Petrifying Worm Cultures: Scalable Detection and Immunization in Untrusted Environments

Joel O. Sandin
Department of Computer Science
Stanford University
Stanford, CA
Email: jsandin@cs.stanford.edu

Bilal Khan
Department of Mathematics and Computer Science
John Jay College of Criminal Justice
New York, NY
Email: bkhan@jjay.cuny.edu

***Abstract*— We present and evaluate the design of a new and comprehensive solution for automated worm detection and immunization. The system engages a peer-to-peer network of untrusted machines on the Internet to detect new worms and facilitate rapid preventative response. We evaluate the efficacy and scalability of the proposed system through large-scale simulations and assessments of a functional real-world prototype. We find that the system enjoys scalablity in terms of network coverage, fault-tolerance, security, and maintainability. It proves effective against new worms, and supports collaboration among among mutually mistrusting parties.**

I. Introduction

Since the late 90s, computer worms have attacked the consumer community with alarming regularity, e.g. Melissa (1999), Code Red (2001), Slammer (2003), Blaster (2003), Sasser (2004), etc. Although the economic impact of these attacks already exceeds billions of dollars, we still have no proven antidote against emergent worms and remain vulnerable to the dangers they pose.

Network worms spread by using the Internet to access services with exploitable vulnerabilities. Newly "infected" hosts serve as a stepping stone, advancing the infection exponentially and potentially leading to thousands of vulnerable hosts becoming compromised in a very short time.

Since worms are largely static in their propagation strategies, an attack against a vulnerable host typically follows a predictable pattern or *signature*. Modern intrusion detection systems (IDS) such as Bro [1] and Snort [2] leverage this fact: by matching the ports and byte sequences of incoming traffic to specific signatures, they can identify worm traffic as it arrives, and prevent the vulnerable services from seeing virulent packets.

While signatures-based IDSes are useful against known threats, they remain impotent against new worms because well-designed network worms can propagate much more quickly than signatures can be generated. Most commercial products rely on hand-generated signatures, a labor-intensive process that takes on the order of hours or days. In constrast, modern worms spread exponentially fast (e.g. the Slammer worm [3] doubled its number of infections every 8.5 seconds and infected more than 90 percent of vulnerable hosts within 10 minutes).

A comprehensive solution must detect new worms and rapidly provide an active response without human intervention.

II. Prior and Related Work

A number of systems for detecting and responding to worm threats have been proposed in the literature.

Detection systems follow a model similar to [4] which uses a "network telescope" – large, unallocated blocks of IP addresses – to capture scan traffic and thus detect worms quickly. However, passive detection allows only the crudest form of active response—all clients must be denied access to the vulnerable service until it has been secured. In addition, collaborative detectors of this form require total trust among all participants.

Throttling solutions use local network anomaly detection to identify infected machines, and react by throttling and isolating misbehaving hosts to control the spread of worms without affecting the traffic of uninfected machines [5]. Paxton et. al. [6] give a complete and practical system that uses anomaly detection to prevent infected machines from infecting other hosts on the local network. Since they work by controlling infections at the *source*, however, the effectiveness of these schemes relies of wide-scale deployment.

Content filtering solutions, in contrast, stop infections at the *destination*. Signature systems such as EarlyBird [7] and Autograph [8] observe flows from infected machines, identify blocks common across many of those flows, and dynamically create IDS signatures to enable content filtering. However, in order to generate signatures a source of malicious flows is required. Autograph [8] presented a solution to the problem of obtaining malicious flows rapidly, but their scheme requires a large deployment and total trust among the participants. The Honeycomb project [9] collects malicious flows through medium interaction honeypots to facilitate automatic signature generation, but does not address the problem of rapid worm detection among mutually mistrusting participants.

Active response honeypot-based systems such as Autopatching [10] and Vigilante [11], use instrumented honeypots to detect buffer overflows and develop deployable fixes for previously unknown vulnerabilities. These systems allow participants to *verify* the existence of a threat locally, thus minimizing the trust in the system. However, they specifically

target overflow-based vulnerabilities, and may fail to detect worms that spread via other attack vectors.

The Collapsar honeyfarm system [12] uses a centralized collection of virtual honeypots for cost-effective monitoring and event correlation. The Potemkin honeyfarm [13] uses a modified Xen [14] to run hundreds of high-interaction honeypots on a single physical server, thus making honeypot deployment significantly cheaper. These systems do not consider automatic worm detection, and are designed to operate within a single administrative domain.

