Using the Strombringer System Tool Suite to Test for Vulnerabilities in a University Research and Development Autonomous System

Dmitri Podkorytov	Dennis Guster	Jake Soenneker
Kurgan State University	St. Cloud State University	St. Cloud State University
Kurgan, Russia	St. Cloud, MN 56301	St. Cloud, MN 56301
podkorytov@mail.ru	dguster@stcloudstate.edu	soja0704@stcloudstate.edu

Abstract

This paper will analyze a new tools set recently devised by the primary author of this proposal. The goal of this analysis will be twofold. First, to ascertain if any of the tools pose a major risk if used by hackers compared to existing tools. Second, to determine how these tools might be used proactively by system administration to proactively check their autonomous systems for vulnerabilities. It was found that the tools are somewhat analogous to NMAP, but function primarily on the data link layer (OSI layer 2). The analysis revealed that this tools set does offers functionality beyond already existing tools primarily due to its layer 2 orientation. It was found that the tool set could be used quite effectively to check for vulnerabilities in regard to denial of service attacks (DOS). Further, the test scenarios revealed an interesting vulnerability in regard to virtual zones. Specifically, the interfaces in virtual zones often have null hardware address (all 0's) which it make it very difficult to trace a DOS attack back to a physical host so the problem can be rectified. This paper just scratched the surface in regard to using these tools for proactive testing. The authors believe further analysis is warranted.

Introduction and Purpose

The landscape in computing unfortunately is marred with instance of attacks. For example, the Computer Crime Research Organization reports that hacker attacks grew 37% in the first quarter of 2004 (Keefe, 2004). Holt and Kilger, 2008 report that things have gotten worse and the frequency and sophistication of computer attacks have increased in the last decade as have reports concerning the involvement of organized crime and state sponsored groups in hacker attacks. There are many different types of attack methods used, but many of the methods deal with manipulating/modifying the incoming or outgoing packet structure in some way. Commonly available utilities such as NCAT (the networking swiss army knife) (Netcat User's Guide, 2009) and HPING (Whitman and Mattord, 2012) are widely used in such endeavors. Further there a number of proven vulnerabilities associated with the improper use of each of these tools (Stanger, 2009; Symantec, 2005). Of course there have been numerous other utilities that have been developed along these same lines and have proven even more dangerous (Sectools.org, 2006).

Given that these tools are in wide spread use it is important for system administrators to realize it is not a matter if their system will be attacked, but rather when and at what intensity. Therefore, the system administrator needs a means to proactively test the vulnerability of their systems against these tools. This of course means that the administrator needs to think like a hacker and use the tool to launch the type of attacks that might be expected from a hacker and then learn from the attacks and remediate the problem.

This paper will analyze a new tools set recently devised by the primary author of this proposal. The goal of this analysis will be twofold. First, to ascertain if any of the tools pose a major risk if used by hackers compared to existing tools. Second, to determine how these tools might be used proactively by system administration to proactively check their autonomous systems for vulnerabilities. Once again, the analysis will also address whether this tools set offers functionality beyond already existing tools.

The Importance of Proactive Testing

The literature indicates that proactive testing can be an integral part of an autonomous system's security strategy and can be quite successful. In perhaps the most common type of attack, denial of service on the network level, Ye, Shi and Ye, 2009 found that through proactive testing of TCP/IP header information they can identify and isolate denial of service attacks. While their methodology was successful the amount of time and resources required was significant. So given this large investment in resources is it effective to pursue proactive testing? Varian, 2004 points out that in most cases the risk is so high that a company can't afford not to implement a proactive testing program as part of their regular monitoring cycle. This logic can be imbedded in the consequences of failure. Keep in mind that 93% of businesses that experience a major security disaster

never recover and go out of business (National Archives, 2008). In fact, the National Institute of Standards and Technology supports proactive testing and has included a chapter in their published guidelines on vulnerability testing (Scarfone, Souppaya, Cody and Orebaugh, 2008).

Implementing proactive testing as part of the monitoring cycle can be quite attractive because once implemented it automates the process. However, because of the dynamic nature of attacks, testing methods will need to be regularly updated. Further, testing procedures will need to be aligned with existing policy and for this alignment to be successful a formal architecture needs to be devised (Strembeck, 2005). The work of Kotenko and Bogdanov, 2009 takes this concept further and creates an operational model for this architecture. This resulting model provides sound structure for a viable proactive security scanner.

