Port Hopping for Resilient Networks

Henry C.J. Lee, Vrizlynn L.L. Thing
Institute for Infocomm Research
Singapore
Email: {hlee, vriz}@i2r.a-star.edu.sg

Abstract— **With the pervasiveness of the Internet, Denial-of-Service (DoS) and Distributed DoS (DDoS) attacks have become important threats to servers, hosts and devices that are connected. This paper addresses the problem of mitigating the DoS/DDoS attacks so as to ensure that legitimate traffic is given an acceptable level of quality of service. We proposed a new technique, called port hopping where the UDP/TCP port number used by the server varies as a function of time and a shared secret between the server and the client. The main strength of the mechanism lies in the simplification of both the detection and filtering of malicious attacks packets and that it does not require any change to existing protocols. This port hopping technique is compatible with the UDP and TCP protocols and can be implemented using the socket communications for the UDP protocol, and for setting up TCP communications. We performed both theoretical analysis and empirical studies through actual implementation to study the effectiveness of the scheme against DoS/DDoS flooding attacks. Our experiments show that the port hopping technique is effective in detecting and filtering malicious traffic, and hence improved the reliability of good traffic flow.**

I. Introduction

The Internet is becoming increasingly pervasive everyday with the emergence of new wireless technologies and devices that provide anytime anywhere access to the Internet. Today, hosts connected to the Internet include mainly servers and client computers, PDA etc. In the future, other devices such as embedded appliances, cars and anything that runs on electricity will be connected, especially with the emergence of IPv6. These hosts will benefit enormously from the Internet connectivity. However, when they are connected to the Internet, they will become potential attack targets. The more pervasive the connection, the more vulnerable they will become. The pervasiveness of the Internet becomes a double-edge sword. Presently, conventional enterprise security mechanisms, such as firewall and intrusion detection/prevention systems (IDS/IPS) are used to provide managed security services in an enterprise or ISP framework. As we have more types of devices and appliances connected, the current security measures will not be appropriate for the new paradigms. From another angle, if the appropriate security measures are not in place, the pervasiveness of the Internet will be hampered as users will be reluctant to connect their devices to the Internet due to the insecurity. Consequently, new and simple security methodologies are needed to address the new security concerns of new types of devices or appliances, which typically run very simple applications.

One of the main threats of Internet security is DoS (Denial of Service) and DDoS (Distributed DoS) attacks [3]. Such attacks have paralyzed high profile web sites. Although there has been less publicity since then, research and measurements [6] have shown that the attacks are still raging on everyday. As such devices typically have lower networking bandwidth and computing resources, they are more vulnerable to such attacks as compared to servers protected by layers of firewalls and IDS/IPS. Many techniques have been proposed (see section II) and studied to address the problem of DoS/DDoS attacks. However, the drawback of most approaches is they are complex and that significant changes to the Internet routers are generally required.

In the TCP/IP protocol suite, UDP [8] and TCP [10] protocols are used for connectionless and connection-oriented data transfer. Internet applications (e.g. client and server) use the socket interface to establish a communication channel for exchanging data using the UDP or TCP protocols. The end points of the communications channels are defined by the end hosts' IP addresses and their UDP and TCP port numbers, i.e. <source and destination IP address, source and destination port number>. For a communications session using UDP or TCP, both ends use fixed port numbers. The port number varies from 0 to 65535. Well known ports (0 through 1023) [4] are used by servers to provide standard services. For example, web services running the http protocol typically listen on port 80. This paradigm has some major problems. Firstly, the well known port design is vulnerable to port scanning by attackers. When the vulnerability is discovered, DoS/DDoS attack can be launched against the target (e.g. SYN flood directed at port 80 for http services) and it is very difficult to detect and filter out attack packets reliably.

In this paper, we proposed and studied a new practical technique, called *port hopping*, to detect and mitigate the effect of DoS/DDoS flooding attacks. In this technique, the server's port numbers change dynamically as a function of time and a cryptographic key shared between the server and the client. Authorized clients who have the key will be able to determine the current port number used by the server, whereas the malicious users will not know the current valid port number. The server can then easily filter off illegitimate packets by inspecting the port number contained in the UDP/TCP headers.

This paper is organized as follows: section II gives some background information on related research work to counter or mitigate DoS/DDoS attacks. Section III describes the port hopping techniques and section IV describes the performance aspects. Finally section V concludes this paper.

