Journal of Computer Science 1 (2): 290-295, 2005
ISSN 1549-3636

On Analysis and Design of the Enhanced Firewall for Intranet Security

[1]U.K. Singh, [2]A.K. Ramani and [3]N.S. Chaudhari
[1]MICS, Mahakal Institute of Technology, MIT Campus, Ujjain (MP)-India
[2]Institute of Computer Science and Electronics (ICSE), D.A.V.V., Indore (MP)-India
[3]School of Computer Engineering, Nanyang Technological University (NTU), Singapore

Abstract: Intranet often employ an Internet firewall to mitigate risks of system penetration, data theft, data destruction and other security breaches. Conventional Internet firewalls impose an overly simple inside vs. outside model of security that is incompatible with many business practices that require extending limited trust to external entities. Additionally, firewall security perimeters are somewhat weak: they provide no protection from inside attacks and do not protect sensitive data, which can be exported by tunneling through permitted protocols. In this study we have suggested the integration of some useful additional information along-with intrusion detection system and virus monitors into firewall. In particular, we aim at integrating as many security measures as possible into the firewall, creating what we will call an enhanced firewall. One of the main features of the enhanced firewall will be protecting intranet against various malicious attack.

Key words: Firewall, Intranet, IP-datagram, Packet

INTRODUCTION

Firewalls have been around for several years and they are a natural component of most networks connected to the Internet. In general, a firewall protects a network from unintended access from the external network, which could be the Internet. At the same time, the firewall allows the protected network to communicate with the external network. This is possible because a firewall is able to distinguish between a connection initiated from the outside (inbound) and a connection initiated from the inside (outbound). It is therefore able to restrict inbound connections to specific services, intentionally offered to the external network. At the same time, outbound connections are largely allowed. This makes the firewall almost transparent to the protected network. It also means that any application, started on the protected network, is able to communicate freely to the external network. Unfortunately, this is not always what we want. For instance, a backdoor application may run inside the protected network and secretly make a connection to a hacker on the Internet. The firewall is not able to tell whether network packets are sent from a backdoor application or a simple web-browser.

We can enhance the firewall in such a way that it is able to distinguish outgoing network packets based on the identity of the host, user and application responsible for transmitting them. If a protected host wants to communicate with the external network it is forced to reveal its identity, together with the identity of the sending application and user, for each network packet sent to the firewall. The identities are communicated using cryptographic authentication, which ensures the firewall that they are correct. This enables the firewall to employ a very strong rule-set. The rule-set guarantees that no network packets pass the firewall except when the firewall explicitly allows the specific host, user and application to send it to the external network.

In this study we recommends the design of an enhanced firewall with some additional information and techniques. These recommendations are based on the study of various research papers and work already carried out in this area. The suggested design of enhanced firewall for an Intranet prohibits the adversary from using arbitrary applications to transmit packets to the external network through the firewall, except when the adversary has valid credentials to do so. For each IP-datagram, destined to the external network, we want the firewall to be able to distinguish the identity of the sending host, user and application responsible for transmitting it. This requires the protected hosts to supply the firewall with this information, as it is not present in ordinary IP-datagrams. Since applications use direct calls to the operating system, when transmitting IP-datagrams, we will have to change the operating system on the protected hosts in order to supply this information. Furthermore, it is essential that the additional information is authenticated to the firewall, as our adversary is able to read, modify and inject arbitrary IP-datagrams on the protected network. Therefore, the protected hosts will have to authenticate each IP- datagram sent to the firewall. In turn, the firewall must verify the authenticity of the IP-datagrams before making a decision on whether to forward or drop the outgoing packet. The extended firewall has a better

basis when making the forwarding decision, compared tso the conventional firewall, since it has access to the host ID, user ID and application ID.

Thus we propose to integrate intrusion detection system and virus monitors along with the additional information into the existing firewall. The enhanced firewall with integration of IDS and virus monitors is our proposal for increasing the strength of firewalls. Which is much advanced than the earlier ones as stated in the next section. It will have the ability to examine entire data packets and to apply standard as well as intelligent detection techniques to identify misuse.