While strong well thought out policy is a critical component it is also important to recognize that the attack methodology will change as hackers realize the protection mechanisms are becoming effective. As stated earlier a prime reason to utilize proactive testing is to discover vulnerabilities and correct them before they become highly exploited by the attackers. Once again dedicating the resources to accomplish this on an operational level is prohibitive. Therefore, methodologies that can to some extent automate the updating process are very attractive. This problem has been addressed by the algorithmic design community and it appears that machine learning may offer solution. While not every denial of service attack is the same often they share some basic commonalities such as a rapid spike in workload. The work of Suresh and Anitha (2011) illustrates how machine learning might be employed to more effectively manage DDoS attacks. Specifically, they use the chi-square and Information gain feature selection mechanisms for selecting important attributes. Once the attributes are selected various machine learning models, like Navies Bayes, C4.5, SVM, KNN, K-means and Fuzzy c-means clustering are tested to ascertain their efficiency in detecting DDoS attacks. There experimental results determined that Fuzzy c-means clustering provides better accuracy in the identification of the attacks.

Because the majority of data is being carried on digital networks the danger extends beyond just traditional “computer data” it is not unusual for voice traffic to be carried on vulnerable networks in the form of voice over IP (Shevtekar and Ansari, 2006). As one might expect the same proactive test concepts also apply to wireless application as well. Epstein, 2009 states that service level assurance is the category of networking where service levels are actively measured by proactively injecting traffic into live networks. This allows constant testing of real, live networks, with traffic that represents the applications that mean the most for that network. This is especially critical in a wireless world because wireless networks can and often change in ways wired networks don’t. Therefore this proactive approach is even more crucial in the wireless world.

In summary, it is clear that the potential damage that today’s dynamic attack strategies could inflict is staggering. It is therefore necessary to have a monitoring methodology that is both current and automated linked to sound policy. In meeting that goal it is a good

idea to explore the various system monitoring tools to ascertain whether they provide a potential advantage to either the attacker or the defender.

Description of Strombringer

The Strombringer system tools (named after an infamous black sword in the work of Michael Moorcock) in many ways is similar to familiar tools such as ncat (Netcat, 2009) and nmap (Nmap, 2012) but focuses on the layer 2 (MAC) address structure (Podkorytov, 2012). Strombringer is designed only for test purposes and consists of several individual tools for generating Ethernet packets which can be used to proactively probe networks. A brief description of the major tool will follow.

There are two tools designed to allow the basic transmission of packets. First, *psend-rb* is a lightweight tool that allows reverse broadcasts with a fixed packet size of 100 bytes. It broadcasts packets across networks via its defined MAC source address (Ethernet hardware address). Second, the *psend2* offers a higher degree of sophistication and both the source and destination MAC address can be controlled. Other option include intensity parameters, packet counters, spoofing other MAC addresses on the network, generating random content within the packets and setting the packet size.

There are also several commands that allow for basic monitoring on the data link level. First, *listen* allows a user to just listen and display the MAC addresses of transmitting interfaces within the network. Second, *e-pin*g allows and Ethernet level ping that will display all of the MAC source addresses of senders within the network. Third, *nfork* is a utility that allows a tester to determine how many processes can be forked within a very intense utility called *pstorm*. Fourth, *e-stat* prints statistics about an interface (useful for determining intensity).Last, there are several prewritten scripts such as *Strombringer.sh* which makes it easy to set up a scenario for running *pstorm* and *ps-mcast.sh* which makes it easy to set an a multicasting environment.

Annotated Examples of Strombringer

Perhaps the eaiest of the utility programs to understand is *e-ping*. In the example below we are sending packets on the lo (local or loopback interface) to MAC address 00:00:00:00:00:00 from MAC address 00:00:00:00:00:00 (note MAC addresses are 48 bits represented by 12 hex characters). Running the command produces a to/from entry for each packet. Running the often used packet sniffing program *tcpdump* produces very interesting results. Note that all that appears is a very truncated packet. In fact, other than the time stamp all that appears is the Ethernet (data link layer information. Further from the dump it does not appear to be the traditional Ethernet type style frame either which would encompass 14 bytes (header). Rather only 8 bytes (zeroed out) appear which may indicate that it is using the SubNetwork Access Protocol (SNAP). SNAP is often used for encapsulating IP datagrams and ARP requests and is designed to function on IEEE 802 networks. Therefore, the footprint of this packet is minimal. Trying to filter out this type