0-7803-8521-7/04/$20.00 (C) 2004 IEEE

II. Related Work

Many countermeasures have been proposed to address the problem of DoS/DDoS. Ingress filtering [2] seeks to prevent IP [9] packets with spoofed source address to leave stub networks. However, its effectiveness is limited by the difficulties in enforcing its universal deployment. Another approach that has been extensively studied is IP Traceback, which seeks to determine the true topological source address of spoofed IP packets, which can then be used to institute accountability. Proposed IP Traceback schemes include probabilistic packet marking (PPM) [12], IP logging [13] and ICMP Traceback [1]. In PPM, the routers probabilistically add partial path information in the IP header the packets that they forward. The markings are typically done in the 16-bit identification field of the IP packets. When a victim detects an attack, it can then reconstruct the attack path using a sufficient number of such marked packets. In the IP logging approach, the routers log the passage of all IP packets. The key challenge lies in the potential huge amount of storage required. The SPIE (Source Path isolation Engine) is proposed in [13] for a router to determine if it has processed a given packet. When a victim detects an attack, it can query successively the upstream routers if they have processed the attack packet and hence determine the attack path. In this scheme, the victim only needs one single attack packet to reconstruct the attack path. In the ICMP Traceback scheme, whenever a router forward an IP packet, it generates a new ICMP packet, called ICMP Traceback, with a low probability and sends it to the same destination address as the IP packet. The ICMP Traceback packet contains information about the router address. Upon reception of a sufficient number of ICMP Traceback packets, the victim can reconstruct the attack path and hence discover the true IP address of the attacker. However, most of these solutions requires significant changes to the routers and end hosts, and their practicability effectiveness in real time DoS/DDoS mitigation have not been demonstrated.

In [5], the authors proposed an architecture called *Secure Overlay Service (SOS)* that proactively mitigates DoS attacks. It is geared towards supporting emergency services or similar types of communications. The architecture is constructed using a combinations of secure overlay tunnelling, routing via consistent hashing and filtering. Again, the drawback of this scheme is the necessity to make widespread changes to the Internet infrastructure routers.

In [11], the authors proposed an interesting technique called *Netbouncer* for DoS/DDoS mitigation which proposed to test the legitimacy of incoming packets. This approach is an end-point-based solution to DoS/DDoS protection, in that changes are made to the servers or clients, but not to the Internet routers. The legitimacy tests are carried out by the end hosts, and can be conducted at the network (IP) layer, transport (TCP) layer or application layer. For example, at the IP layer, the end host can test if the IP address in an incoming packet is associated with a live host. Such test can be carried out using ICMP echo message. The key issue of this approach is to find a simple and effective legitimacy test.

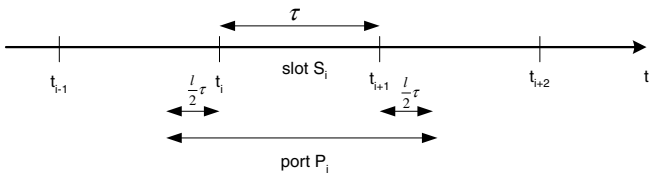

Fig. 1. Time Slot Structure

III. Port hopping technique

A. Basic Mechanism

In this paper, we proposed and studied a new technique, called *port hopping*, to detect and mitigate the effect of DoS/DDoS attacks. We proposed to change the server's port numbers dynamically as a function of time. In this scheme, time is divided into discrete slots S_i, where $i = \{0, 1, 2...\}$, each of duration τ. In conventional UDP/TCP services, the port number used is unchanged for a specific communication session. In our scheme, different port numbers are used in different time slots for the same service.

Let P_i represents the port number used by the server in time slot S_i. P_i is determined by equation (1), where k is a shared cryptographic key between the server and the client and f a pseudo-random number generator.

$$P_i = f(i, k) \tag{1}$$

When a client needs to communicate with the server, it will determine the server's current port number P_i using the shared secret key k and the time slot number i. When the server receives packets that carry "invalid" port numbers, they can be easily detected and filtered off. There is no need for the server to examine the contents of the packets in order to determine if a packet is malicious. As a result, the computational resources needed to detect and filter off the malicious packets is reduced significantly. However, packet arrival may take place near the boundary of time slots. In order to take into consideration the time synchronization errors between the server and the client, two ports are used at the boundaries of time slots. We define l as the overlapping time slot factor. This parameter is used to address two issues: the time synchronization error between the client and the server, and the transmission delay between them. This mechanism reserves a part of the previous and next time slot to allow for packet exchange taking place near the boundaries of time slots. Let t_i be the start of time slot i and P_i the associated port number. Then P_i is defined to be valid from time $(t_i - \frac{l}{2}\tau)$ to $(t_{i+1} + \frac{l}{2}\tau)$ (see Figure 1). For example, When $l = 0$, a port number is valid only within a specific time slot. When $l = 0.2$, a port number is valid for a total duration of 1.2τ, from $(t_i - 0.1\tau)$ to $(t_{i+1} + 0.1\tau)$.

