A Distributed Defense Mechanism Against Low–Bandwidth TCP SYN Flooding Attacks

Emmanuel Guiton
Helsinki University of Technology
Networking Laboratory
P.O. Box 3000
FIN–02015 HUT, Finland
Email: emmanuel.guiton@hut.fi

***Abstract*— TCP servers have typically few resources dedicated to connection establishments; only a small amount of half–open connections can be queued. TCP SYN flooding attacks perform denial of service by exhausting this backlog buffer capacity. Typical local networks include normal workstations whose backlog queues are not used as they are not meant to be servers. Thus, these networks have a pool of available resources that can be used to release a server from a flood of TCP connection requests. This paper presents a new firewall–based defense mechanism that silently distributes the load of TCP SYN requests amongst several hosts when a server is overloaded. A prototype implementation showed the viability of this end–system defense approach as long as the bandwidth capacity of the network links is not the bottleneck resource. This method can be easily implemented and a prototype showed that 3 computers can withstand a 280 kb/s (650 messages/s) TCP SYN flooding attack.**

***Index Terms*— Denial of Service, TCP SYN flooding, firewall, Network Address Translation, Connection Request Distribution.**

I. Introduction

In spite of its oldness, the TCP SYN flooding technique [1] is still one of the most common methods to perform Denial of Service (DoS) attacks [2]. Well-known DoS attack tools such as TFN [3] [4], TFN2k [5], Stacheldraht [6], and also viruses or worms like W32/Blaster [7] include this functionality. Several mechanisms have been proposed to cope with that problem but no complete solution still exists. Some issues are directly related to the place of the defense mechanisms on the attack path: the closer to the attack sources they are, the more false–positives they trigger. In return, the closer to the target, the less able to cope with flooding they are. However, when bottleneck links are not overloaded, end–system defenses can be the most effective and reliable. [1] presents such methods, including firewall based proposals like the one described hereafter.

This paper proposes a new approach that aims to distribute the TCP connection establishment load amongst several hosts in a local network. The mechanism is managed by a firewall that forwards and redirects messages so that connection requests are always sent to an host which has available resources. First, every connection request is normally forwarded to the server. When an attack occurs and the server's backlog queue gets full, the firewall uses the Network Address Port Translation (NAPT) method to redirect the TCP SYN requests towards one or several another host in the local network, which we will call relays. When a connection is established with a relay, the firewall moves it to the real server so that the legitimate client and the server can communicate normally.

In the next section, overall information about DoS attacks, the TCP SYN flooding mechanism, and NAPT is provided. Then, two firewall–based defense mechanisms closely related to the proposal in this study are presented. Following is the description of the proposal itself with particular attention on its critical issues. This proposal has been implemented and tested in a Linux environment. The results are provided in the last but one section before concluding this paper.

II. Background

A. Denial of Service attacks

DoS attacks aim to deny legitimate users the access to a service. A famous example is the wave of attacks in the early February 2000, when the websites of Amazon, CNN, eBay, Yahoo!, etc. were unreachable during several hours [8]. More recently, the SCO Group, Inc. has suffered a distributed DoS attack against its website from 1st to 12th of February 2004, rendering it inaccessible. To maintain the website accessibility, the company chose to change the address and to remove the original one from the DNS databases.

DoS attacks can be separated into two different categories. Logic attacks rely on intelligent exploitations of software vulnerabilities. Flooding attacks aim to exhaust a resource such as link bandwidth, processing power, disk space, etc. This second category includes the TCP SYN flooding attack, described in detail in [1]. The TCP SYN flooding was the relying technique used in the attacks mentioned above. The W32/Novarg.A virus (also called Mydoom) [9], which was used in the attack against SCO illustrates that this method is still used and very effective. From now on, we will concentrate on that particular kind of attack.