of packet based on OSI layer three logic would be difficult. Certainly it is a case that would highly depended on a policy such as: all packets not explicitly accepted should be dropped. Using a deny logic would not be appropriate using the OSI layer 4 (TCP) reset flag because that layer is not implemented within the packet.

```
Admin2@mis481:~/Strombringer/Strombringer-src#./e-ping lo 00 00 00 00 00 00  00 00 00 00
00 00
To:  0  0  0  0  0  0   From:  0  0  0  0  0  0
To:  0  0  0  0  0  0   From:  0  0  0  0  0  0

root@Strombringer:~# tcpdump -e -n -c 125 -vvv -XX -i lo
tcpdump: listening on lo, link-type EN10MB (Ethernet), capture size 65535 bytes
11:40:44.572599 [|ether]
        0x0000:  0000 0000 0000 0000                      ........
11:40:44.572645 [|ether]
        0x0000:  0000 0000 0000 0000                      ........
11:40:44.572652 [|ether]
        0x0000:  0000 0000 0000 0000                      ........
```

Further, using the *e-ping* utility was the only case in which the authors were able to trap packets on the host interface level using *tcpdump.* However, packets were trapped on a network level that were directed to other hosts within the autonomous system. It first it was thought that this might be due to the fact the test host was a virtual zone with virtual network interfaces. In the example below the host mis481 has two virtual interfaces virnet0 and virnet0:0 (logical interface 0 of the main virtual interface 0) in both cases there is no real MAC address and the address appear as all zeros. In the case of the host: *storage* there is an actual 48 bit address represented by 12 hex numbers: 72:e6:6d:72:f8:5c. Even with that real MAC address packets could not be trapped with *tcpdump* on the eth0 interface level.

```
[admin2@mis481 ~]# ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0

virnet0    Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:127.0.0.1  P-t-P:127.0.0.1  Bcast:0.0.0.0  Mask:255.255.255.255

virnet0:0  Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:10.11.6.27  P-t-P:10.14.8.17  Bcast:10.11.6.27  Mask:255.255.255.255
          UP BROADCAST POINTOPOINT RUNNING NOARP  MTU:1500  Metric:1

.
storage@ubuntu-stor:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 72:e6:6d:72:f8:5c
          inet addr:10.10.27.101  Bcast:10.255.255.255  Mask:255.0.0.0
          Interrupt:15

lo        Link encap:Local Loopback
          inet6 addr: ::1/128 Scope:Host
```

However, it was possible to see the traffic using the *e-stat* tool with in Strombringer on virtual interface 0. The number of packets received and dropped would raise quit significantly when packet storms were unleashed.

```
Admin2@storage:~/Strombringer/Strombringer-src# ./e-stat virnet0
```

```
->

  ps_recv   ps_drop  ps_ifdrop

 00000112  00000003  00000000  Speed=0000002 p/sec 000000248 bytes/sec ^C
```

Also, it is possible to see a trace of the packets as they are transmitted across an interface with the e-listen tool. In the example below a random pattern is being transmitted from the source address virnet0

```
Admin2@storage:~/Strombringer/Strombringer-src# ./e-listen virnet0

 0  7  FF  FF  0  0     <-  00  00  00  00  00  00
 0  e  FF  FF  0  0     <-  00  00  00  00  00  00
 0  4  FF  FF  0  0     <-  00  00  00  00  00  00
 0  9  FF  FF  0  0     <-  00  00  00  00  00  00
 0  c  FF  FF  0  0     <-  00  00  00  00  00  00
```

