High Performance Internet Connection Filtering Through an In-Kernel Architecture

Naser Ezzati Jivan, Alireza Shameli Sendi, Michel Dagenais
Computer and Software Engineering
École Polytechnique de Montréal
Montreal, Canada
e-mail: {n.ezzati, alireza.shameli-sendi, michel.dagenais}@polymtl.ca

Naser Nematbakhsh
Computer Engineering
University of Isfahan
Isfahan, Iran
e-mail: nemat@eng.ui.ac.ir

***Abstract*—A firewall is a tool that protects users and applications from unauthorized accesses and network attacks, and secures network connections and resources. It rejects unauthorized access while permitting authorized connections based upon network security rules and policies. Although the importance of a firewall in securing a network is vital, a poor architecture and inefficient mechanism for inspecting network traffic may lead to reduced network performance. Therefore, the performance of a firewall is considered as one of its main characteristics. Several methods have been proposed to increase firewall performance. In this paper, an in-kernel architecture has been proposed. It changes the structure of application proxies and moves a portion of their functionalities to the operating system kernel level. This kernel proxy inspects and filters the connections passing through the firewall with the help of a user daemon. Tests under different loads show that the performance of the firewall increases with the proposed architecture. The main reasons are the reduction of context switches and elimination of extra copies between kernel and user space. The Kernel proxy supports the HTTP, FTP and TELNET protocols although a better performance could be reached using a kernel URL filter.**

Keywords-firewall; proxy; content filter; kernel proxy; performance.

I. INTRODUCTION

Connecting a local private network to the global public networks facilitates the communication between internal staff and outside clients and suppliers. However, some remote users may attempt to gain access to computers on the local private network for purposes such as stealing or destroying valuable company information, vandalism or even extortion.

A simple solution for local network security is to run a firewall, protecting it from unauthorized Internet accesses. However, the associated level of protection obtained depends on the chosen architecture and type of the firewall. Firewalls can be divided into two general types: packet filters and proxies.

Packet filter firewalls filter packets based on examining the source and destination addresses and ports. They examine the packet's IP (and/or TCP) headers and accept or reject the packet according to the firewall filtering policy. In this organization, packets are inspected at the network layer and then sent to the destination.

Proxy firewalls are protocol-aware firewalls. They use the packet application layer content in order to decide to accept or reject the packet, providing a more precise control [6].

A Proxy firewall may have several application proxies. In order to protect the local network, connections are usually made indirectly through these proxies. They play the role of a mediator to transfer data between the external and internal networks [6].

In an application proxy, each packet must come up to the application layer, be analyzed on that layer and ultimately it is rejected or sent to the destination. There is a risk for the application proxy to slow down the transfer of data and cause a bottleneck in the network. Therefore, the design of an application proxy should be such that it is highly efficient and has a reduced impact on the speed of data transfer between the local and global networks. The purpose of this paper is to discuss a novel method for increasing the efficiency of application proxies by moving some parts of the functionalities to the kernel level. The idea is based on splitting up the firewall responsibilities between the user and kernel levels. The kernel level tasks include some of the standard proxy tasks - authentication, rule management, and state management. The application level tasks include rule-based, detailed data and packet level analysis. The main benefits of the revised firewall architecture come from savings in the number of context switches needed to process each packet sent and a reduction in the number of copies between user and kernel space.

The paper continues in the next section with a discussion of related work for increasing the firewall performance. This is followed by a section describing the architecture of in-kernel proxies and differences with the Linux Netfilter and other application layer proxies.

 ISBN: 978-1-61208-125-0

Subsequently, we will discuss our method in detail and will show how it can increase the performance of a firewall. Finally, the paper concludes by identifying the main features of the method and possibilities for future work.

II. RELATED WORK

Different methods have been introduced for improving the performance of firewalls. The first and oldest method to increase the efficiency of a firewall is called state-full filtering [3]. In a simple stateless firewall, the inspection of a packet is performed separately from other packets. Thus, whenever a packet comes to the firewall, regardless of whether it belongs to a previously setup TCP connection or not, or the state of that connection or other packets, the firewall will analyze the packet according to the rules, and reach a decision for it. The analysis and investigation of all the packets causes a severe performance cost to the firewall. Therefore, keeping track of the state of the connections can speed up the analysis of the packets and thus increase its efficiency. In this way, only the first packet of a connection is verified against the firewall rules, and little verification is required for the remaining packets [3].