B. Compatibility with existing protocols

1) UDP: This port hopping technique can be implemented for the UDP protocol using the UDP socket mechanism. We

discussed earlier that an overlapping factor is required to address the problems of time synchronization error as well as transmission delay. At the server end, it opens a new datagram socket at time $(t_i - \frac{l}{2}\tau)$ and subsequently closes the datagram socket after a duration of $(l+1)\tau$. At the same time, low level packet filter is set to accept only UDP packets containing the correct port number field. As for the client, it needs only maintain the same datagram socket connection as its own port number remains the same. The client only needs to change the destination port number periodically.

2) TCP: This scheme can be adapted for TCP as well but with some limitations. This is because TCP connection requires a 3-way handshake process. Once a connection is established, both ends have to use the same <source IP address, source port, destination IP address, destination port>. In other words, once the TCP connection is established, the endpoints port numbers cannot be changed. Due to the limitation, the TCP protocol will reuse the given port number for the entire connection.

The client starts by sending the SYN packet to the port number P_i used by server. Upon reception of this SYN, the server will reserve P_i for the entire duration of the connection with this client, by setting a low level packet filter for this particular TCP connection. This low level packet filter will be deactivated when the TCP connected is closed. The potential problem is that if the TCP connection is long, then it may be vulnerable to attack. However, studies by [14] have shown that the average duration of TCP traffic flow in the Internet is about 12-19 seconds. Consequently, the vulnerability window is small.

C. Key Management

This port hopping scheme requires the clients to know the cryptographic key k_s used by the server to determine the port number used in the current time slot. This can be achieved in two ways: using symmetric key cipher or Public Key Cryptography (PKC).

The first solution consists in using a symmetric key mechanism. With this mechanism, each client C_i shares a master key K_i with the server. These master keys should be distributed securely in an offline manner to the clients. Whenever a client needs to communicate with the server, it first requests the k_s used by the server. The server will then send the k_s to the client by encrypting it with the client's master key K_i as follows:

Client $\rightarrow$ server: Request for key k_s
Server $\rightarrow$ client: $E_{K_i}(k_s)$

Upon reception, the client only needs to carry out the decryption using the key K_i to obtain the key k_s.

The second solution consists in using the public key mechanism. With this mechanism, each client C_i has a public/private key pair, denoted PK_i and SK_i. Assuming that the PKI (Public Key Infrastructure) is set up, whenever a client needs to communicate with the server, it requests for the k_s, and the server sends the k_s to the client by encrypting it with the client's public key PK_i as follows:

Client $\rightarrow$ server: Request for key k_s
Server $\rightarrow$ client: $E_{PK_i}(k_s)$

Upon reception, the client will carry out the decryption using his private key SK_i to obtain the key k_s.

D. Upstream filtering

We have so far made the assumption that the filtering of packets is carried out at the application server. However, this arrangement may not be optimal because the bottleneck of the traffic flow may not be at the last mile. For example, an application server is connected via high speed (e.g. 100 Mbps) Intranet to a gateway, which has a much smaller pipe to the ISP router. In this scenario, the bottleneck is most likely to be the ISP router-Gateway link but not the Gateway-server link.

Consequently, the proposed port hopping scheme will be much more effective if it is implemented at the ISP end to filter off the malicious packets so as to ease the congestion at the ISP-Gateway link. For this, a secured communications link has to be established between the application server and the ISP router which is assumed to be a trusted entity. The server will periodically update the ISP router with the legitimate port number. The ISP router can then filter off packets whose <destination address, port number> pair are not correct.