In the example below the transmission characteristics of the p-send2 utility are shown. In this case virtual interface 0:0 is used to broadcast packets to a real mac address 7E EE 93 EB B1 C2. Note that there are numerous options to control packet intensity and packet size.

```
Admin2@storage:~/Strombringer/Strombringer-src# ./psend2 venet0:0 adr 00 00 00 00 00 00
addr 7E EE 93 EB B1 C2  P10 trace

parse_opt P10 parse_opt trace
 Run with options:
 simm:Generate From->To and To->From two ways traffic
 slow:Generate with 1 sec delay in packet generation loop
 rand_delay:Generate with random delay (from 0 to 16 seconds ) in main loop
 infinity:allways working
 trace:log activity
 no_send:no send packets
 show_addr:show address of sending
 show_counter:show packet counter
 P1:send only one  packet
 P10:send 10  packets

Run 000000000:000000001
FF:FF:FF:FF:FF:FF->7E:EE:93:EB:B1:C2
7E:EE:93:EB:B1:C2->FF:FF:FF:FF:FF:FF

Run 000000000:000000001
FF:FF:FF:FF:FF:FF->7E:EE:93:EB:B1:C2
7E:EE:93:EB:B1:C2->FF:FF:FF:FF:FF:FF
```

This example illustrates how prewritten scripts can be used. In this example the e-storm tools is executed from the Strombringer shell. In several attempts this execution string was able to easily lock up the virtual terminal used, perhaps because of the load it placed on the network interface level.

```
root@Strombringer:~/Strombringer/Strombringer-src# ./scripts/Strombringer.sh venet0:0
Run on $1 interface
kernel.pid_max = 5000
venet0:0  Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
```

```
sudo: ./e-storm:
To:  0  0  0  0  0  0   From:  0  0  0  0  0  0
To:  0  0  0  0  0  0   From:  0  0  0  0  0  0
To:  0  0  0  0  0  0   From:  0  0  0  0  0  0 ^C
```

The last example illustrates a utility that can be used to probe system capacities: *n-fork* which allows the number of processes that can be (theoretically safely) forked within the *p-storm* utility. The value returned with the virtual system utilized was 1048. Based on observations with the limited resources of the virtual zone a (perhaps unintentional) denial service appeared to occur before the 1048 thresh hold was reached.

```
Admin2@storage:~/Strombringer/Strombringer-src# ./nfork

 1048
```

The packets that appeared on the network within the autonomous system as a result of running the p-storm tool once again provided limited information because they were crafted to be transmitted on the data link level. These packets used an LLC (Ethernet II) frame type and were 60 bytes in length and because the MAC addresses were all zeros probably originated from a virtual interface. The packet inter-arrival rate is fairly intense at 0.00212 seconds. The packet was viewed by the system level monitoring tools as an error, specifically a malformed packet. The contents of the dump indicate that it is basically an empty packet because it contains null values (hex 0s). Generating packets of this type is quick/easy and facilitates proactive testing. However, the structure of this packet type can be viewed as dangerous because it is very difficult to relate it back to its source. This is especially true in a large autonomous system that widely uses virtualization and virtual interfaces because they all might have MAC address of 00:00:00:00:00:00.