Another method to increase the efficiency of a firewall is the use of caching in the application layer. With this mechanism, the results of recent user requests are saved. In the case of a repeated request (e.g. Web page or DNS entry), instead of creating a new connection to the server, the saved pages in cache memory are used. This will decrease the response time and use of the network bandwidth. This mechanism has been used in the Squid application proxy [5].

Fall [13] introduced another method to speed up the copying between two sockets and two files. Hence, this method can increase the data transfer speed and efficiency, in comparison with application proxies which only copy and transfer data.

In 1998, at the IBM research center, Bhaqwat [14] introduced another method called connection link. This method is based on the separation of control and pass-through of the proxies and proposes a fast route to transfer the data in pass-through mode. The main idea of connection link is that one should determine when a proxy moves from control to pass-through and then link the two separate TCP connections as a single connection [14]. Studies [1], [2], [4] and [7] suggest different kernel methods to increase the performance and efficiency. Gopinath [9] discusses kernel support for firewalls. Knobbe et al. [6] propose high performance architecture for network firewalls. Most of the aforementioned methods are based on separating the control and pass-through modes which is not cost effective and can't be accomplished completely. This paper introduces a method that increases the performance of firewalls to an acceptable level, without having to isolate the control and pass-through modes.

III. IN-KERNEL PROXY

When two clients of a network are connected to each other through a proxy, this proxy mediates the connection and controls data transfers between these two clients. The proxy checks the authorization of these clients and decides whether these two clients can be connected to each other or not, and if the connection is permitted, the data being transferred between them is controlled by the proxy [10].

Generally, proxies act in two following modes [7]:

- Control mode
- Pass-through mode

In control mode, the proxy performs some analysis on the data before it is accepted and transferred. When the control phase is over, the proxy moves to the pass-through mode in which, the proxy only passes the data. After the data is transferred, the proxy may turn back to the control state. For example, a TELNET proxy starts in control state considers and analyses a TELNET request and whether it is permitted or not. When it is permitted and the connection is made, the proxy changes to pass-through mode and copies the data between the two ends of the connection. Distinguishing these two modes and moving from one mode to another is the most important task in the improvement of the proxies' performance [4].

The functionalities of various proxies are different in the control mode. These vary from simple control, at the connection starting time, to continuous analysis of data being transferred during the connection. The proxies can be categorized into four groups according to the amount of processing that they do.

The first group of proxies performs a very little control. They are in control mode only at the time of connection start, and then remain in pass-through mode until the end of the connection. An FTP proxy is an example of this type. In the FTP protocol there are two kinds of connection, control connection and data connection. The FTP proxy processes an FTP request in control mode and connects the two clients through a new data connection. Then, the proxy processes the data connection, in pass-through mode. It keeps this mode until the end of the connection. The control connection stays in control mode in order to process the subsequent requests and commands [3].

The second group of proxies performs a large amount of control. These proxies authenticate the user, and the connections remain in control mode for all the data being transferred in both directions. One example of these proxies is the HTTP proxy that lets the user have access to the HTTP servers on the Internet. HTTP proxies can refine and modify the access of external agents or users by filtering and constraining permitted agents or users, and also it can modify and refine the response to internal requests through deleting unsecured applets (as JAVA applets and other potentially malicious codes) [7].

A firewall may combine different proxies, each of which controls a specific aspect of the transfer or exchange of data between two networks or clients. Typically, a proxy receives a connection, authenticates the client or user, and possibly after having refined and modified the request, passes the data to another network or client. The firewall either uses its own IP address to be an intermediary between the two ends of the connection, in

 ISBN: 978-1-61208-125-0

which case it is called classical proxy, or it is hidden from both ends of the connection, in which case it is called a transparent proxy.