The idea of collaborative intrusion detection is not new. Yegneswaran, et. al. make a strong case for collaborative intrusion detection with The DOMINO System [15], by demonstrating that blacklists can be constructed much more quickly using a large-scale IDS. The Worminator system [16] uses shared alerts from distributed IDS elements to detect an attack in progress. However, the veracity of shared alerts is difficult to verify in such schemes; participants are required to trust one another.

III. Architectural Overview

There are many challenges in developing a scalable and comprehensive solution. We must be able to detect worms early before they become widespread. A system that provides an active response must be accurate: there should be no false negatives (no worms should go undetected) and no false positives (since network messages misdiagnosed as worms can cause denied service). Automatically generated countermeasures must be fully tested and guaranteed to be effective. The proposed technique must be general, not only in catching known worms, but also new worms that may be engineered to evade or subvert the system. The system must be practical, reliable, easily deployed and maintained.

In addition to these challenges, a serious obstacle is *adoption*: a detector is not effective unless it is deployed in the first place. Network worms are fast enough to overrun any detector deployed in a single administrative domain, thus making the case for a distributed, collaborative detector; however, collaboration that requires a high degree of trust among participants is unlikely to be deployed, hence our goal to maximize coverage in the network, while minimizing trust.

Our system works as follows: a set of *sensor* machines and *honeypot* machines are (voluntarily) deployed throughout the Internet. The sensors divert all unsolicited traffic by tunnelling it to one or more honeypots. If a worm successfully infects a honeypot, it will attempt to make outgoing infection attempts to propagate itself. The host honeypot, however, tunnels all outgoing attacks *towards other honeypots*. Once many honeypots have been infected, a signature can be developed, and the people administering these honeypots learn of the platform and services affected by the worm and the signatures developed. Our system is designed to be transparently distributed across multiple networks, allowing participants everywhere to join. A machine can join the system by becoming a sensor or a honeypot. In contrast to prior efforts, participants in our system share actual infections (rather than just sharing information) which can be locally verified using honeypots. The system maximizes participation by minimizing trust required among the participants.

Conceptually, the honeypot community acts as a "petri dish" for worms. The distribution of sensors ensures that this petri dish is a microcosm reflecting worm demographics in the Internet at large. The relatively small size of the petri dish intensifies the rate at which infections percolate, allowing automated antidote generation to become feasible well in advance of the time when the infection disables the ambient larger network.

> If you see this in the map of my microcosm, follows it that I am known well enough too? – Coriolanus (Shakespeare's *The Tragedy of Coriolanus*)

IV. Design

We now describe the components and functions of the system in more detail.

Sensors are machines that are configured to send "unwanted" traffic to a honeypot. Unwanted traffic is any traffic that would otherwise be dropped by the sensor, or anything specific that sensor is configured to ignore. Examples include unsolicited connections (TCP SYN packets), UDP packets to unused ports, incoming ICMP requests, and so on, all of which are sent to a honeypot by means of an SSL tunnel. Sensors are cheap since existing machines and firewalls at the front of the network can be configured to route unwanted traffic for many IP addresses to distinct honeypots.

Honeypots are machines with known vulnerabilities deployed in the network to learn about attackers' motivations and techniques. In our system, participants are free to deploy honeypots in any configuration, but for detection of new worms we expect that honeypots will be up-to-date on patches and thus only exploitable via previously unknown vulnerabilities. Specifically, participants will deploy honeypots for services they are interested in protecting. Each honeypot needs an associated manager process that is responsible for communicating with other hosts via the overlay network, setting up tunnels with sensors and other managers, recording outbound traffic, and generating signatures when infection occurs. If the honeypot is implemented using virtualization technology like VMWare [17] or Xen [18], then it can reside on the same physical machine as the manager. Though deploying honeypots is more expensive than deploying sensors, honeypots allow the contributor to observe infections on a machine they control and thus obtain conclusive proof of a threat.

Interconnectivity. When a sensor/honeypot joins the system, it builds tunnels to honeypots by contacting its manager and negotiating the details of the tunnel interface (e.g. private IP addresses, routes, IP address of the honeypot, traffic policies, encryption). We assume honeypots are heterogeneous and leave it to the people deploying the sensors/honeypots to decide what kind of "personality" they will have. Our system uses the Chord [19] distributed hash table lookup primitive for

sensors/honeypots to locate honeypots on a particular platform or hosting a particular service.

