TAO: Protecting against Hitlist Worms using Transparent Address Obfuscation

Spiros Antonatos[1] and Kostas G. Anagnostakis[2]

[1] Distributed Computing Systems Group
Institute of Computer Science
Foundation for Research Technology Hellas, Greece
antonat@ics.forth.gr
[2] Internet Security Lab,
Institute for Infocomm Research
21 Heng Mui Keng Terrace, Singapore
kostas@i2r.a-star.edu.sg

Abstract. Sophisticated worms that use precomputed hitlists of vulnerable targets are especially hard to contain, since they are harder to detect, and spread at rates where even automated defenses may not be able to react in a timely fashion. Recent work has examined a proactive defense mechanism called Network Address Space Randomization (NASR) whose objective is to harden networks specifically against hitlist worms. The idea behind NASR is that hitlist information could be rendered stale if nodes are forced to frequently change their IP addresses. However, the originally proposed DHCP-based implementation may induce passive failures on hosts that change their addresses when connections are still in progress. The risk of such collateral damage also makes it harder to perform address changes at the timescales necessary for containing fast hitlist generators.
In this paper we examine an alternative approach to NASR that allows both more aggressive address changes and also eliminates the problem of connection failures, at the expense of increased implementation and deployment cost. Rather than controlling address changes through a DHCP server, we explore the design and performance of *transparent address obfuscation* (TAO). In TAO, network elements transparently change the *external* address of internal hosts, while ensuring that existing connections on previously used addresses are preserved without any adverse consequences. In this paper we present the TAO approach in more detail and examine its performance.

Keywords: worms, address space randomization, network security

1 Introduction

Worms are widely regarded to be a major security threat facing the Internet today. Incidents such as Code Red[1, 16] and Slammer[4] have clearly demonstrated that worms can infect tens of thousands of hosts in less than half an hour,

a timescale where human intervention is unlikely to be feasible. More recent research studies have estimated that worms can infect one million hosts in less than two seconds [22, 23, 24]. Unlike most of the currently known worms that spread by targeting random hosts, these extremely fast worms rely on predetermined lists of vulnerable targets, called *hitlists*, in order to spread efficiently.

The threat of worms and the speed at which they can spread have motivated research in automated worm defense mechanisms. For instance, several recent studies have focused on detecting scanning worms [27, 12, 26, 18, 21, 25]. These techniques detect scanning activity and either block or throttle further connection attempts. These techniques are unlikely to be effective against hitlist worms, given that hitlist worms do not exhibit the failed-connection feature that scan detection techniques are looking for. To improve the effectiveness of worm detection, several distributed early-warning systems have been proposed [29, 17, 30, 5]. The goal of these systems is to aggregate and analyze information on scanning or other indications of worm activity from different sites. The accuracy of these systems is improved as they have a more "global" picture of suspicious activity. However, these systems are usually slower than local detectors, as they require data collection and correlation among different sites. Thus, both reactive mechanisms and cooperative detection techniques are unlikely to be able to react to an extremely fast hitlist worm in a timely fashion.

Observing this *gap* in the worm defense space, a recent study has considered the question of whether it is possible to develop defenses *specifically* against hitlist worms, and proposed a specific technique called *network address space randomization* (NASR). This technique is primarily inspired by similar efforts for security at the host-level [28, 10, 9, 19, 13, 8]. It is also similar in principle to the "IP hopping" mechanism in the APOD architecture[7], BBN's DYNAT[14] and Sandia's DYNAT[15] systems, all three designed to confuse targeted attacks by dynamically changing network addresses. In its simplest form, NASR can be implemented by adapting dynamic network address allocation services such as DHCP[11] to *force* more frequent address changes.

The major drawback of the DHCP-based implementation of NASR as presented in [6] is the damage caused in terms of aborted connections. The damage depends on how frequently the address changes occur, whether hosts have active connections that are terminated and whether the applications can recover from the transient connectivity problems caused by an address change. Although the results of [6] suggest that the failure rates are small when measured in comparison to the total number of unaffected connections, the failures may cause significant disruption to specific services that users value a lot more than other connections, such as long-lived remote terminal session (e.g., ssh), etc. Furthermore, the acceptable operating range of DHCP-based NASR does not fully cover the likely spectrum of hitlist generation strategies. In particular, there are likely scenarios that involve very fast, distributed hitlist generation, which cannot be thwarted without extremely aggressive address changes. Aggressive address changes in the DHCP-based NASR implementation have a profound effect on connection failure rates, and the approach hereby becomes less attractive.