MOTIVATION AND OBSERVATIONS

In [8] Xu and Singhal proposed an ATM firewall using a proxy cache, which uses a QoF (Quality of Firewalling) scheme. Its main components are call screening, proxy, traffic monitoring service, packet filtering service and firewall management. These combined components determine a packet's safeness. The packet-filtering service inspects the headers of IP packets to block unsafe packets, while allowing safe packets to pass. The traffic-monitoring service checks the packet headers against the traffic-monitoring rules, which are similar to the packet-filtering rules. To determine whether or not a packet is safe, only the first cell will be checked, which contains the IP header, protocol, TCP/UDP ports and TCP flags. However, there are limitations: IP packets with IP option fields are not accepted, because IP options can be as large as 40 bytes and may push the TCP headers to the second cell. Using CAM (Content Addressable Memory) to cache a safe header is not a scalable solution. CAM cannot scale to a large size due to technological constraints and is extremely expensive. This work only considers TCP/IP headers-no payload information is used-to detect whether data packets are safe or not, even though they aim to develop a new firewall architecture. It does not seem that inspecting only header information is sufficient to overcome weaknesses of firewalls.

We have observed some of limitations and possibilities our adversary has, as it helps us to understand what we must protect ourselves against. We observed two different scenarios, the hostile application and the hostile user, which define two different kinds of adversaries: The malicious user, who is interested in bending the rules of the security policy to their own advantages which is internal adversary and the external hacker, who is able to make an internal user start an application of their choice, which is external adversary.

The internal adversary might not be interested in deliberately harming the intranet, but they tries to circumvent the security policy to their own benefits, which might harm the organization indirectly. To achieve their goal, the internal adversary might attach an extra host to the internal network. This enables them to do as they pleases on that host, which includes installing any software and eavesdrop on the network. This host would be a perfect place for the internal adversary to install a backdoor that connects to an external host, which in turn enables them to connect to the internal network from the outside.

The external adversary does not have the same direct access to the internal network as the internal adversary. However, once they has an application of choice started on an internal host (for instance by luring an internal user to start an application sent to him through an email attachment), they should be considered more dangerous. They are less likely to suffer the consequences of being caught, as they are often a total stranger. As the external adversary has limited capabilities compared to the internal adversary, we concentrate on the capabilities of the internal adversary. If we can defeat the internal adversary we have defeated the external adversary as well.

If we want the firewall to protect the trusted network from the internal adversary, the firewall must have a full control of what is sent between the inside network and the outside. If we can somehow make the firewall reject all outbound packets that are not supposed to leave the inside of the firewall, then the adversary cannot make any unintended connections through the firewall. If the firewall can somehow retrieve information about the identity of the host, the application and the user, responsible for sending each network packet it receives, it has a chance of determining whether the network packet should be allowed to proceed to the outside or not [2].

Mainly we restrict our attention to IP-based traffic and assume that all traffic leaving the firewall uses the IP-protocol (Internet Protocol). Any IP-datagram sent over the network contains the sender and receiver IP-addresses, which uniquely identify the sending and the receiving hosts on that network [3]. Normally, we would agree that the firewall knows which host sent any given IP-datagram based solely on the IP-address of the sender. Unfortunately, our adversary is capable of faking an IP-address. This means that they can impersonate any host on the trusted network-including the firewall. Moreover, the IP protocol contains neither information about which application sent a given network packet, nor anything about the user that started the application in the first place.

Therefore we have observed that the firewall is not able to distinguish between the good and the bad network packets based solely on the information available in the IP-protocol-it needs more information. Still, IP is a very capable data transport layer and it is not likely to be replaced, at least not within the next few years [5]. We just have to accept that the information contained in an IP-datagram is not enough, we therefore need something additional.

Thus we want to ensure that no information leaves the trusted network, except when the communication link originates from a trusted source (a trusted host running a trusted application, under an authenticated user ID). This means that we must provide the firewall with this extra information in every single packet a

communication link consists of. Providing the extra information just in the first packet of a communication link is insufficient. The adversary is capable of hijacking an established communication link, due to their physical access to the trusted network. Also, providing each packet with just a user ID, application ID and host ID, is a very naive approach. Our adversary is capable of sniffing the trusted network and of sending/modifying packets, so duplicating the credentials of a valid packet is easy for them to do. We can already see that this requires changes to the operating systems of all hosts, on the protected network, that need to communicate through the firewall.