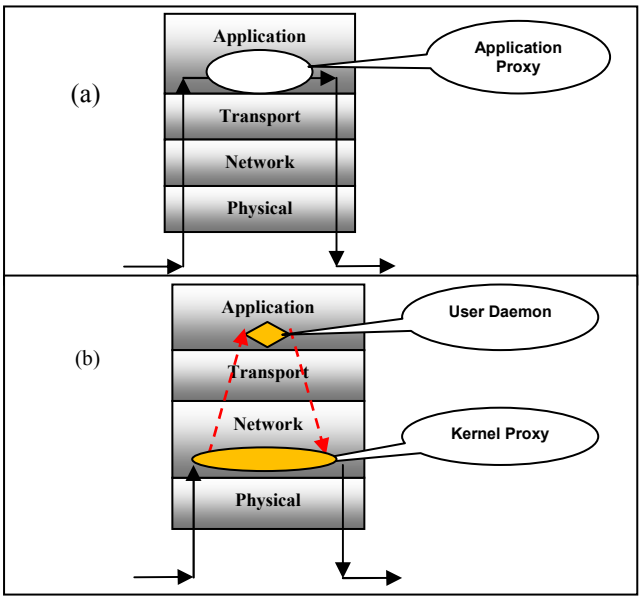

Figure 1. Path of Packets: (a) in a normal proxy (b) in the proposed in-kernel proxy.

An important point is that in many applications only a small part of packet (usually in the header) is controlled, and the proxy is mostly in pass-through mode. However, here a packet should also move from the kernel layer to the application layer and return to the kernel layer again before being transmitted to the destination. Therefore, the transfer of a considerable percentage of data from kernel layer to the application layer seems inevitable but wasteful [8] (Figure 1 (a)).

However, it is possible to design a new in-kernel structure for proxies that can solve the problem of unnecessary copying of packets from kernel layer to application layer, decreasing the number of context switches, and potentially greatly improving the efficiency. In this method, the proxies will have the responsibility of controlling the packets of the connection in the kernel network layer, and will avoid the transfer of data to the application layer. Thus, many cases of copying between kernel layer and application layer will be avoided and the transfer and efficiency of the firewall will be increased. Figure 1 (b) depicts this organization.

Of course, completely moving the proxy functionalities to the kernel may complexify the proxy structure. As a consequence, this could decrease of efficiency of the firewall. A solution is only moving the necessary and convenient parts of proxies to the kernel, keeping the remaining part in the application layer. This user-level part is shown in figure 1(b) as user daemon.

IV. . DESIGN OF THE IN-KERNEL PROXY

The in-kernel proxy is composed of two parts: one central part in the IP layer and one part in the application layer. The location of the in-kernel proxy in relation to the packet filter (Linux Netfilter) and IP layer is shown in figure 2.

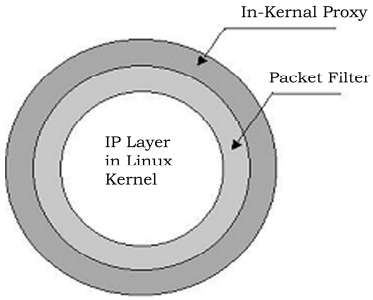

Figure 2. The location of the in-kernel proxy

As shown, whenever the network layer receives a packet, it is first analyzed in the packet filter of the firewall and then given to the in-kernel proxy.

Similar to the Linux Netfilter, the in-kernel proxy implementation is divided into two sections, a kernel module and a user daemon that interfaces with users and creates the rules and interactions.

However, the in-kernel proxy is a content filter which means that it has knowledge about the higher level protocols and inspects the packet based on its data part as well as its TCP/IP headers, while Netfilter is a packet filter and can just filter the network packets based on the TCP/IP header of a packet and not the data content [15].

Unlike the proxies in the application layer, where there is a specific proxy for each protocol, in the in-kernel proxy, a common proxy is set for all protocols. Of course it will have to distinguish the protocols in some parts of this proxy. However, all proxies are designed within this single framework. The proposed in-kernel proxy supports the HTTP, FTP, and TELNET protocols. However, due to memory and resource limitations of the kernel, the main usage of the in-kernel proxy is for URL filtering. This is an important difference between a complete and comprehensive proxy like SQUID and this in-kernel proxy.