As an alternative to the DHCP-based implementation of NASR, in this paper we consider a different approach that allows both more aggressive address changes and also eliminates the problem of connection failures, at the expense of increased implementation and deployment cost. Rather than controlling address changes through a DHCP server, we explore the design and performance of *transparent address obfuscation* (TAO). In TAO, we assume that hosts of a subnet are located behind a network element that transparently changes the *external* addresses of the hosts, while ensuring that existing connections on previously used addresses are preserved without any adverse consequences.

In the rest of this paper, we first discuss in more detail how network address space randomization works generally, and then discuss how transparent address obfuscation can be implemented, and how well it performs.

2 Network Address Space Randomization

The goal of network address space randomization (NASR) as originally proposed in [6] is to force hosts to change their IP addresses frequently enough so that the information gathered in hitlists is rendered stale by the time the worm is unleashed. The authors of [6] have demonstrated that NASR can slow down the worm outbreak, in terms of the time to reach 90% infection, from 5 minutes when no NASR is used to between 24 and 32 minutes when hosts change their addresses very frequently. Their results are based on simulations, varying how fast the hitlist is generated and how fast the host addresses are changed. It appears that the mean time between address changes needs to be 3-5 times less than the time needed to generate the hitlist for the approach to reach around 80% of its maximum effectiveness, while more frequent address changes give diminishing returns. The assumption of global deployment of NASR is unreasonable, thus it is more likely that only a fraction of subnets will employ the mechanism, such as dynamic address pools. NASR continues to be effective in slowing down the worm, even when deployed in 20% or 40% of the network.

The authors of [6] have proposed to implement NASR by configuring the DHCP server to expire DHCP leases at intervals suitable for effective randomization. The DHCP server would normally allow a host to renew the lease if the host issues a request before the lease expires. Thus, forcing addresses changes even when a host requests to renew the lease before it expires requires some minor modifications to the DHCP server. This approach does not require any modifications to the protocol or the client. In their implementation, three timers on the DHCP server for controlling host addresses were used. The *refresh* timer determines the duration of the lease communicated to the client. The client is forced to query the server when the timer expires. The server may or may not decide to renew the lease using the same address. The *soft-change* timer is used internally by the server to specify the interval between address changes, assuming that the flow monitor does not report any activity for the host. A third, *hard-change* timer is used to specify the maximum time that a host is allowed to keep the same address. If this timer expires, the host is forced to change address,

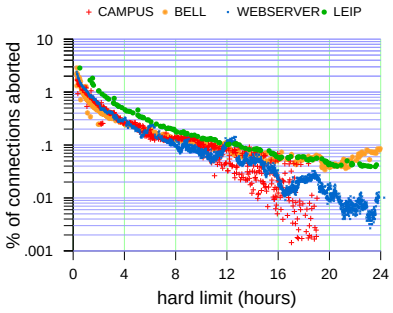

Fig. 1. Percentage of aborted connections as a function of the hard change limit

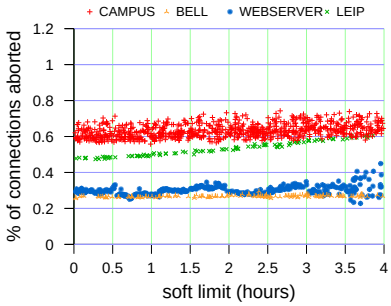

Fig. 2. Percentage of aborted connections as a function of the soft change limit

as the DHCP server does not renew the lease, despite the damage that may be caused.

The main drawback of this approach is the damage caused in terms of aborted connections. The damage depends on how frequently the address changes occur, whether hosts have active connections that are terminated and whether the applications can recover from the transient connectivity problems caused by an address change. As shown in Figures 1 and 2, the damage varies from 0.01 to 5%. Experiments were done using traces collected at different network environments: a one-week contiguous IP header trace collected at Bell Labs research[2], a 5-day trace from the University of Leipzig[3], a 1-day trace from a local University Campus, and a 20-day trace from a link serving a single Web server at the institute of the authors.