B. Defense methods

To defend against TCP SYN flooding attacks, general methods against DoS attacks and also several specific methods

can be used. Specific methods aim to improve the attack resistance or the reliability of the TCP connection establishment mechanism. Such methods include an optimization of servers' resources, the use of firewalls [1], active monitors [1], puzzle auctions [10], and so on. Some other defense mechanisms like blocking or several systems based on rate–limiting [11] [12] [13], aim to disrupt the flooding itself and can generally be applied to mitigate other flooding attacks. Usually, these methods also rely on traffic identification and selection, for the purpose of which Intrusion Detection Systems (IDS) are often used. Defense techniques also differ by their places on the attack path. While the first quoted ones are implemented on the target side, anti–flooding methods gain in being applied as close as possible to the attack sources. For that reason, a lot of recent work has focused on mechanisms that are able to trace attacks back to their sources [12], [14], [15], [16], [17], [18], [19], [20], [21].

All of these proposals have their own advantages and drawbacks. The closer to the target, the better the reliability of attack detection usually is. Traceback and rate–limiting techniques are commonly based on statistical analyses, which are liable to false–positives. Thus, legitimate users can be denied services because of these methods. On the other hand, when the defense is implemented close to the target, it cannot defend against every DoS attack. It is particularly useless against any high–bandwidth flooding attack since it cannot avoid the overload in upstream links. However, if waste of resources is not considered, it is worth noticing that an attack should not be mitigated in the upstream nodes if the target can handle it without problem. Otherwise, the defense mechanisms would risk denying services to some legitimate users without compensating with positive effects. Therefore, reliable and effective end–target defense methods are still useful, or even necessary. Moreover, it has been shown in [2] that more than 90% of the attacks on the Internet during a three week period in the beginning of year 2001 were TCP based and most probably used the TCP SYN mechanism. We can further deduce from the results in [2] that around 80% of the attacks used less than 2000 packets per second. Considering a TCP message size of 54 bytes (a TCP SYN message witout options), a bottleneck link of 1Mb/s requires 2315 messages to be overwhelmed. Thus, a reliable end–target defense method can be expected to be more effective and trigger less undue damage to legitimate users in a good proportion of attacks. However this proportion cannot be reliably deduced from [2]. For example, attack bandwidths may be adapted to the capacities of the targets' connections to the Internet. Moreover, since 2001, the substructures of the Internet has continued to grow worldwide in size and also in bandwidth capacity. Although the data given in [2] may be outdated, we can only refer to them as more recent measurements on the Internet DoS activity have not been published.

C. Network Address Port Translation

The security mechanism described in this paper is meant to be used in a local network, protected by a firewall from the public network. Indeed, the method relies on the Network Address Translation (NAT) principle, whose first description was published in [22]. It enables assigning freely IP addresses inside a local network. When a host wants to communicate outside the local network, its address is mapped into another one, which is globally unique on the outer network (the Internet, for example). The Network Address Port Translation (NAPT) [23] extended the number of possible mappings by using the {IP address, TCP/UDP port} pair instead of the single IP address.

The application areas of NAT, NAPT, and the various variants on the method [24] overtook the original goals, which was to provide a short term solution to the shortage of IPv4 addresses. Particularly, applications in security and network administration were quickly understood [25].

III. Related Work

Firewall–based defenses against TCP SYN flooding have already been proposed. In addition to a precise analysis of the TCP SYN flooding attack, [1] proposes several defense methods against it, including two firewall–based approaches.

A. Firewall as a relay

The first of these two proposals uses a firewall as a relay. The firewall is located at the interface between the local and the public network. When a TCP SYN request from the public network is sent to a local host, the firewall answers on its behalf. If the final TCP ACK message of the three-way handshake never arrives, the firewall terminates the connection. On the contrary, if the three-way handshake proceeds correctly, the firewall creates a second connection between itself and the true destination so that the communication path is completed. After that, the firewall acts as a proxy, forwarding the datagrams and translating the sequence numbers between the two ends of the connection. This method avoids that the destination host receives any spoofed TCP message. However, as reported in the same paper [1], this mechanism works only if the firewall is not itself vulnerable to TCP SYN flooding attacks. Indeed, the resource allocation issue is simply moved from the server to the firewall. Moreover, this method adds delay to every legitimate connection. Finally, the timeout period must be carefully selected; otherwise legitimate hosts with long response times may not be able to establish TCP connections.