Routing into honeypots. By manipulating iptables rules we DNAT the address on unwanted traffic for the sensor towards the address of the honeypot, forwarding the packet over the tunnel to the honeypot's manager. Many current worms (e.g. Blaster and Sasser) require the infected host to make fresh TCP connections to the infector to retrieve the executable that comprises the bulk of the worm's logic. By configuring the sensor like a masquerading firewall for the honeypot, the sensor can proxy some connections from the honeypot and allow it to communicate with the infector.

Routing out of honeypots. It is obvious to the honeypot manager when a honeypot has been infected: suddenly the largely inactive honeypot begins making many outgoing connections per second to random IP addresses. To spread the infection, within the petri culture, these infection attempts are re-routed to other honeypots. Each outbound infection attempt (originally destined for a random IP address) is DNATed instead to a randomly chosen honeypot over a tunnel established by the corresponding honeypot managers. The target honeypot manager acts as a masquerading firewall for the infector, and source NATs traffic from itself so that it appears to be coming from target.

Active Response. Once the infection has spread among the honeypots, each associated manager has undeniable, locally observable evidence of a threat. This information is easy to trust because it has been observed using resources that the manager controls. In addition, the honeypot manager has observed many details about the infection, including the content of any traffic the worm might generate on the network. The particular choice of active response mechanism is orthogonal to the design of our system; previous work has showed that one can generate a signature [8] by taking many observed infections, breaking the associated traffic flows into pieces, and identifying subsegments which occur with high frequency across multiple flows. In our system a honeypot manager queries a (small) random set of peer honeypots, retrieves their observed flows, and uses these to attempt generation of a signature. Even if some number of participants are malicious, they cannot significantly influence the signature generation. Once a signature has been generated, its effectiveness can be tested using the live copy of the worm that has been captured. To deploy the signature, the honeypot manager simply needs to augment the IDSes it controls. While sensors lack the resources to observe an infection locally, they can each periodically query a random set of honeypot monitors, retrieve observed flows to learn the nature of the threat, and generate a signature using the same techniques as the honeypot monitors.

V. Key Features

Open, peer-to-peer with low trust assumptions. Our approach uses a simple, peer-to-peer algorithm that can detect a worm outbreak in its early stage by harnessing computer cycles donated by individuals and corporations that may not necessarily trust each other. Given the tremendous damages caused by worm outbreaks in recent years and the inevitability of further attacks, we believe many will heed to the call for contribution to worm detection, provided that the technique is safe, effective, non-intrusive, and easy to deploy.

Accurate diagnosis. The best way to diagnose worms is to watch them in action. By "culturing" the worms within the honeypot network, this approach gives conclusive evidence of a threat to all members of the system. This approach does not generate any false negatives or positives, because it will catch the worm if and only if it is infectious.

Automatic generation of fully-tested responses. We automatically generate the signature using the method described by Kim and Karp [8]. More importantly, our honeypots allow us to test the signature immediately by seeing if it guards the infectious message sent to other honeypots. We can verify that the signature works if it stops the worm from propagating.

Fast distribution of responses. The speed of worm propagation is harnessed to a healthy end: rapid dissemination of news of an infection. Any individual or organization owning a honeypot can directly observe the effect of the worm as well as the effectiveness of the generated signature to halt the flow. Those who own only sensors can still benefit by querying their peers to learn about the threat and response reliably.

Prevention of False-Positives. We believe that the use of honeypots in our system is an effective safeguard against false positives since honeypots infecting other honeypots constitutes conclusive evidence of a threat. Since our system provides *local verification* of a worm, malicious participants cannot fool other members of the system. Users who don't have honeypots can trust that there is a threat only if it is confirmed by a majority of randomly selected honeypots.

Thwarting Malicious Participants. As our system relies on collaboration, we need to guard against both malfunctioning components and compromised or malicious participants (e.g. sensors malfunctioning if they fall victim to a worm attack). Compromised components may deliberately lie to other participants but signature generation is not susceptible to this unless a majority of participants are malicious. Participants who join solely to learn more about the system can only acquire limited knowledge of sensors/honeypot locations since each participant only knows a small a number of other participants in the overlay.