However, as we need to perform randomization in small timescales, where the failure rates wave between 3 and 5%, failure rates may not be acceptable. We can avoid network failures by using *Transparent Address Obfuscation*, an approach which needs more deployment resources than the standard NASR implementation. We describe the *Transparent Address Obfuscation* in the following section.

3 Transparent Address Obfuscation

The damage caused by network address space randomization (NASR) in terms of aborted connections may not be acceptable in some cases. Terminating, for example, a large web transfer or an SSH session would be both irritating and frustrating. Additionally, it would possibly increase network traffic as users or applications may repeat the aborted transfer or try to reconnect. To address these issues, we suggest `Transparent Address Obfuscation`, an external mechanism for deploying NASR avoiding connection failures.

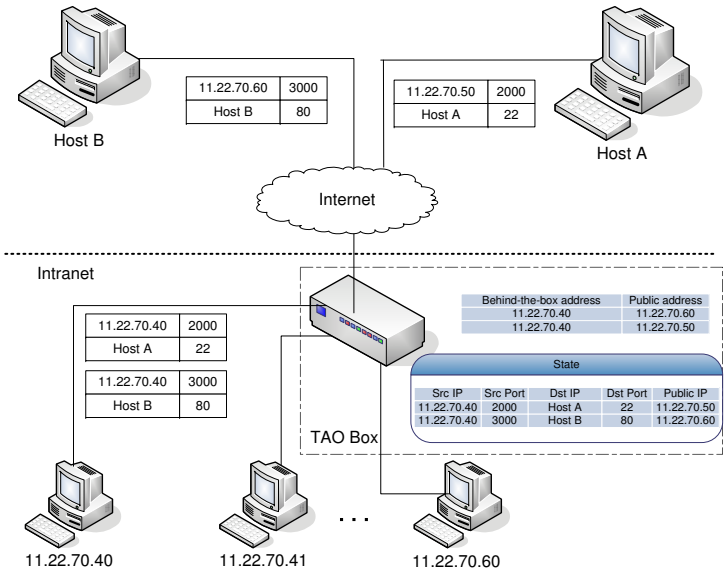

Fig. 3. An advanced example of NASR using the TAO box. Host has two public IP addresses, one (11.22.70.50) devoted for the SSH session to Host A and the other (11.22.70.60) for new connections, such as a HTTP connection to Host B

The idea behind the mechanism is the existence of an "address randomization box", called from now on "TAO box", inside the LAN environment. This box performs the randomization on behalf of the end hosts, without the need of any modifications to the DHCP behavior, as suggested in [6]. TAO box controls all traffic passing by the subnet(s) it is responsible for, analogous to the firewall or NAT concept. The address used for communication between the host and the box remains the same. We should note that there is no need for private addresses, unlike the case of NAT, as end hosts can obtain any address from the organization they belong. The public address of the end host – that is the IP that outside world sees – changes periodically according to soft and hard timers, similar to the procedure described in [6]. Old connections continue to operate over the old address, the one that host had before the change, until they are terminated.

The TAO box is responsible for two things. First, to prevent new connections on the old addresses (before randomization) reaching the host. Second, to perform address translation to the packets based on which connection they belong, similar to the NAT case. Until all old connections are terminated, a host would require multiple addresses to be allocated.

An example of how the TAO box works is illustrated in Figure 3. The box is

responsible for address randomization on the 11.22.70.0/24 subnet, that is it can pick up addresses only from this subnet. Initially the host has the IP address 11.22.70.40 and TAO box sets the public IP address of this host to 11.22.70.50. The host starts a new SSH connection to Host A and sends packets with its own IP address (11.22.70.40). The box translates the source IP address and replaces it with the public one, setting it to 11.22.70.50. Simultaneously, the box keeps state that the connection from port 2000 to Host A on port 22 belongs to the host with behind-the-box address 11.22.70.40 and public address 11.22.70.50. Thus, on the Host A side we see packets coming from 11.22.70.50. When Host A responds back to 11.22.70.50, box has to perform the reverse translation. Consulting its state, it sees that this connection was initiated by host 11.22.70.40 so it rewrites the destination IP address.