MATERIALS AND METHODS

The design of the advanced packet filter involves both the firewall and the protected hosts and contains several different technologies, strategies and algorithms. The method of design which we propose is briefly described here:

When a protected host needs to send an IP-datagram to the external network, it needs to send some additional information along with it to the firewall. The extra information needed is a host ID, a user ID and an application ID. Further we need a way for the protected host to determine whether an IP-datagram is destined for the external network. Only packets for the external network should be taken care of. This require the method to intercept the external packets in the kernel before they leave the protected host. This enables us to add an identification-token to the IP-datagram before it is sent to the firewall. We are describing this in next few paragraphs in this section.

In order for the firewall to determine whether the packet is authentic, i.e. it is sent from a protected host and has not been altered in transit, the protected host needs to authenticate each IP-datagram and identification token pair sent to the firewall. When an IP-datagram is intercepted on the protected host and has had the extra information appended, we need a way to transmit the extended IP-datagram to the firewall.

When the firewall has received an extended IP-datagram, it must determine whether the datagram should be allowed to pass on or not. To do this, we need to extend the firewall rules, since we have some extra information to base the decision on. Furthermore, if an extended IP-datagram is allowed to go through, we need some way to inject the original (unmodified) IP-datagram into the conventional firewall.

Identifying Host, User and Application: We have previously determined that each IP-datagram, sent from an internal host to the external network, must contain the identity of the sending host, the user and the application. We refer to this extra information as the identification token associated with the IP-datagram. Proving that the identification token is authentic is not a concern of this chapter; this is covered in a later chapter. In this section, we define the contents of the identification token in such a way that the sender of any authentic IP-datagram can be determined uniquely by the firewall.

As each IP-datagram has to contain an identification token, the size of the identification token has an impact on the bandwidth between the protected host and the firewall. Thus, we want the identification token to be as small as possible, while maintaining the property of being able to distinguish a large number of hosts, users and applications.

Host Identification: An IP-datagram sent from a protected host to the firewall already contains a host ID, namely the IP-address in its header. The information is obtained for free, as it is contained in each IP-datagram already. The IP-address of the sender uniquely identifies each protected host to the firewall. Dynamic Host Configuration Protocol (DHCP) is a protocol, which enables a host to obtain an IP-address dynamically from a DHCP-server upon boot time. This means that the host can have a different IP-address whenever it is rebooted. Therefore, we need to manage an enumeration of all protected hosts behind the firewall and initialise each host with its unique ID. 16 bits is a reasonable size of the host ID. It enables us to manage more than 64,000 hosts behind the firewall, which is enough for practical uses in a foreseeable future. The host ID should reside in a configuration file to which only root has access.

User Identification: A user ID on one protected host behind the firewall might not be unique for all of the protected hosts-this depends totally on the Intranet policy, but it is definitely unique for a single host. As long as we can identify the protected host uniquely, the user can be determined uniquely as well by using the host ID along with the user ID. The user ID must be large enough to contain the identity of all users within a single operating system protected behind the firewall. In the Linux operating system, a user ID is 16 bits, so for a Linux implementation, 16 bits is a reasonable size. This allows about 64,000 users.

Application Identification: When identifying the application responsible for sending an IP-datagram, we are interested in exactly what application we are dealing with, as a certain version of an application, started by a specific user on a certain host, might be allowed to communicate, while another version of the same application might not.

Apart from application input, an application is fully determined by the binary code the process is running. Therefore, one solution could be to send the binary code of the running process to the firewall for verification. At the firewall, the binary code would be matched to a set of registered application binaries to find the communication policy for this application.

The firewall and the protected hosts agree on an enumeration of all applications, which means that each application is given a unique identification number. The appropriate identification number will then be present in each IP-datagram sent by any application. This would, however, mean that we have to manage this list

on every protected host and at the firewall as well. Furthermore, we have to add tokens to those lists whenever new applications are installed. Distributing such a list should be done with care, as unauthorised changes to the file could prove disastrous. Another solution is to calculate a hash value for the binary code of the application and to use this value as an application ID.

Incorporating External Packet in OS: In previous section we have described that how the identification token is to be associated with each IP-datagram, which is needed in order to identify the host, user and application to the firewall. In next few paragraphs we describe how this information can be retrieved/ calculated in a modern operating system. Our suggestions for design so far tells us that we have a user-level daemon, which is responsible for calculating the identification token. Since a user-level process has no direct access to the kernel, we have to modify the kernel to supply it with the information it needs. IP- datagrams sent between the internal hosts do not need special handling, as they never cross the firewall. This means that somewhere, we have to distinguish between the IP-datagrams for the external network and for the internal network. As the operations are inherently connected with the operating system that is used, we describe how it is done in the Linux operating system, since this is the platform on which we implement our design. However, there should be no problem in making similar operations in most other operating systems. Thus in this section we actually describe two tasks. We describe how to divert an IP-datagram to a user-level daemon and we describe how we decide which packets actually pass the firewall and should therefore be diverted to the user-level daemon. In the literature and similar work carried out in this area the description regarding a utility which can solve this issue is available. This utility which is a patch i.e. called divert sockets for the Linux kernel [3]. It immediately solves the how issue, since it is designed to let us pass IP-datagrams from kernel to user space.