VI. Simulation Design and Experiments

We have written a discrete-time event simulator to measure the efficacy of the *culturing* approach to worm detection under various parameters. The simulation considers a network consisting of S sensors, H honeypots, and O ordinary machines. Each sensor (resp. honeypot) knows only a small set of `FEED_HP` (resp. `PEER_HP`) honeypots. Each of these $S + H + O$ machines is "vulnerable" to the worm with a probability `VULN_PROB`. Initially, a set of `INITIAL_POP` worms are instantiated on distinct randomly chosen machines. Every `SCAN_PERIOD` seconds, each worm hurls itself towards an IP address which is taken to be in the set of

FEED_HP	2
PEER_HP	8
VULN_PROB	0.99
SCAN_PERIOD	0.05
INITIAL_POP	1000

TABLE I

SIMULATION PARAMETERS

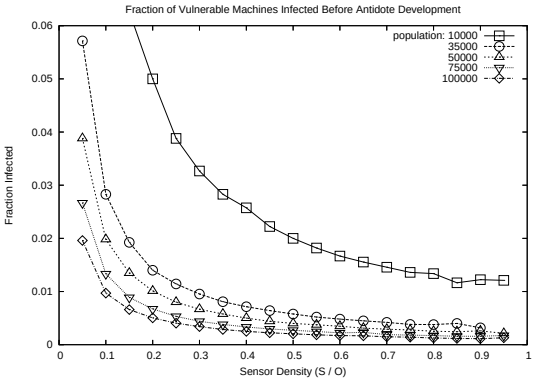

Fig. 1. Complete Simulation Results

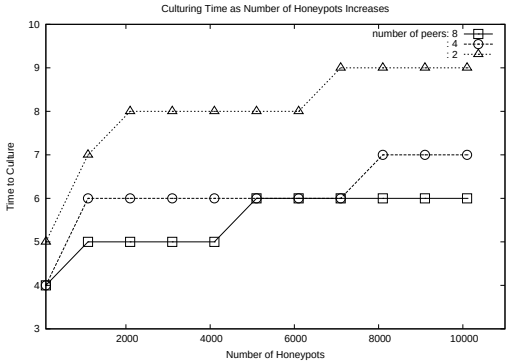

Fig. 2. Culturing Simulation Results

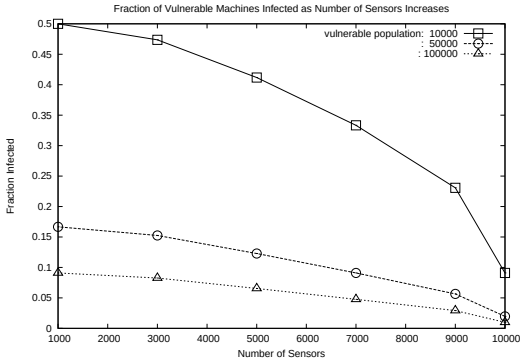

Fig. 3. Sensor Simulation Results

sensor/honeypot/ordinary machines with probability $(S+O+M)/2^{32}$. If the source of the worm is a honeypot or sensor, then tunnelling is simulated by taking the destination to be a randomly chosen honeypot. If the source of the worm is an ordinary machine, then the destination is simply taken to be a random machine. Finally, if the destination of the transmission happens to be both vulnerable and uninfected, a new copy of the worm is instantiated there. While the parameters are adjustable in our simulation environment, we fix them at the realistic values listed in Table I; changing these values does not alter the qualitative aspects of our conclusions.

Figures 1 and 2 illustrate how altering the density of sensors influences the time it takes the honeypot community to generate an antidote.

In Figure 1, we consider a system of a fixed number of $H = 1000$ honeypots. We vary the sensor density from 5% to 95% and plot the extent to which the infection has spread by the time all honeypots have acquired the antidote. For example, if 25% of the 10^4 node network are sensors, then less than 1% of the machines have been infected by the time all honeypots have developed an antidote. We repeat this experiment for networks having 10^4, $3.5 \cdot 10^5$, $5 \cdot 10^5$, $7.5 \cdot 10^5$ and 10^5 nodes and show each scenario as a separate curve.