After an interval, the public address of host 11.22.70.40 changes. TAO box now sets its public address to 11.22.70.60. Any connections initiated by external hosts can reach the host through this new public IP address. As it can be seen in Figure 3 the new connection to Host B website has the new public IP as source. Note that in the behind-the-box and public address mapping table host now has two entries, with the top being chosen for new connections. The only connection permitted to communicate with the host at 11.22.70.50 address is the SSH connection from Host A. For each incoming packet, the box checks its state to find an entry. If no entry is found, then packet is not forwarded to the internal hosts, else the "src IP" field of the state is used to forward the packet. As long as the SSH connection lasts, the 11.22.70.50 IP will be bound to the particular host and cannot be assigned to any other internal host. When SSH session finishes, the address will be released. For stateless transport protocols, like UDP or ICMP, only the latest mapping between public and behind-the-box IP address is used.

4 Simulation Study

The drawback of the TAO box is the extra address space required for keeping alive old connections. An excessive requirement of address space would empty the address pool, making the box abort connections. We tried to quantify the amount of extra space needed by simulating the TAO box on top of four traffic traces. The first two traces, `CAMPUS` and `CAMPUS(2)`, come from a local university campus and include traffic from 760 and 1675 hosts respectively. All hosts of this trace belong to a /16 subnet. The second trace, `BELL`, is a one-week contiguous IP header trace collected at Bell Labs research with 395 hosts located in a /16 subnet. Finally, the `WEBSERVER` trace is a 20-day trace from a link serving a single Web server at our institute. In this trace, we have only one host and we assume it is the only host in a /24 subnet. In our simulation, the soft timer had a constant value of 90 seconds, while the hard timer varied from 15 minutes to 24 hours.

The results of the simulation are presented in Figure 4. In almost all cases, we need 1% more address space in order to keep alive the old connections. We

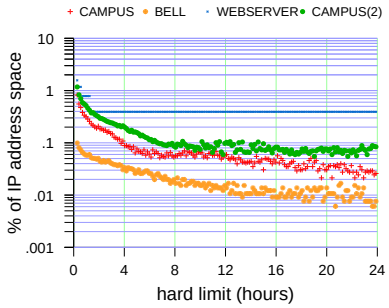

Fig. 4. The percentage of extra IP space needed

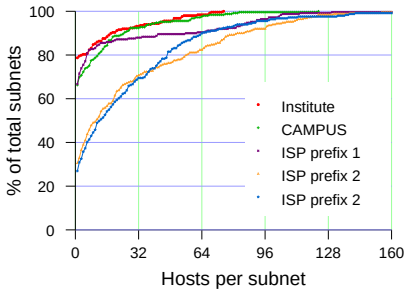

Fig. 5. Subnet address space utilization

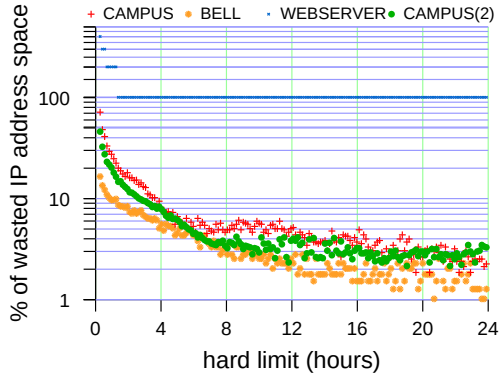

Fig. 6. The percentage of extra IP space needed relative to the load of subnets

measured the number of hosts that are alive in several subnets. We used full TCP scans to identify the number of hosts that were alive in 5 subnets: our local institute, a local University campus and three subnets of a local ISP. Our results, as shown at Figure 5, indicate that 95% of the subnets are less than half-loaded and thus we can safely assume that this 1% of extra space is not an obstacle in the operation of the TAO box. However, the little extra address space needed derives from the fact that subnets are lightly loaded. For example, the 760 hosts of the `CAMPUS` trace correspond to the 1.15% of the /16 address space. In Figure 6, the relative results of the previous simulation are shown. On average, 10% more address space for hard timer over one hour is needed, which seems a reasonable overhead. In the case of the `WEBSERVER` trace the percentage is 100% but this is expected as we have only one host.

5 Related Work

Our work on network address space randomization was inspired by similar techniques for randomization performed at the OS level [28, 10, 9, 19, 13, 8]. The general principle in randomization schemes is that attacks can be disrupted by reducing the knowledge that the attacker has about the system. For instance, instruction set randomization[13] changes the instruction set opcodes used on each host, so that an attacker cannot inject compiled code using the standard instruction set opcodes. Similarly, address obfuscation[9] changes the locations of functions in a host's address space so that buffer-overflow exploits cannot predict the addresses of the functions they would like to utilize for hijacking control of the system. Our work at the network level is similar, as it reduces the ability of the attacker to build accurate hitlists of vulnerable hosts.