If we install an enhanced firewall, the individual hosts have to know more about the network, since any host must be able to decide whether a given packet has to pass through the firewall or not. In particular, they have to know which one is the firewall, since this is where IP-datagrams should be sent after they have been diverted. The divert sockets allow us either to divert packets destined for one or more hosts/networks and nothing else or not divert packets that are destined for some hosts/networks but everything else.

To use divert sockets on the clients, we must have firewall capabilities compiled into the client's kernel. On another operating system, the diversion may very well be implemented in another way, or one might even have to implement it from scratch. We find the divert solution very flexible and easy to use for our purpose- especially since we can easily configure the diversion from user space through ipchains. We must have super-user privileges in order to configure diversion and the application that opens the divert socket must also be run with super-user privileges. This helps guarantee that no ordinary user can circumvent security, by creating his own daemon to read from the divert socket.

Thus solution for both the how and which issue is available, which means that we know which packets will have to pass the firewall and how to divert these packets. However, we still have to supply some extra information from the kernel when the packets are diverted.

Data Packet Detection: There are several types of computer virus classes: viruses, Trojan horses, worms, hoaxes, jokes etc. Among these types, a virus is a piece of code that adds itself to other programs and cannot run independently. However, worms are programs that can run by themselves and propagate a fully working version of themselves to other machines. As Microsoft Windows became popular, windows viruses and windows- application-derived viruses using VBA (Visual Basic for Applications) spread widely. Moreover, a common way of windows virus prevalence is through emails. The recent important one was Code Red. The Code Red worm is a malicious self-propagating code that spreads surreptitiously through a hole in certain Microsoft software, such as Internet Information Server (IIS) Web software and the Windows NT and Windows 2000 operating systems. To identify viruses/worms in e-mail attachments, one exploits that data packets have a unique character, the virus signature. In addition to the usual network control ability of a firewall, data packet detection is necessary in the intelligent firewall to identify packets containing malicious data: The virus signatures are also appearing in data packets. For instance, the beginning of the Code Red's attack packet looks like the following:

Network-based IDSs also monitor network traffic on the wire for specific activities or signatures that represent an attack [7]. Strengths of IDSs are to monitor a large network and to have little impact on an existing network [1]. Moreover, they detect malicious and suspicious behaviour in true real-time and provide faster response and notification to the attack at hand. They examine all packet headers for signs of malicious and suspicious activity and can also investigate the content of the payload. They use live network traffic for attack detection in real-time and a hacker cannot remove this evidence once captured. However, a weakness of IDSs is their inability to process all packets, which can lead to possibly failing to recognise an attack during high traffic and the need to analyse packets quickly. Performance problems, especially with increasing network speeds and resource exhaustion problems can lead to difficulties [6]. Resource exhaustion problems can occur when an IDS must maintain attack- state information for many attacked hosts over a long period of time. It is also possible to misunderstand normal traffic as malicious traffic. Many approaches can trigger numerous false positives, because of normal traffic looking very close to

malicious traffic. In addition, a network-based IDS does not control the network or maintain its connectivity. Hence these systems are vulnerable to DoS (Denial of Service) attacks.

Deriving Identification Tokens in OS: When the user-space daemon receives a diverted IP-datagram from the kernel, it needs to create an identification token for the IP-datagram. It therefore has to know the host ID, user ID and application ID.

As described earlier that the host ID is an enumeration of all hosts. Each host is given a unique ID at installation time, which is read by the user-space daemon from a configuration file to which only root has access. Obtaining the host ID is therefore very simple for the daemon, since it is run as root. When the user-space application receives the diverted IP-datagrams, it has no way to derive the user or process ID that caused a given packet to be sent. As the actual diverting takes place in the kernel and the kernel holds this information, we need to change the diverting code in the kernel. We want the diverting code to send the user and process ID along with each IP-datagram to the user process.