In Figure 2, we consider a system where the number of honeypots varies from 100 to 100000, and plot the time, measured from the first infection, that is needed to completely culture the worm among the honeypots under varying number of honeypot peers. In this simulation, each honeypot spreads infections *only* to adjacent peers. For example, in a system with 8000 honeypots, with each honeypot spreading infections to 8 peers, the total time to culture the infection among the honeypots is 9 seconds.

In Figure 3, we fix the number of honeypots, and measure the sensitivity of the system to varying number of sensors, under fixed vulnerable population sizes of 10^5, $5 \cdot 10^5$, and 10^6. Each graph plots the response time of the system as the number of sensors increases.

A. Analysis Conclusions

Our analysis allows us to answer three questions: What is the behavior of the system as the ratio of sensors to vulnerable machines increases? How does performance of culturing decrease as the number of honeypots increases? And finally, what is the response time of the system for various population sizes as the number of sensors increases?

Figure 1 shows the asymptotic behavior of the system as sensor density approaches one in the system. A density of 30 percent catches an initial infection quickly enough to culture the worm among the honeypots before more than 1 percent of vulnerable machines have been infected. For larger population sizes – those greater than 35000 machines – a sensor density greater than 50 percent yields little return in terms of response time of the system. 30% seems prohibitively large; however, since sensors can be deployed using one physical router to monitor large blocks of unused IP addresses, we believe the cost is acceptable.

Figure 3 confirms our expectation that response time decreases *linearly* as the number of sensors increases; this is expected, as each scan has a constant probability of hitting a single monitored IP address.

PEER_HP	4
SENSORS	30000
HONEYPOTS	1000

TABLE II
ACCEPTABLE VALUES

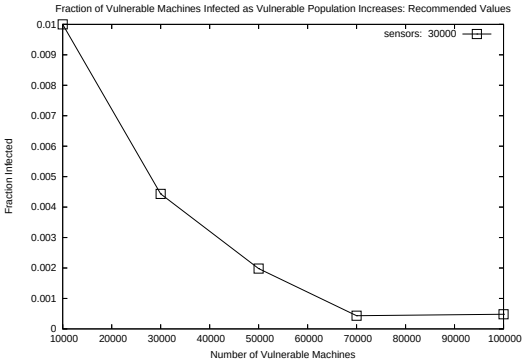

Fig. 4. Simulation Results Under Acceptable Values

Figure 2 shows the overhead of culturing worms as number of honeypot participants grows is quite low, even with a small infection out-degree among participants. Clearly, culturing leverages the epidemic properties of worms to spread news of the threat quickly among all participants.

Interestingly, smaller vulnerable populations demand extra vigilance; a larger number of sensors is required to capture an initial infection fast enough to beat the epidemic growth among the vulnerable machines.

From our analysis, we can extrapolate some recommended values for a concrete deployment of the system. We choose a honeypot population of 1000, but the simulations show that we can culture efficiently for larger populations. Our sensor population is set conservatively at 30000; this ensures the population of infected machines remains small – less than 1 percent in the worst case – while still allowing time for full culturing among the honeypots.

VII. PROTOTYPE DESIGN AND DEPLOYMENT

We have obtained some preliminary experiences with a prototype implementation of our system. Our prototype we developed supports multiple sensors forwarding traffic to a chain of managers running honeypots. All components can reside in different networks, and components communicate using GRE tunneling. Our honeypots are implemented using virtual machines under VMWare. Cleaning up after an infection is easy; we just shut down the virtual machines and they revert to a clean state. VMWare also supports the suspension of virtual machines, allowing us to save captured worms and attempt to reinfect our system after signatures have been deployed. To support active response we use the tcpflow tool to reassemble flows, and use Rabin fingerprints to break flows into non-overlapping blocks, as described in [8]. Once all observed flows have been decomposed and broken into chunks, we compute a count of the number of times a block has been seen among distinct flows, and as described in the paper, choose the most frequently occuring blocks as likely signatures. Once we have a signature, we use the snort-inline [20] tool to interpose on all network traffic, and statefully inspect and kill flows if they contain specified content.

To test our system, we deployed 8 Windows XP honeypots and deployed 1 sensor on a DSL connection. Our goal was to immunize our system against a worm taken from the wild. The local network was very active, and the worms that we targeted could be caught in under a minute in most cases.