The use of IP address changes as a mechanism to defend against attacks was proposed independently in [7], [14] and [15]. Although these mechanisms are similar to ours, there are several important differences in the threat model as well as the way they are implemented. The main difference is that they focus on targeted attacks, performing address changes to confuse attackers during reconnaissance and planning. Neither project discusses or analyzes the use of such a mechanism for defending against worm attacks.

Reference [20] proposes the use of honeypots with instrumented versions of software services to be protected, coupled with an automated patch-generation facility. This allows for quick (*i.e.,* less than 1 minute) fixing of buffer overflow vulnerabilities, even against zero-day worms. However, that work is limited to worms that use buffer overflows as an infection vector.

While some of these reactive defense proposals may be able to detect the worm, it is unclear whether they can effectively do so in the timescales of hitlist worm propagation.

6 Summary and Concluding Remarks

Fast-spreading malware such as hitlist worms represent a major threat for the Internet, as most reactive defenses currently being investigated are unlikely to be fast enough to respond to such worms in a timely fashion. Recent work on network address space randomization has shown that hitlist worms can be significantly slowed down and exposed to detection if hosts are forced to change their address frequently enough to make the hitlists stale. However, the implications of changing addresses in a DHCP-based implementation, as proposed in [6] hamper the adoption of this defense, as it can cause disruption under normal operation and cannot be performed fast enough to contain advanced hitlist generation strategies.

The approach examined in this paper, Transparent Address Obfuscation (TAO), offers more leeway for administrators to more frequently change addresses, while at the same time eliminating the problem of "collateral damage" in terms of failed connections, when compared to the DHCP-based implementation. The experiments presented in this paper demonstrated that the cost of

TAO in terms of additional address space utilization is modest and that the operation of the system is transparent and straightforward.

Acknowledgments

This work was supported in part by the projects CyberScope, EAR and Miltiades, funded by the Greek General Secretariat for Research and Technology under contract numbers PENED 03ED440, USA-022 and 05NON-EU-109 respectively. We also thank Sotiris Ioannidis for his "constructive" comments on an earlier draft of this paper.

References

1. CERT Advisory CA-2001-19: 'Code Red' Worm Exploiting Buffer Overflow in IIS Indexing Service DLL. http://www.cert.org/advisories/CA-2001-19.html, July 2001.
2. NLANR-PMA Traffic Archive: Bell Labs-I trace. http://pma.nlanr.net/Traces/Traces/long/bell/1, 2002.
3. NLANR-PMA Traffic Archive: Leipzig-I trace. http://pma.nlanr.net/Traces/Traces/long/leip/1, 2002.
4. The Spread of the Sapphire/Slammer Worm. http://www.silicondefense.com/research/worms/slammer.php, February 2003.
5. K. G. Anagnostakis, M. B. Greenwald, S. Ioannidis, A. D. Keromytis, and D. Li. A Cooperative Immunization System for an Untrusting Internet. In *Proceedings of the 11th IEEE Internation Conference on Networking (ICON)*, pages 403–408, September/October 2003.
6. Spiros Antonatos, Periklis Akritidis, Evangelos P. Markatos, and Kostas G. Anagnostakis. Defending against Hitlist Worms using Network Address Space Randomization. In *Proceedings of the 3rd ACM Workshop on Rapid Malcode (WORM)*, November 2005.
7. Michael Atighetchi, Partha Pal, Franklin Webber, Rick Schantz, and Chris Jones. Adaptive use of network-centric mechanisms in cyber-defense. In *Proceedings of the 6th IEEE International Symposium on Object-oriented Real-time Distributed Computing*, May 2003.
8. Elena Gabriela Barrantes, David H. Ackley, Trek S. Palmer, Darko Stefanovic, and Dino Dai Zovi. Randomized instruction set emulation to disrupt binary code injection attacks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, October 2003.
9. S. Bhatkar, D. DuVarney, and R. Sekar. Address obfuscation: An efficient approach to combat a broad range of memory error exploits. In *In Proceedings of the 12th USENIX Security Symposium*, pages 105–120, August 2003.
10. Jeffrey S. Chase, Henry M. Levy, Michael J. Feeley, and Edward D. Lazowska. Sharing and protection in a single-address-space operating system. *ACM Transactions on Computer Systems*, 12(4):271–307, 1994.
11. R. Droms. Dynamic Host Configuration Protocol. RFC 2131, http://www.rfc-editor.org/, March 1997.