B. Firewall as a semi–transparent gateway

The second firewall–based proposal in [1] uses a firewall as a semi–transparent gateway. In this case, the firewall monitors and reacts to the traffic exchanged between a local network and a public network. Unlike in the previous method, the TCP SYN and TCP ACK messages go through the firewall. However, when an internal host generates a TCP SYN+ACK message, the firewall establishes the connection with the local host by sending a TCP ACK message to it. This operation aims to free the resources allocated for the half–open connection in the local host's backlog queue. Once in this situation, two cases can occur. Firstly, the TCP SYN message can be part

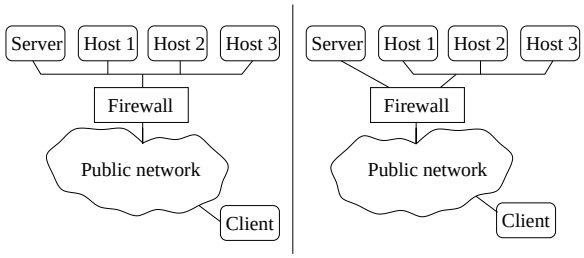

Fig. 1. Typical network environments of the proposal.

of an attack. Then, no TCP ACK message is generated by the source host and after a defined length of time the firewall sends a TCP RST message to terminate the connection at the local host side. Secondly, the TCP SYN message can be issued by a legitimate station, which finishes the three-way handshake by sending a TCP ACK message. This second TCP ACK message reaches normally the internal host and it does not create any problem since the TCP protocol is designed to cope with duplicate messages. Thus, the second TCP ACK message is silently discarded and the connection can continue without any additional operations from the firewall. As also explained in [1], this method does not create additional delays. However it transforms half–open connections into illegitimate open connections. As the resource limit on open connections (mainly processing power and memory) is much higher than the resource limit on half–open connections, this mechanism is both the main advantage and the main drawback of the proposal. Also, for the same reasons than in the previous method, the timeout length has to be carefully selected.

IV. Firewall as a Connection Request Distributor

A. Description of the defense mechanism

The goal of a TCP SYN flooding attack is to fill the victim's backlog queue with pending half–open connection. In many real-life environments, the target is typically part of a local network or it can be placed in a different but adjacent local network. These cases, represented in Fig. 1, are commonly found in small and medium–size companies. Most of the hosts in such local networks are normal workstations which are used, for example, for office work. Thus, they seldom use their resources to accept TCP connections as they are not configured to provide TCP services. These workstations form a pool of available resources that can be used to release an overloaded server of TCP connection establishments. In that way, there can most probably be enough resources to handle any flood of TCP SYN requests as long as network link capacity is not the bottleneck resource.

The defense mechanism relies on using a firewall that performs NAPT [23]. The transport layer information must be accessible as actions are taken according to the establishment status of connections. When a client wishes to connect to the server from the public network, it uses the IP address of the firewall or directly the address of the website if its publicly available (when it is separated from the rest of the network

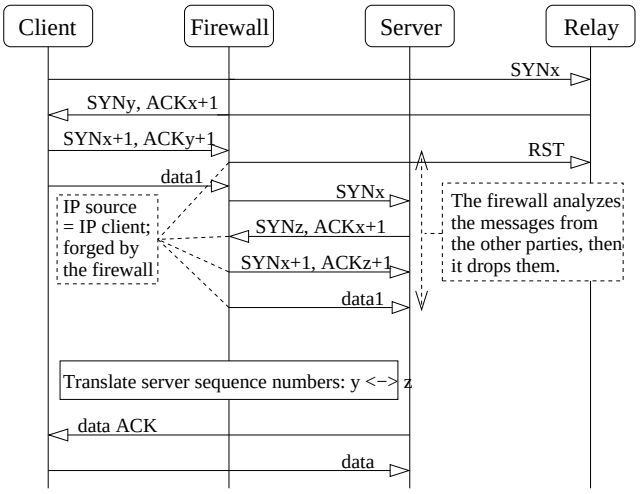

Fig. 2. Sequence chart of a legitimate client connection when redirected to a relay host.

like the case on the right in Fig. 1). None of the hosts involved in the process (server, local hosts, remote client) need to be aware of the presence of the firewall. Thus, the actions of the firewall are transparent for all of the parties. At this point, two cases are possible:

- The server is not under attack.
  The firewall directs the client's messages to the server, possibly performing usual NAPT operations, which improve the security of the local networks (by hiding local addresses to the hosts in the public network).
- The server is under attack and its queue of half–open connections is nearly full.
  The firewall fairly redirects the TCP SYN requests on a few hosts in the local network, which we will call relays. Relays are only used to establish TCP connections and every successful connection is moved back to the server. The relays only need an available port on which the TCP three–way handshakes can be performed; it does not even need to accept any data. The proceeding of the operations is depicted in the sequence chart of the Fig. 2. The firewall is used as a man–in–the–middle to act in behalf of the client. For that purpose, it must record some client data when tracking the connection so that it is able to forge messages to establish a connection with the real server. The firewall does not have direct knowledge of the relay's resource statuses, it only deduces them thanks to pre–configured parameters and calculations.

B. Critiques

This proposal addresses the drawbacks of the two firewall-based systems proposed in [1]. Here, the firewall is not vulnerable to TCP SYN flooding as it does not manage the connections itself but it only forwards the packets. The end hosts of a TCP connection need to allocate several data structures such as the socket structure and the TCP and IP control block structures in BSD style network code [1].

These structures include data and buffers which are used for sending and receiving; operations which are not performed by the firewall in the connection request distribution (CRD) scheme. In that case, the firewall uses only a limited amount of resources; mainly to store the critical data it needs to monitor connections. The firewall will only allocate the data structures used in TCP connections when forging packets. This case only happens when a legitimate connection is established with a relay. Four messages are forged (see Fig. 2), then the data structures are released.

Moreover, as long as the server is not under attack, legitimate connections do not suffer from additional delay. Extra-delay only occurs during the connection establishment when an attack is in progress. Compared to the second proposal in [1], there is no illegitimate connections that are created. Finally, in the firewall as a CRD system, there is more liberty in choosing the TCP timeout period as the mechanism does not only rely on the resources of one but of several stations.

Compared to other defense mechanisms like the ones referenced in the second section, this defense system ensures that none of the legitimate connection requests is dropped. Moreover, using NAPT enables hiding the infrastructure of the local network. Thus, there is no risk that an attacker learns about the local hosts, they cannot be distinguished from the server. The firewall as a CRD system is also a transparent mechanism for all clients in the public network and servers and hosts in the local network. Thus, this proposal can be easily implemented; it only requires modifying the firewall software. As only the TCP three-way handshake is allowed with the local hosts, the CRD system cannot be abused to attack them, except when exploiting a faulty implementation of the TCP protocol.

This defense mechanism has also its own drawbacks. It does not avoid the illegal consumptions of certain resources: link bandwidth, TCP ports as NATP identifies the different connections thanks to the {IP address, TCP port number} pairs and memory to store the data required to track connections. However, the availability of these resources exceed the consumption capabilities of the attacker inside the local network. In the case of a successful connection establishment, the systems generates 4 additional messages compared to a direct connection with the server. However this network load is negligible compared to the load triggered by the flooding attack. Finally, this solution relies on several programs that are not free from bugs and security vulnerabilities (e.g. [26]).

C. Critical issues

1) Counting the number of half–open connections: In order to control the distribution of the TCP connection requests, the firewall has to know the state of the half–open queues in the server and in the relays. These calculations are critical because the system can fail to detect the TCP request flooding if deduction is not properly done or if an attacker can deceive it. However, there is no need to involve the local hosts in the process as long as the firewall knows the TCP timeout length of the stations (but no synchronization is needed) and the maximum number of half–open connection they can handle. Then, the amount of pending half–connection can be calculated considering that:

- Each new TCP connection attempt is considered as being one more half–open connection.
- Each TCP connection reaching the established state (a correct TCP SYN ACK message has been received) decreases the number of half–open connections by one.
- Each connection expiring without having reached the established state decreases the number of half–open connections by one.

2) Translating the Server's Send Sequence numbers: When establishing a connection with a relay, the client receives from it a certain Initial Send Sequence (ISS) number value. This value is critical for both parties to communicate correctly. However, when the connection is moved to the server, a new ISS number is generated (and blocked at the firewall, see Fig. 2). The server expects the client to use this new number while the client expects the server to use the ISS it received. In order to reconcile these two requirements, the firewall must calculate the gap between the two sequence numbers and then it has to keep on translating the sequence number values between the two hosts. Note that the client's ISS does not need to be translated as the firewall can correctly replay it when establishing the connection to the server.