With respect to the responsibilities of the proxies, and also the characteristics of the operating system kernel, the kernel-based proxy is composed of the following modules:

- Authentication
- Rule Manager
- Connection Manager
- Connection Filter
- Log manager

Each module has specific responsibilities and duties, being described in detail in the following sections. The relations among these modules are shown in Figure 3.

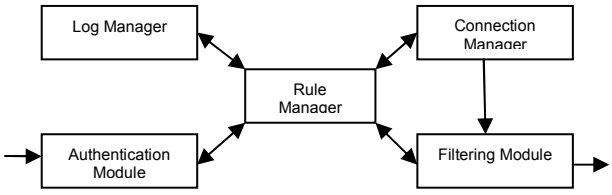

Figure 3. Architecture Elements

 ISBN: 978-1-61208-125-0

A. Authentication Module

In the kernel-based proxy, the authentication of the client is used to authenticate the users and clients. In this module, the client or user is identified once in order to check its permission to use the proxy and then all the connections and delivered packets are permitted for a predefined time duration. For authentication, users must connect to a specific port in the firewall and authenticate themselves by entering their username and password.

B. Rule manager module

This module is responsible for management and maintenance of inspection rules. The rule manager is one of the most important and active parts of the kernel proxy. This module receives the rules from a user interface, stores them to become available to other parts such the authentication, log manager, connection filter module. This module is also responsible for receiving upgrades of these rules and keeping the consistency. The rule manager is thus somewhat related to all the modules as a database in which all modules get their operation instructions and orders.

In the kernel proxy, the rule manager is composed of two parts: application layer and kernel layer. This module receives the rules from the system administrator through a user interface. Administrator writes the rules in synthetic forms, and the user interface, having received these rules, parses them to a canonical form saved in the data structures and files. The application layer part of the rule manager checks these rules and after deleting possible errors, copies the rules to the kernel layer.

C. Connection Manager Module

The responsibility of this module is to identify the protocols, classify the packets in the kernel, and then create and manage the table of different states of the proxies. In other words, this module identifies the protocols type of the packets, and accordingly reads the content of the packet and adds it as a new entry in the states table. For example, for HTTP packets, this module extracts the HTTP request and response contents and puts them in the proxy state table. All the proxies use the same state table. The data stored in the state table are as follows:

- Requested URL or address
- Client and Server address and port
- Protocol type (HTTP, FTP, TELNET)
- Connection state (connected, known ID, unknown ID, filtered, waiting, …)
- The list of the rules taken before for the connection packets
- Connection timeout value
- Bytes sent and received
- The policy applicable to the packets
- Display of the whole saved packets

D. Filtering Module

The duty of this module is to filter the data, if required-by different filtering rules for the packets passing through. Its responsibilities are as follows:

- Receiving the packets from the connection manager module: in the previous section it is mentioned that the connection manager module reads the packets, determines their protocol and then classifies them in accordingly. The filter module receives its own input data and packets from the connection and consults with the manager module to perform the required type of filtering.
- Inspecting the connection packets through the rule manager module: this module having received the data of the connection, according to the required filter, sends a request to the rule manager module concerning the characteristics of the connection and the related data. The rule manager having analyzed the request responds to the filter based on its database of rules. Moreover, the rule manager extracts the type of logging needed for that request and performs logging.
- Making decision about the packets: the filtering module decides according to the response from the rule manager to reject or accept the packet and to continue the processing. If the response is to wait, the filter saves the packet until the response from the rule manager is available. The rule manager may also consult with the daemon in the application layer for the decision.

Three types of filters to be used by the filtering module:

1) Command filter: this filter is used for HTTP and FTP. Here, the commands are received from the packets in the connection, and compared to the rules inside the kernel. The commands in the FTP proxy are “get”, “put”, “dir”, and “pass”, while in the HTTP proxy they are “get”, “head”, “post”, “connect” and etc.

2) URL filter: the filter is used for the HTTP and FTP protocols. To use this filter, the extracted URL from a connection is comparedwith rules inside the kernel. The database for inspecting the URLs is located in the user level agent but there is a small URL database cache in kernel which helps the kernel proxy to filter the most used URLs. However, for the URLs out of this cache, the kernel asks the user level agent to get the correct decision.