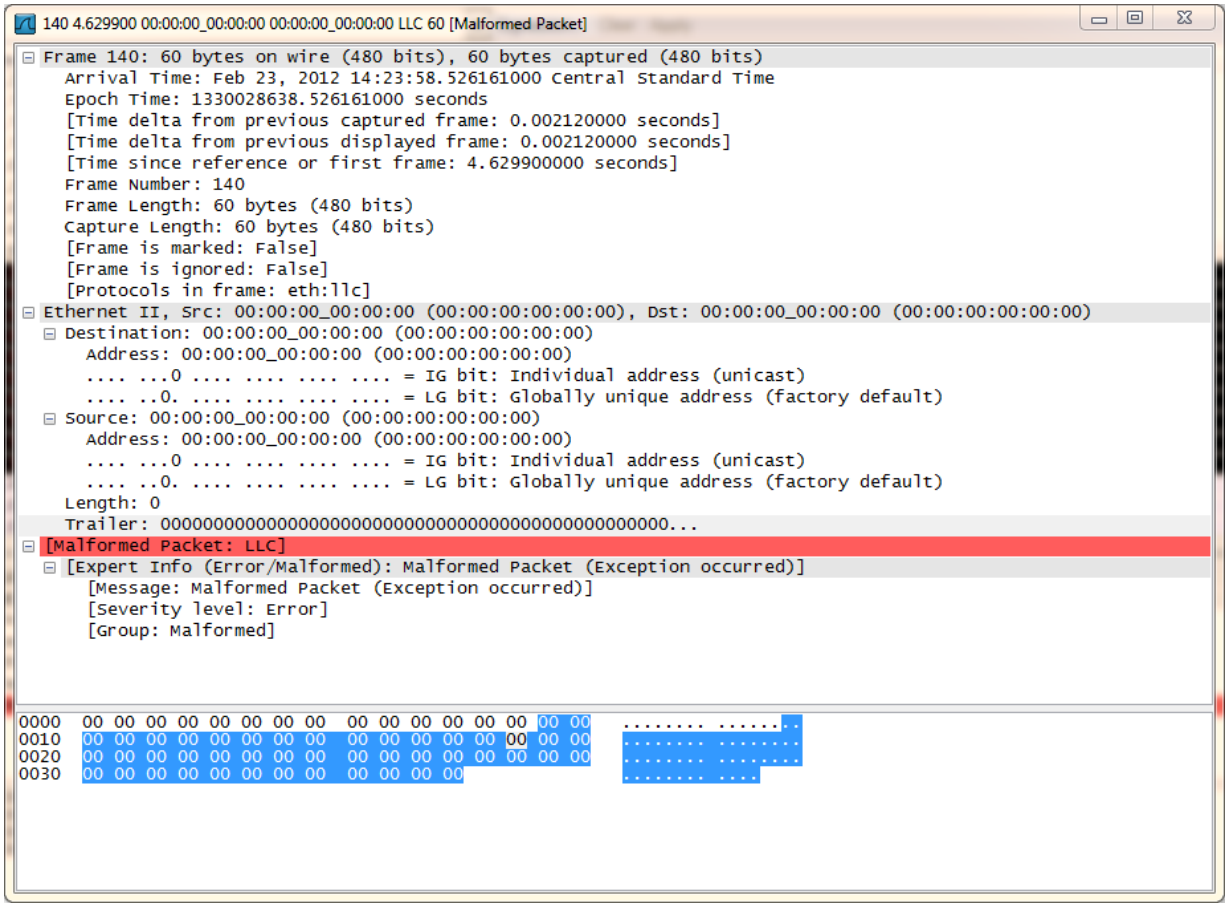

Figure 1: A Strombringer packet being recognized as malformed.

Discussion and Conclusions

The authors went into this endeavor with the attitude that Strombringer would be similar in function to existing tools such as *nmap*. However, this tool set while similar in the sense that one can probe a network is distinctly different. That difference lies in the fact that it is based on MAC rather than IP addresses. The first thought that came to mind is that many system administers are trained primarily on the logical IP (network) address level. In most cases the address resolution protocol (ARP) is enabled and the physical addressing layer is almost transparent to system administrators. While some system administrator may run static ARP tables that they have to configure and maintain that is rare except on highly secure systems. The fact these packets contain no layer 3 information could be problematic for some system administrator, especially if they try to trace them back to their source.

The fact that this testing was undertaken using virtual zones added some interesting twists. First, the virtual interface with their all zero MAC addresses made it difficult to track packets once the packets left that virtual host. One would think that the packets

would obtain the actual MAC address of the physical card within the physical host, but the packets trapped do not support that logic. This could be explained to some degree if one assumes that a NIC (network interface card) is actually a special purpose processor and one could speculate that these tools to some degree may reprogram the basic functions of that NIC. Second, running this software within virtual zones did provide some protection to the autonomous system, however, the limited resources of the virtual zone were easily overwhelmed by the packet storm tool. It appears that although the host itself was taxed the actual denial of service came from locking up the interface that the virtual terminal program would use to connect to that virtual zone. Fortunately, the storm process appeared to time out once the terminal session was lost as a result of closure of the bash shell. So without rebooting the user could reconnect to the host about 30 minutes later and there was no evidence (at least by running the *ps* command) that the packet storm processes were still staged in memory.

As far as a tool to proactively test the autonomous system for vulnerabilities in the classical sense of the internet world based on layer 3 addressing the logic is different. With IP some of your addressing information is readily available via DNS and the rest can often by determined by using a tool such as *nmap*. With Strombringer which is MAC based a potential hacker would need to compromise a host at the root level and install it. If the hacker has already gained root access to a host then they might not bother with the type of MAC layer attacks generated by Strombringer and rely on classical methods. However, if subtlety is desired the intensity controls and being able to launch attacks on the unexpected MAC level might be attractive to a hacker. In some cases MAC address are used as a secondary means of authentication and should be kept secret. If a hacker was able to configure Strombringer within an autonomous system then it could easily be used to collect all of the MAC address within that autonomous system.