The same issue also applies to a few other options. The selective acknowledgement option is directly linked to sequence numbers and must be changed accordingly. The timestamp option works in a fashion similar to the sequence numbers and also requires translating the server's values.

3) Storing data: When tracking the connections, the firewall must be aware about the IP addresses and TCP ports involved. It requires also additional data as connections established with relays must be moved to the real server. To forge the necessary messages, it must know the three different ISSs, the TCP and IP flags (such as no fragmentation, explicit congestion notification) and the possible optional fields like TCP timestamps.

The firewall must also store any data packet coming from the client while the connection is not yet established with the server. This issue does not require much resources: only one packet may be sent along with the SYN ACK packet and the memory will be released quickly as the connection is legitimate.

V. Implementation and Network Environment

A. Implementing the defense mechanism

The test implementation was carried out using the code of the netfilter/iptables open source project [27]. It was implemented on a Linux kernel version 2.4.23, to which was added a patch provided on the netfilter/iptable project website. The netfilter firewall is divided into two parts: a user level application, which simply allows users to provide configuration data, and several kernel modules, which perform the firewall and NAPT operations. Additions to both of these

parts were necessary to implement the connection request distributor.

B. Configuring the Linux server and hosts

To implement the defense system, the first step is to optimize the resources of the server and hosts, so that they can handle the highest possible load of TCP SYN requests by themselves. Using a Linux OS, the behaviour of the TCP protocol can be customized by modifying the files located in the /proc/sys/net/ipv4/ and /proc/sys/net/core directories. (Old versions of the Linux kernel may require to change directly all or part of these variables in the kernel code sources.) Several guides, like [28] and [29], indicate how to set the values contained in these files. A simple way to make the changes is to use the sysctl utility. The following parameters are of particular interest:

The tcp_max_syn_backlog file indicates the maximum number of TCP SYN requests that can be stored in the backlog queue. The default value is dependent on the host's memory but it can be extended. However this value will be practically limited by the host's resource capacities.

The tcp_synack_retries contains a figure that corresponds to the maximum number of TCP SYN ACK messages that a destination host will send before closing the connection when the source host is not responding. With the default value, which is 5, the timeout occurs after 3 minutes. To cope with TCP SYN flooding attacks, it is recommended to leave only 1 retransmission so that the connection timeouts after 9 seconds without receiving an ACK message.

Finally, the memory allocated to TCP reception buffers can be increased by changing the default values in the following files: core/rmem_max, core/rmem_default, ipv4/tcp_rmem. The whole memory allocated to TCP can also be increased in ipv4/tcp_mem.

C. The test network environment

The performances of the whole CRD system depend on the performances of each station involved in the process. TCP performances depend on both software and hardware thus they vary from host to host. In that regard, the tests were affected by a low–performance environment. The prototype was tested on a network made up of Linux PCs with 2.4.20-2.4.23 kernels, 100 to 400 MHz processors, less than 100 MB of memory except one host which had 256 MB. Figure 3 shows the overall arrangement of the hosts. The network links are 100 Mb/s ethernet.

The SYN flooding attacks were carried out by a kernel-level program and targeted an HTTP server. The experimentations were successfully conducted using different port numbers on the relays (21 (FTP), 22 (SSH), 23 (telnet), 80 (HTTP)...) to ensure that the system was not dependent on a particular service. By experimenting, it was found that the server and one relay could handle up to 1537 SYN requests at the same time while the best relay could handle twice that amount.

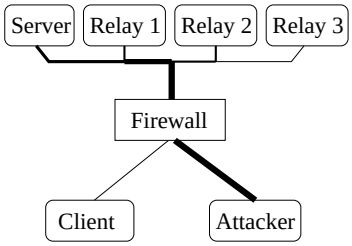

Fig. 3. The test network. The path of the TCP SYN flow is emphasized.

VI. Tests and Analyses

A. Overall performances of the CRD system

The main test was to determine the maximum attack bandwidth that the CRD system could handle. In order to find that limit, the flooding program was used with different sending rates during time intervals up to 10 minutes. In theory, the target and the two relays could handle up to 682 messages per second (almost 295kb/s). This rate is much less than what the network links and the network cards could stand. Thus, the TCP implementations on the different hosts, not the physical network, were the bottleneck. In practice, the experiments show that the CRD system could withstand a maximum of 650 messages per second (280kb/s) without a message being lost nor a legitimate connection failing.

