must be evaluated with a large-scale, realistic simulation of a worm spread in the Internet environment, while localized solutions require realistic worm and legitimate traffic conditions in a local network. To support Internet-wide experiments we have developed a scalable, high-fidelity worm spread simulator, called PAWS and demonstrated through experiments that it can be easily customized to meet current and future worm researchers' needs. To support localized experimentation we have developed the WE emulator and demonstrated through experiments that this tool, together with other DETER tools for legitimate traffic generation, can reproduce realistic worm spread conditions in a local network. We believe that our support for worm experimentation in DETER may lead to faster prototyping of worm solutions, and easier, more standardized testing. The biggest advantage we hope to see is an improved realism of worm tests and leveling of worm research field. The test realism should increase through use of PAWS because of its high fidelity in reproducing worm spread events. The leveling of the playing field should occur because WE enables tests that previously could only be done by a handful of researchers who either had access to full-packet traces or could deploy their solutions in real networks.

Our future work lies in two directions. First, we hope to enlarge our database of exploits, worm payloads and legitimate traffic traces thus increasing diversity of possible tests researchers can perform in emulation framework. Second, we aim to improve the user interface for both of our tools, to ease their adoption by other researchers. These tools should help researchers understand the worm phenomenon and realistically test any defenses they build.

References

[1] T. Benzel, R. Braden, D. Kim, C. Neuman, A. Joseph, K. Sklower, R. Ostrenga, and S. Schwab, "Experience with DETER: A Testbed for Security Research," in *Proceedings of Tridentcom*, March 2006.

[2] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *Proc. of the OSDI*. Boston, MA: USENIX Association, Dec. 2002, pp. 255–270.

[3] D. Moore, C. Shannon, G. Voelker, and S. Savage, "Internet Quarantine: Requirement for Containing Self-Propagating Code," in *Proceedings of the IEEE INFOCOM*, April 2003.

[4] S. Wei and J. Mirkovic, "A Realistic Simulation of Internet-Scale Events," in *Proceedings of the VALUETOOLS*, October 2006.

[5] "The Metasploit Project," http://www.metasploit.com/.

[6] "The Network Simulator - ns-2," http://www.isi.edu/nsnam/ns/.

[7] G. I. of Technology, "The Georgia Tech Network Simulator," http://www.ece.gatech.edu/research/labs/MANIACS/GTNetS/.

[8] "Scalable Simulation Framework," http://www.ssfnet.org/homePage.html.

[9] C. C. Zou, W. Gong, and D. Towsley, "Worm Propagation Modeling and Analysis under Dynamic Quarantine Defense," in *Proceedings of ACM CCS Workshop on Rapid Malcode (WORM'03)*, October 2003.

[10] N. Weaver, I. Hamadeh, G. Kesidis, and V. Paxson, "Preliminary Results Using ScaleDown to Explore Worm Dynamics," in *Proceedings of ACM CCS Workshop on Rapid Malcode (WORM'04)*, October 2004.

[11] U. of Oregon, "Route Views Project," http://www.routeviews.org/.

[12] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the Slammer Worm," *IEEE Security and Privacy*, vol. 1(4), pp. 33–39, July/August 2003.

[13] C. Shannon and D. Moore, "The Spread of the Witty Worm," *IEEE Security and Privacy*, vol. 2(4), pp. 46–50, July 2004.

[14] N. Hu, L. E. Li, Z. M. Mao, P. Steenkiste, and J. Wang, "Locating Internet Bottlenecks: Algorithms, Measurements, and Implications," in *Proceedings of ACM SIGCOMM*, September 2004.

[15] D. Moore and C. Shannon, "The Spread of the Code-Red Worm (CRv2)," CAIDA, http://www.caida.org/research/security/code-red/coderedv2_analysis.xml.

[16] S. Schwab, B. Wilson, C. Ko, and A. Hussain, "SEER: A Security Experimentation EnviRonment for DETER," in *Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test*, 2007.

[17] K. Vishwanath and A. Vahdat, "Realistic and Responsive Network Traffic Generation," *IEEE/ACM Transactions on Networking*, 2009.

[18] S. Staniford, V. Paxson, and N. Weaver, "How to 0wn the Internet in Your Spare Time ," in *Proceedings of the 11th USENIX Security Symposium*, 2002.

[19] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated Worm Fingerprinting," in *Proceedings of the OSDI*, 2004.

[20] D. Moore, C. Shannon, G. M. Voelker, and S. Savage, "Internet Quarantine: Requirements for Containing Self-Propagating Code," in *Proceedings of IEEE INFOCOM*, vol. 3, March 2003, pp. 1901–1910.

[21] A. Wagner, T. Dubendorfer, B. Plattner, and R. Hiestand, "Experiences with worm propagation simulations," in *Proceedings of the 2003 ACM workshop on Rapid Malcode (WORM)*, 2003, pp. 34–41.

[22] M. Liljenstam, Y. Yuan, B. Premore, and D. Nicol, "A Mixed Abstraction Level Simulation Model of Large-Scale Internet Worm Infestations," in *Proceedings of the 10th IEEE MASCOTS*, 2002, p. 109.

[23] M. Liljenstam, D. M. Nicol, V. H. Berk, and R. S. Gray, "Simulating Realistic Network Worm Traffic for Worm Warning System Design and Testing," in *Proceedings of the 2003 ACM workshop on Rapid malcode (WORM)*, 2003, pp. 24–33.

[24] G. F. Riley, M. I. Sharif, and W. Lee, "Simulating Internet Worms," in *Proceedings of the IEEE MASCOTS*, 2004, pp. 268–274.

[25] K. S. Perumalla and S. Sundaragopalan, "High-Fidelity Modeling of Computer Network Worms," in *Proceedings of the 20th ACSAC*, 2004, pp. 126–135.

[26] "Parallel/Distributed NS," http://www.cc.gatech.edu/computing/compass/pdns/index.html.

[27] R. Fujimoto, K. Perumalla, A. Park, H. Wu, M. Ammar, and G. Riley, "Large-Scale Network Simulation — How Big? How Fast?" in *Proceedings of the IEEE/ACM MASCOTS*, 2003.

[28] A. Turner, "Tcpreplay tool," http://tcpreplay.synfin.net/trac/.

[29] J. Sommers, V. Yegneswaran, and P. Barford, "A Framework for Malicious Workload Generation," in *Proceedings of the ACM SIGCOMM/USENIX Internet Measurement Conference*, 2004.

[30] G. Vigna, W. Robertson, and D. Balzarotti, "Testing Network-based Intrusion Detection Signatures Using Mutant Exploits," in *Proceedings of the ACM CCS*, 2004.

[31] "Inguma Testing and Penetration Framework," http://inguma.sorceforge.net.

[32] "SecurityForest: ExploitTree," http://www.securityforest.com.

[33] "The Attack ToolKit," http://www.computec.ch/projekte/atk.