E. Vulnerability issue

Assuming that the attacker knows about the port hopping scheme and will generate port numbers randomly from the range of total port numbers. Let x denotes the number of malicious packets that contain the correct port number. The average value of x, $E(x)$, is given by the equation (2) where m is the number of attackers or zombies in the case of DDoS attack. We assume that the attack codes on the compromised hosts are similar and they randomly generate port numbers to individually attack the victim. r is the rate of packets generated by the attackers. τ refers to the time slot size. The larger the time slot the higher the probability attacker will be able to guess the port numbers correctly. N is the total number of ports.

$$E(x) = \frac{mr(l+1)\tau}{N} \tag{2}$$

For example, taking values of $mr = 20000$ (assuming attack rate of 19.2 Mbit/s, packet size of 60 bytes inclusive of all the headers) $\tau = 0.5s$, $l = 1.2$, $N = 64000$, we have $E(x) = 0.38$. As a result, the rate of false positive is very low.

F. Implementation issues

From an implementation perspective, this mechanism can be implemented using a middleware approach. The middleware sits between the application and the socket layer. The middleware will handle the key management issues as well as maintaining the socket connections.

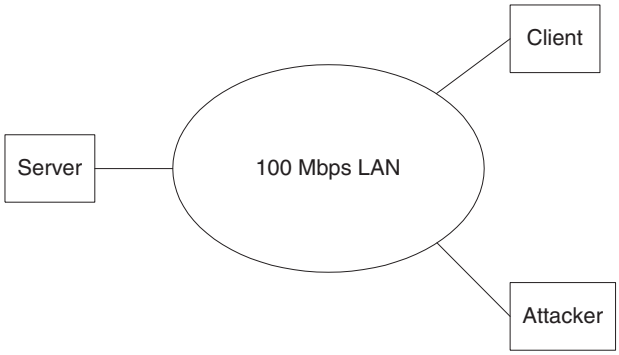

Fig. 2. Test bed for Performance Evaluation

IV. Performance Evaluation

We have carried out implementation and testing on Linux (kernel 2.4) based computers to evaluate the effectiveness of the port hopping techniques in mitigating DoS/DDoS flooding attack. The key factor that is evaluated here lies in the process of detecting if a packet is legitimate. Conventionally, an application has to inspect the payload of UDP or TCP packets in order to conclude that a received packet is malicious. In the case of the port hopping technique, this detection is greatly simplified. Compared to the conventional approach, the port hopping technique is less computational intensive and more robust in that the false positive rate is expected to be lower.

Our implementation focus on comparing the computational resource required by the port hopping technique with the conventional payload inspection approach. The test bed scenario is shown in Figure 2. The server is a Pentium 3 500MHz PC, while the client and attackers PCs are using Pentium 4 at 2.5 MHz.

In the test bed, the client and server application use UDP for data exchange. For the purpose of this studies, we emulate the port hopping mechanism by getting the attacker to send malicious data to a different port number as the client. The key issue here is the ability to distinguish good and bad packets through the inspection of the port number. In the test bed, we used the *iptables* [7] command to activate the filtering of packets with incorrect port number.

The client and server implements the pseudo port hopping mechanism. The client sends a number of UDP packets to the server. In the meantime, the attacker tries to flood the server with UDP packets at different rates. We record the rate at which the attacker flood the server and for each rate, measure the percentage of the client packets that have been correctly received by the server. The same test is performed with and without the port hopping mechanism.

The results are shown in Figure 3 which plots the percentage of good traffic received (with and without port hopping) versus the attack traffic rate in Mbps. At the lower end of the attack traffic rate (less than 22 Mbps), port hopping provides slight improvement (about 4%). At this point, the server's computational resource is still able to cope with the attack traffic. However, when the attack traffic increases beyond

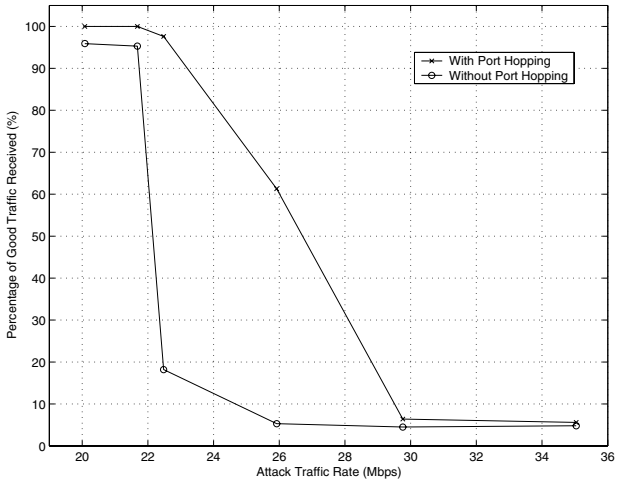

Fig. 3. Port Hopping Filtering Performance

22 Mbps, the port hopping mechanism provides significant performance improvement. Without port hopping, the good traffic received decreased significantly to below 20%. At the bad traffic attack rate increases further, the good traffic received decreases further too. However, this is due to the additional factor of network congestion on the LAN; as the bad traffic increases, the client cannot send data out onto the LAN, thus decreasing the performance of the port hopping mechanism.