The difference between practice and theory is due to the firewall's calculations. Regarding the target's and the relays' queue statuses, there was always a gap between reality and the estimations of the firewall. Minimizing this gap was one important issue to reach the 650 message per second limit.

It is worth noticing that the best relay was enduring half of the connection request load (140kb/s). 8 hosts like this one could withstand more than 1 Mb/s of TCP SYN flooding (1,12 Mb/s). Moreover, one should remember that the hardware equipments used for the tests do not reflect today's computers' capabilities. Using modern equipments can certainly lead to better results, but this could not be tested here.

B. Performances from the user point of view.

From the user point of view, this method produces nearly perfect results, only limited by the scope of the mechanism: low-bandwidth attacks. In case the server is not overwhelmed by connection requests, the user will connect directly to the server without performance degradation. On the other hand, while the server is flooded, a small delay happens: the length of time required to move the established connection from a relay to the server. Considering that the server and the relays are very close, probably on the same local network, this additional delay is very small when comparing to delays experienced in the public Internet, for example. In the tests described above, an average of 1.26 ms additional delay was measured for each connection which was first established with a relay while the system was under attack.

VII. Conclusion

This paper introduced a new end-system mechanism to defend a server against TCP SYN flooding attacks. The connection request distribution system uses a transport level packet based firewall to distribute the load of TCP SYN requests between several hosts. When a connection is established with a relay (the connection is legitimate), it is moved back to the real server. This method does not require special resources and it is also easy to implement (as an additional feature to existing firewall software), deploy and configure. A Linux–based prototype system showed the viability of the mechanism. Tests showed that a three-computer network could withstand 280 kb/s of TCP SYN flooding. The prototype should now being further developed and optimized to create a safely deployable system whose performances should easily approach theoretical bounds. The implementation flaws should be corrected and the design should be improved to cope with the complexity of real life environments (where potential relays are not always up and running, for example). This paper does not investigate all the performance aspects of the system. More tests could be performed with an improved version of the prototype and a network environment reflecting modern equipment capabilities. Particularly, performance issues in the firewall and the relays should be addressed. Finally, this paper introduced the idea of a distributed defense to address a particular issue: the TCP SYN flooding. The method may suit similar issues for other protocols.

References

[1] C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, and D. Zamboni, "Analysis of a denial of service attack on TCP," in *Proc. IEEE Symposium on Security and Privacy 1997*, May 4–7, 1997, pp. 208–223.

[2] D. Moore, G. M. Voelker, and S. Savage, "Inferring internet denial of service activity," in *Proc. Tenth USENIX Security Symposium*, Washington D.C., Aug. 2001.

[3] "Distributed denial of service tools," CERT Coordination Center, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, Incident Note IN–99–07, Jan. 15 2001. [Online]. Available: http://www.cert.org/incident_notes/IN–99–07.html#tfn

[4] D. Dittrich. (1999, Oct. 21) The DoS project's "trinoo" distributed denial of service attack tool. University of Washington. Seattle, Washington, USA. [Online]. Available: http://www.staff.washington.edu/dittrich/misc/trinoo.analysis

[5] "Denial of service tools," CERT Coordination Center, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, Advisory CA–1999–17, Mar. 3 2000. [Online]. Available: http://www.cert.org/advisories/CA–1999–17.html

[6] D. Dittrich. (1999, Dec. 31) The "stacheldraht" distributed denial of service attack tool. University of Washington. Seattle, Washington, USA. [Online]. Available: http://www.staff.washington.edu/dittrich/misc/stacheldraht.analysis

[7] "W32/blaster worm," CERT Coordination Center, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, Advisory CA–2003–20, Aug. 14 2003. [Online]. Available: http://www.cert.org/advisories/CA–2003–20.html

[8] L. Garber, "Denial–of–service attacks rip the internet," *IEEE Computer*, vol. 33, no. 4, pp. 12–17, Apr. 2000.