12. J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast Portscan Detection Using Sequential Hypothesis Testing. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2004.
13. Gaurav S. Kc, Angelos D. Keromytis, and Vassilis Prevelakis. Countering Code-Injection Attacks With Instruction-Set Randomization . In *Proceedings of the ACM Computer and Communications Security Conference (CCS)*, pages 272–280, October 2003.
14. Dorene Kewley, John Lowry, Russ Fink, and Mike Dean. Dynamic approaches to thwart adversary intelligence gathering. In *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX)*, 2001.
15. John Michalski, Carrie Price, Eric Stanton, Erik Lee Chua, Kuan Seah, Wong Yip Heng, and Tan Chung Pheng. Final Report for the Network Security Mechanisms Utilizing Network Address Translation LDRD Project. Technical Report SAND2002-3613, Sandia National Laboratories, November 2002.
16. D. Moore, C. Shannon, and J. Brown. Code-Red: a case study on the spread and victims of an Internet worm. In *Proceedings of the 2nd Internet Measurement Workshop (IMW)*, pages 273–284, November 2002.
17. D. Nojiri, J. Rowe, and K. Levitt. Cooperative response strategies for large scale attack mitigation. In *Proceedings of the 3rd DARPA Information Survivability Conference and Exposition (DISCEX)*, April 2003.
18. S. E. Schechter, J. Jung, and A. W. Berger. Fast Detection of Scanning Worm Infections. In *Proceedings of the* 7^{th} *International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 59–81, October 2004.
19. Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, and Dan Boneh. On the effectiveness of address-space randomization. In *CCS '04: Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 298–307, New York, NY, USA, 2004. ACM Press.
20. S. Sidiroglou and A. D. Keromytis. A Network Worm Vaccine Architecture. In *Proceedings of the IEEE Workshop on Enterprise Technologies: Infrastructure for Collaborative Enterprises (WETICE), Workshop on Enterprise Security*, pages 220–225, June 2003.
21. S. Staniford. Containment of Scanning Worms in Enterprise Networks. *Journal of Computer Security*, 2004.
22. S. Staniford, D. Moore, V. Paxson, and N. Weaver. The top speed of flash worms. In *Proc. ACM CCS WORM*, October 2004.
23. S. Staniford, V. Paxson, and N. Weaver. How to Own the Internet in Your Spare Time. In *Proceedings of the 11th USENIX Security Symposium*, pages 149–167, August 2002.
24. N. Weaver and V. Paxson. A worst-case worm. In *Proc. Third Annual Workshop on Economics and Information Security (WEIS'04)*, May 2004.
25. N. Weaver, S. Staniford, and V. Paxson. Very Fast Containment of Scanning Worms. In *Proceedings of the* 13^{th} *USENIX Security Symposium*, pages 29–44, August 2004.
26. M. Williamson. Throttling Viruses: Restricting Propagation to Defeat Malicious Mobile Code. Technical Report HPL-2002-172, HP Laboratories Bristol, 2002.
27. Jian Wu, Sarma Vangala, Lixin Gao, and Kevin Kwiat. An Effective Architecture and Algorithm for Detecting Worms with Various Scan Techniques. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, pages 143–156, February 2004.

28. J. Xu, Z. Kalbarczyk, and R. Iyer. Transparent runtime randomization for security. In *A. Fantechi, editor, Proc. 22nd Symp. on Reliable Distributed Systems –SRDS 2003*, pages 260–269, October 2003.
29. Vinod Yegneswaran, Paul Barford, and Somesh Jha. Global Intrusion Detection in the DOMINO Overlay System. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, February 2004.
30. C. C. Zou, L. Gao, W. Gong, and D. Towsley. Monitoring and Early Warning for Internet Worms. In *Proceedings of the* 10^{th} *ACM International Conference on Computer and Communications Security (CCS)*, pages 190–199, October 2003.