In summary, it appears that the main value of this analysis is gaining a better understanding of how layer 2 attacks might be launched and testing these tools provides an idea of the dangers those MAC attacks might pose. While proactively using the tools to probe for vulnerabilities has value perhaps an equally important aspect may be the educational value of these tools. Having a system administrator install and proactively experiment with the tools could make them aware of the nature of layer 2 attacks. Further, the subtleties of how layer 2 attacks might be launched on virtual systems/virtual networks where MAC address are zeros could pose a problem from the uninformed system administrator. This software could allow that scenario to be safely simulated and serve as a training platform for system administrators to experiment with a means to trace layer 2 packets coming from virtual interfaces to their physical source.

REFERENCES

CERT. *Password File Protection*, http://www.cert.org/tech_tips/passwd_file_protection.html, 2002.

Conklin, A., Dietrich, G. & Walz, D. *Password-Based Authentication: A System Perspective*, Proceedings of the 37th Hawaii International Conference on System Sciences, 2004.

Epstein, J. *What is a Network Assurance Platform, You Ask? Signal2Noise*. http://s2n.merunetworks.com/2009/10/what-is-a-network-assurance-platform-you-ask/, October 2009.

Guster, D., Safonov, P. Hall, C. & Podkorytov, D. *Business Computer Information Systems Security Against Hacking Attacks:Application of Distributed Processing and Software Modifiers in Defense of Password Files*, Proceeding of the Academy of Business Administration's National Conference, Las Vegas, NV, 2004.

Holt, T. and Kilger, M. *Techcrafters and Makecrafters: A Comparison of Two Population of Hackers*. Proceedings of the WOMBAT Workshop on Information Security Threats Data Collection and Sharing, IEEE Computer Society, Washington, DC, USA, 2003.

John the Ripper Password Cracker. http://www.openwall.com/john/, 2003.

Keefe, B. *Computer Crime Research Organization News*, http://www.crimeresearch. org/news/2003/04/Mess0902.html, 2004.

Kotenko, I. and Bogdanov, V. *Proactive Monitoring of Security Policy Accomplishment in Computer Networks*. IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications. Rende, Italy, 2009.

National Archives *Vital records and records disaster mitigation and Recovery: An instructional guide, Web Edition*, http://www.archives.gov/records mgmt/vital-records/index.html, 2008.

Netcat. *Netcat Users Guide*, http://66.14.166.45/whitepapers/netforensics/netcat/Ncat%20Users%20Guide.pdf, 2009.

Nmap. *User's Manual for Nmap*, http://nmap.org/, 2012.

Podkorytov, D. *User's Manual for Strombringer*, Kurgan State University, Russia, 2012.

Scarfone, K., Souppaya, M., Cody, A. and Orebaugh, A. *Technical Guide to Information Security Testing and Assessment.* NIST. Special Publication: 800-115, 2008.

Sectools.org. http://sectools.org/, 2006.

Shevtekar, A. and Ansari, N. *Do Low Rate DoS Attacks Affect QoS Sensitive VoIP Traffic?*, Proceedings of IEEE ICC 2006, Istanbul, Turkey. 2153-2158, June 2006.

Stanger, J. *Security Testing with HPing at the hop*. Linux Magazine, February 2009.

Strembeck, M. *Embedding Policy Rules for Software-Based Systems in a Requirements Context*, Proc. of the Sixth IEEE Intern. Workshop on Policies for Distributed Systems and Networks, Stockholm, 2005.

Suresh, M. and Anitha, R. *Evaluating Machine Learning Algorithms for Detecting DDoS Attacks*, Advances in Network Security and Applications: Communications in Computer and Information Science, 196(1), 441-452, 2011.

Symantec. *Symantec Security Response: Hacktool.netcat*, December 2005.

Varian, H. *System reliability and free riding, In Economics of Information Security*, L. J. Camp, S. Lewis, eds. (Kluwer Academic Publishers, 2004), vol. 12 of Advances in Information Security, 2004.

Whitman, M. and Mattord, H. *Principles of Information Security*, Course Technology, page 332, 2002.

Ye, Z., Shi, W. and Ye, D. DDoS *Defense Using TCP/IP Header Analysis and Proactive Tests*, Proceeding of the International Conference on Information Technology and Computer Science, IEEE Computer Society, Washington, 2:548-552, 2009.