[9] "W32/novarg.a virus," CERT Coordination Center, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, Incident Note IN–2004–01, Jan. 30 2004. [Online]. Available: http://www.cert.org/incident_notes/IN–2004–01.html

[10] X. Wang and M. K. Reiter, "Defending against denial–of–service attacks with puzzle auctions," in *Proc. Symposium on Security and Privacy*, May 11–14 2003, pp. 78–92.

[11] A. Gaeil, K. Kiyoung, and J. Jongsoo, "MF (minority first) scheme for defeating distributed denial of service attacks," in *Proc. Eighth IEEE International Symposium on Computers and Communication (ISCC 2003)*, 2003, pp. 1233–1238.

[12] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker, "Controlling high bandwidth aggregates in the network," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 3, pp. 62–73, July 2002.

[13] D. Sterne, K. Djahandari, B. Wilson, B. Babson, D. Schnackenberg, H. Holliday, and T. Reid, "Autonomic response to distributed denial of service attacks," in *Proc. Fourth International Sympsium on Recent Advances in Intrusion Detection*, Davis, California, USA, Oct. 2001, pp. 134–149.

[14] S. Bellovin and T. Taylor. (2001, Oct.) ICMP traceback messages. Internet draft. [Online]. Available: www.globecom.net/ietf/draft/draft–ietf–itrace–01.html

[15] A. Mankin, D. Massey, W. Chien-Lung, S. F. Wu, and L. Zhang, "Autonomic response to distributed denial of service attacks," in *Proc. Tenth International Conference on Computer Communications and Networks*, Oct. 15–17 2001, pp. 159–165.

[16] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer, "Hash–based ip traceback," in *Proc. ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, San Diego, California, USA, 2001, pp. 3–14.

[17] W. T. Strayer, C. E. Jones, F. Tchakountio, A. C. Snoeren, B. Schwartz, R. C. Clements, M. Condell, and C. Partridge, "Traceback of single ip packets using spie," in *Proc. DARPA Information Survivability Conference and Exposition*, vol. 2, Davis, California, USA, Apr. 22–24 2003, pp. 266–270.

[18] H. Aljifri, "IP traceback: a new denial–of–service deterrent?" *IEEE Security & Privacy Magazine*, vol. 1, no. 3, pp. 24–31, May/June 2003.

[19] K. T. Law, J. C. S. Lui, and D. K. Y. Yau, "You can run, but you can't hide: an effective methodology to traceback ddos attackers," in *Proc. Tenth IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS 2002)*, Oct. 11–16 2002, pp. 433–440.

[20] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Network support for ip traceback," *ACM IEEE Transactions on Networking*, vol. 9, no. 3, pp. 226–237, June 2001.

[21] M. T. Goodrich, "Efficient packet marking for large–scale ip traceback," in *Proc. Nineth ACM conference on Computer and communications security*, 2002, pp. 117–126.

[22] K. Egevang and P. Francis, "The ip network address translator (NAT)," RFC 1631, Internet Engineering Task Force, May 1994.

[23] P. Srisuresh and K. Egevang, "Traditional IP network address translator (traditional NAT)," RFC 3022, Internet Engineering Task Force, Jan. 2001.

[24] P. Srisuresh and M. Holdrege, "IP network address translator (NAT) terminology and considerations," RFC 2663, Internet Engineering Task Force, Aug. 1999.

[25] M. Smith and R. Hunt, "Network security using NAT and NAPT," in *Proc. 10th IEEE International Conference on Networks (ICON 2002)*, Aug. 27–30 2002, pp. 355–360.

[26] "Linux kernel netfilter irc dcc helper module creates overly permissive firewall rules," US-CERT, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, Vulnerability Note VU#230307, July 05 2002. [Online]. Available: http://www.kb.cert.org/vuls/id/230307

[27] (2004) The netfilter / iptables project homepage. [Online]. Available: http://www.netfilter.org/

[28] O. Andreasson. (2003, May 21) Ipsysctl tutorial 1.0.4. webpage. [Online]. Available: http://ipsysctl-tutorial.frozentux.net/ipsysctl-tutorial.html

[29] B. L. Tierney. (2001, Feb.) TCP tuning guide. webpage. [Online]. Available: http://www-didc.lbl.gov/TCP-tuning/TCP-tuning.html