The diverting code in the kernel might use the following kernel function to obtain the ID of the process responsible for sending any given IP-datagram:

```
static inline struct task_struct * get_current(void)
```

It returns a pointer to the current running task/process, which holds several items of information including the user and process ID. So, when diverting a packet to a user process, we could simply retrieve the current task struct pointer and ask it for the current user and process ID. Unfortunately, when the diverting code of the kernel runs, we cannot rely on the current task to be the process responsible for sending the IP-datagram. Each IP-datagram sent might reside in buffers for some time, before transmission, or it might get retransmitted at a later time. The current process could therefore be some random process that was running when the kernel decided to actually send/retransmit the packet. We are therefore forced to look for the information in the containers used by the kernel to hold IP-datagrams that are to be transmitted.

All network packets allocated in the kernel, including IP-datagrams, are held in a structure called sk_buff (short for socket buffer). The sk_buff structure holds all information the kernel needs about a network packet in order to process it. In particular, it holds a pointer to a second structure called sock (short for socket). Whenever a user process wants to communicate on the network, it needs to create a user level socket, which is an interface to the kernel. When the process creates a socket, it identifies the type of protocol it wants to use by a call to the following function:

```
int socket(int domain, int type, int protocol);
```

At user level, the socket number returned by this call is simply an integer, but in fact it is a handle to a complex kernel structure. Whenever a process transmits a packet, the kernel's sk_buff has a pointer to the sock structure, which was allocated by the above call to socket. This means that when we are about to divert a packet, we can follow the sock pointer in the sk_buff structure that holds the packet. The sock structure is our link to the process that created it and thus to the process responsible for sending the packet.

Enhanced IP-datagram Authentication: In the section we presented what extra information the firewall needed, in order to make a decision about whether to forward a packet or not. However, it is not sufficient just to supply the extra information. In our problem specification we state:

Providing the extra information just in the first packet of a communication link is insufficient. The adversary is capable of hijacking an established communication link, due to their physical access to the trusted network. Also, providing each packet with just a user ID, application ID and host ID, is a very naive approach. Our adversary is capable of sniffing the trusted network and of sending/modifying packets, so duplicating the credentials of a valid packet is easy for him to do. Thus it means that the authentication of each IP-datagram and identification token pair sent is a necessity.

In our problem specification, we stated the necessity to authenticate each packet from a trusted machine to the firewall, which in turn meant adding extra data to each IP-datagram transmitted to the firewall. In this section, we discuss the various options available when it comes to adding data to existing packets. We are here presenting some investigations carried out in this area about how much authentication data can be send within or along with a packet traversing the network, with a minimal bandwidth overhead. Various approaches are available in the literature and research carried out in this area. We found the approach in which the ad-hoc IP-options is encapsulated into IP-header datagram. Thus we suggest this approach only.

The recommended approach is to encapsulate ad hoc IP-options into the IP-header of the existing IP-datagrams and thereby supply the authentication data.

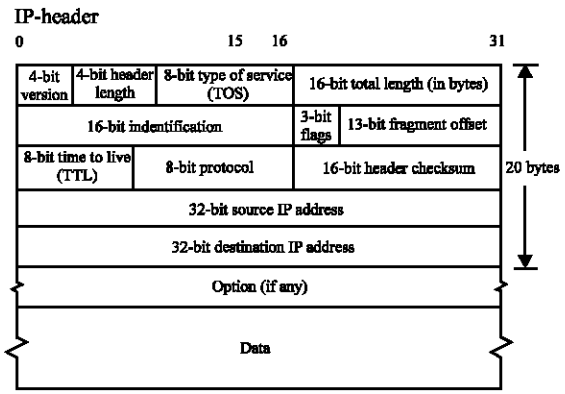

Fig. 1: IP-header

Encapsulating IP-options: The approach which we recommends is a rarely used feature of the IP-protocol. The IP-header supports a set of IP-options, which provide for control functions needed or useful in some situations, but unnecessary for the most common communications.

The above Fig. 1 describes the various fields in the IP-header. Our main interest is the IP-options field.

The IP-header has a 4-bit header length field, which measures the length of the IP-header in 32-bit words. The option part of the IP-header is the only reason for the IP-header to have variable length. Without IP-options the IP header length has a value of 5 (5*4 = 20 bytes), which is the most common value, as IP-options are rarely used. As the header length field is a 4-bit value, it is possible to specify a maximum header length of 15 words (15*4 = 60 bytes), which leaves a maximum IP-option size of 40 bytes.