The experiment results demonstrated the effectiveness of the port hopping scheme in reducing the complexity of malicious packets detection and filtering, thus reducing significantly the computational load at the server. However, the port hopping scheme may have some limitation by itself and need to be carried out further upstream as discussed in Section III-D, in order to be effective in mitigating DoS/DDoS attack.

V. Conclusion

We have presented a new and practical approach called port hopping to mitigate the effect of DoS/DDoS flooding attack. The port hopping scheme is a simple and flexible solution for countering DoS/DDoS through flooding. As compared to other approaches, its main strength lies in the simplification of both the detection and filtering of malicious attacks packets. The server only needs to inspect the port number field of UDP or TCP packets in order to determine their legitimacy.

We have described the basic port hopping mechanism, which consists in changing periodically the UDP/TCP port number used by an application. This mechanism is compatible with the UDP and TCP protocols implementations. We presented solutions for the clients to access the server key. We also studied the vulnerability issue and determined that random port number attack has minimal impact.

Our implementation experimentation demonstrates clearly the advantages of port hopping in mitigating the effect of flooding attack. However, we also discovered that the problem

of network congestion needs to be addressed too. For this, we have proposed to move the filtering to an upstream router, preferably before the bottleneck. Such an approach will be very effective in mitigating DoS/DDoS attacks.

The port hopping scheme has many advantages. Firstly, this scheme does not require any change to existing protocols, nor does it require any new addition protocol. Secondly, this scheme does not requires any changes to be made in the Internet infrastructure. These characteristics will go a long way towards facilitating the deployment of this scheme, which will enable legitimate requests from trusted clients to access the server even when it is under severe DoS/DDoS attacks. It can be used for all types of server and is particularly useful for specific types of systems or devices which have limited computing resources.

REFERENCES

[1] Steve Bellovin et al, "ICMP Traceback messages", IETF Internet Draft "draft-ietf-itrace-04.txt", Feb 2003. Work in progress.

[2] P. Ferguson, D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing", Request for Comments 2827, Internet Engineering Task Force, May 2000.

[3] K.J. Houle, G.M. Weaver, "Trends in Denial of Service Attack Technology", CERT Coordination Center, Oct 2001. http://www.cert.org/archive/pdf/DoS_trends.pdf

[4] IANA Port numbers assignment, http://www.iana.org/assignments/port-numbers, updated 9 June 2004

[5] A.D. Keromytis, Vishal Misra, Dan Rubenstein, "SOS: Secure Overlay Services", ACM SIGCOMM 2002.

[6] David Moore, G.M. Voelker, Stefan Savage, "Inferring Internet Denial-of-Service Activity", 10th USENIX Security Symposium 2001.

[7] Netfilter: Firewalling, NAT and Packet Mangling for Linux 2.4, http://www.netfilter.org/

[8] J. Postel, "User Datagram Protocol", Request for Comments 0768, Internet Engineering Task Force, 1980.

[9] J. Postel, "Internet Protocol", Request for Comments 0791, Internet Engineering Task Force, 1981.

[10] J. Postel, "Transmission Control Protocol", Request for Comments 0793, Internet Engineering Task Force, 1981.

[11] R. Thomas, B. Mark, T Johnson, J. Croall, "NetBouncer: Client-legitimacy-based High-Performance DDoS Filtering", DARPA Information Survivability Conference and Exposition (DISCEX) 2003.

[12] Stefan Savage et al, "Practical network support for IP traceback", ACM SIGCOMM 2000.

[13] Alex C. Snoeren et al, "Hash-Based IP Traceback", ACM SIGCOMM 2001, August 2001.

[14] K. Thompson et al, "Wide-Area Internet Traffic Patterns and Characteristics", IEEE Networks, Vol.11, No.6, Nov/Dec 1997.