For example, we caught several variants of the Blaster (2003), a worm which exploits a vulnerability in Microsoft's DCOM RPC interface in Windows 2000/XP. We observed the worm making several types of connections as it spread. Blaster uses a buffer overflow to execute a small piece of shell code, which then downloads the rest of the worm code from the infecting host via TFTP. After observing multiple connections, our automated signature generator created signatures for the worm. When used with Snort-inline [20], they proved effective in preventing uninfected honeypots from being infected, something we could test immediately using a live copy of the worm. In comparing our signatures to their published equivalents, we also noted a significant overlap. These results show our system is capable of generating effective signatures.

REFERENCES

[1] V. Paxson, "Bro: a system for detecting network intruders in real-time," *Computer Networks (Amsterdam, Netherlands: 1999)*, vol. 31, no. 23–24, pp. 2435–2463, 1999. [Online]. Available: citeseer.ist.psu.edu/article/paxson98bro.html

[2] T. S. Project, "Snort, the open-source network intrusion detection system." [Online]. Available: www.snort.org

[3] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the slammer worm," *IEEE Security and Privacy*, vol. 1, no. 4, pp. 33–39, 2003.

[4] C. C. Zou, L. Gao, W. Gong, and D. Towsley, "Monitoring and early warning for internet worms," in *Proceedings of the 10th ACM conference on Computer and communications security*. ACM Press, 2003, pp. 190–199.

[5] C. C. Zou, W. Gong, and D. Towsley, "Worm propagation modeling and analysis under dynamic quarantine defense," in *Proceedings of the 2003 ACM workshop on Rapid Malcode*. ACM Press, 2003, pp. 51–60.

[6] N. Weaver, S. Staniford, and V. Paxson, "Very fast containment of scanning worms," in *Proceedings of the 13th USENIX Security Symposium*, 2004. [Online]. Available: www.icsi.berkeley.edu/~nweaver/containment/containment.pdf

[7] S. Singh, "Automated worm fingerprinting," 2004. [Online]. Available: citeseer.ist.psu.edu/singh04automated.html

[8] H.-A. Kim and B. Karp, "Autograph: Toward automated, distributed worm signature detection," in *Proceedings of the 13th Usenix Security Symposium (Security 2004)*, 2004. [Online]. Available: www-2.cs.cmu.edu/~bkarp/autograph-usenixsec2004.pdf

[9] C. Kreibich and J. Crowcroft, "Honeycomb: creating intrusion detection signatures using honeypots," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 1, pp. 51–56, 2004.

[10] S. Sidiroglou and A. Keromytis, "Countering network worms through automatic patch generation," 2003. [Online]. Available: citeseer.ist.psu.edu/article/sidiroglou03countering.html

[11] M. Costa, "Vigilante: End-to-end containment of internet worms," 2005. [Online]. Available: citeseer.ist.psu.edu/costa05vigilante.html

[12] X. Jiang and D. Xu, "Collapsar: A vm-based architecture for network attack detention center," 2004. [Online]. Available: citeseer.ist.psu.edu/jiang04collapsar.html

[13] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker, and S. Savage, "Scalability, fidelity, and containment in the potemkin virtual honeyfarm." [Online]. Available: citeseer.ist.psu.edu/vrable05scalability.html

[14] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer, "Xen and the art of virtualization," in *Proceedings of the ACM Symposium on Operating Systems Principles*, 2003. [Online]. Available: citeseer.ist.psu.edu/dragovic03xen.html

[15] V. Yegneswaran, P. Barford, and S. Jha, "Global intrusion detection in the domino overlay system," 2004. [Online]. Available: citeseer.ist.psu.edu/yegneswaran04global.html

[16] M. L. et al., "Towards collaborative security and P2P intrusion detection." [Online]. Available: citeseer.ist.psu.edu/locasto05towards.html

[17] V. Inc., "Vmware workstation 4.5.2 user's manual." [Online]. Available: vmware-svca.www.conxion.com/software/ws45_manual.pdf

[18] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer, "Xen and the art of virtualization," in *Proceedings of the ACM Symposium on Operating Systems Principles*, October 2003. [Online]. Available: citeseer.ist.psu.edu/dragovic03xen.html

[19] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable Peer-To-Peer lookup service for internet applications," in *Proceedings of the 2001 ACM SIGCOMM Conference*, 2001, pp. 149–160. [Online]. Available: citeseer.ist.psu.edu/stoica01chord.html

[20] W. Metcalf. [Online]. Available: snort-inline.sourceforge.net