The currently defined IP-options include provisions for timestamps, security and special routing. Except for a few IP-options (the End of Option list and No Operation options), each option has a one-byte class/number field and a one-byte length field. The actual functionality offered by the existing IP-options are not of any interest for our current investigation. But we do have two important observations.

* There are a total of 40 bytes available for IP-options. For some IP-datagrams a number of those are used.
* We have learn that five bits are reserved for the IP-option number, which allows a total of 32 different options. We know that 25 options are used, which leaves 7 numbers available for ad hoc options.

Thus, inventing an ad hoc IP-option number and using two bytes for the option type and length tag, leaves us with anything from 0 to 38 bytes of authentication data for each IP-datagram, when no other IP-options are present.

Each IP-datagram, sent from any application on a protected host to the gateway, is sent through an Authenticator module, which adds authentication tokens as ad hoc options inside the IP-datagram. This datagram is sent to the gateway/firewall, which in turn sends the datagram to a Verifier module. The Verifier module either discards the datagram or forwards it towards its final destination. Note that none of our ad hoc options leave our own network and that IP-datagrams traveling in the opposite direction are unaffected. The main advantage of this approach is that we only increase the size of the IP-datagram by 2 bytes more than the size increase imposed by the identification/authentication data size. 38 bytes (304 bits) seem more than enough room for identification/authentication data. Recall that our UMAC authentication token takes up 8 bytes and our identification token 14 bytes. So we can do with less.

CONCLUSION

As stated earlier in this study that attacks from the outside are protected against very well by common firewalls and are of no particular interest to us. In order for the firewall to protect against attacks from the inside, it needs more information than what is present in traditional IP-datagrams. This calls for some changes in the protected hosts, since they must provide this extra information to the firewall. Furthermore, we want to be able to trust the information in the packets, which calls for further changes to both the protected hosts and the firewall. We need to enhance the firewall with an extra module that screens all packets sent from the internal network to the external network: This module must enable the firewall to use this extra information when deciding whether to allow or deny any given packet. It would be a nice feature if the module is able to screen packets without introducing changes to the rest of the firewall. The common firewall does a remarkably good job at protecting against external attacks and we would not like our enhancement to require a re-implementation of the common firewall techniques.

We finally recommends that a good place to put our enhanced filter would either be immediately before or immediately after the incoming packet filter on the internal interface. Another solution would be to completely replace the existing filter with an enhanced one (a packet filter is a relatively simple part of a firewall). In this way, we can still use all the other components of the conventional firewall with minimal changes. Replacing the filter, or placing an extended filter on the inside (closer to the forwarding module), is probably more complicated, since the forwarding module might use information gathered in the incoming packet filter. We therefore place the packet filter immediately before the existing packet filter (closer to the network interface). This should enable us to use the existing firewall without any changes. In the enhanced packet filter, we can make a decision on whether any given IP-datagram may proceed based on the sending host, user and application, before it is sent through the conventional firewall, which handles the IP-datagram on the rest of its journey.

REFERENCES

1. Bace, R. and M. Peter, 2001. NIST special publication on Intrusion Detection System. NIST (National Institute of Standards and Technology) Special Publication, pp: 800-31.
2. Browne, H., W. Arbaugh, J. McHugh and W.L. Fithen, 2001. A trend analysis of exploitations. In Proceedings of the 2001 IEEE Symposium on Security and Privacy, pp: 214-229.
3. Divert Sockets for Linux (online at http://www.anr.mcnc.org/~divert/)
4. Frantzen, M., Kerschbaum, Schultz and Fahmy, 2001. A framework for understanding vulnerabilities in firewalls using a dataflow model of firewall internals. Computers and Security, 20: 263-270.
5. Lyu, M.R. and L. Lau, 2000. Firewall security: policies, testing and performance evaluation. In proceedings of the COMSAC. IEEE Computer Society, pp: 116-21.
6. Mchugh, J., A. Christie and J. Allen, 2000. Defending Yourself: The Role of Intrusion Detection Systems. In: IEEE Software, pp: 42-51.
7. Richards, K., 1999. Network Based Intrusion Detection: A Review of Technology. In: Computers and Security, 18: 671-682.
8. Xu, J. and M. Singhal, 1999. Design of a High-performance ATM firewall. In: ACM Transactions on Information and System Security, 2: 269-294.