3) Content filter: considering the importance of the transferred files and data, it is possible that the rule manager force the proxy inside the kernel to analyze the content of some connections. Therefore, the files and data extracted from the packets are filtered and refined by the rule manager through the use of the content filter placed inside the kernel (and sometimes by connecting to the more elaborate content filter in the application layer).

E. The logging module

This module reports important events to the system log. There are three log methods (summarized, full log and no logging) from which the administrator can select the default method:

 ISBN: 978-1-61208-125-0

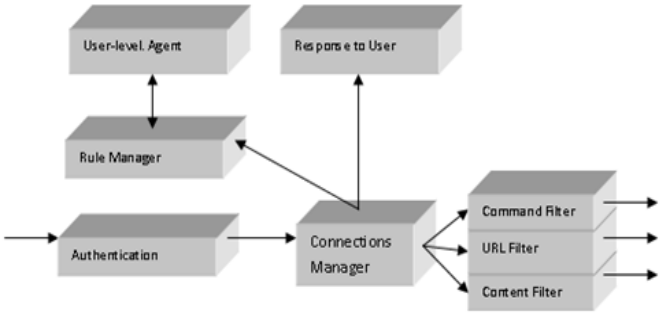

Figure 4. Execuition scenario of a HTTP kernel proxy based on the proposed architecture.

F. Architecture of the In-kernel proxy

In this part, the execution scenario of the HTTP kernel-based proxy is analyzed and reviewed in terms of proposed architecture. In general, the practical scenario of the HTTP proxy can be as follows:

- Analyzing user's authentication status: this is the first step in inspecting the connection. If the user has been authorized already, then the remaining packets are accepted. Otherwise, an "Unauthorized Access Message" will be sent to the user and the connection will be closed.
- Receiving the user's request and URL from the packet.
- Writing the new connection in the state table.
- Connecting and consulting with the rule manager and filtering the command and/or the URL and/or the content.
- Recording correspond events in the log.
- If the connection is valid based on the filtering rules, then a connection will be made with the distant host.
- Otherwise, a suitable response will be sent to the client and the connection will be terminated.

Figure 4 shows this process.

V. EVALUATION

The proposed kernel proxy architecture has been implemented in Linux Red Hat version 7.2 with kernel version 2.4.1. The network, hardware and software configuration of the evaluation setup are first presented. Then the influence of different parameters on the efficiency of the system will be analyzed. Finally, the results of the evaluation for the proposed system will be compared to the efficiency of an existing user-level proxy implementation.

A. Configuration

Table 1 show the configuration of the networks, clients, server and firewall machine (FwTest). In this configuration, two local networks with 100 Mbps bandwidth have been used.

TABLE I. THE CONFIGURATION USED FOR TESTING THE KERNEL PROXY

Host	CPU	RAM	HDD	NIC
Client	Intel PIII 800 MHz	256	Quantom 60G	1*3c905c 10/100
Server	Intel PIV 1000 MHz	256	Quantom 20G	1 *3c905c 10/100
FWTest	Intel PIII 800 MHz	128	Quantom 15G	2 *3c905c 10/100

In this configuration, proxies are transparent, the log method is summarized log and NAT (network address translation) is disabled. The number of rules in the kernel is 30 rules and the tests exercise the HTTP proxy. No cache mechanism is used in clients, server, or firewall.

Fire Bench [12] is used to generate network traffic and monitor the firewall performance. It measures the connections per second and the average response time:

- Connections per second: This test counts the number of the connections per second that are supported by the firewall. The clients make a connection to the server and immediately terminate it and start with a new connection. The number of clients increases gradually up to the point where a maximum is reached. SPECWeb2009 [11] is used to generate the needed connections and loads. Figure 5 depicts the comparison based on this test.
- The average response time test: This test is another criterion to evaluate the firewall performance; it is based on the average time needed to respond to each connection. Figure 6 depicts the comparison based on this test.

The evaluation is started by the execution of a load managing application, the manager, in the first client. This application reads the configuration file and produces the workload file. Then, it creates a TCP/IP socket to other clients and sends the workload and configuration files. At this point, each client waits for a message from the manager. When clients get the START message from the manager, they start to send their request to the server (through the firewall). After finishing the test, the manager gets the test results from the clients. This process runs 3 times and subsequently, the manager calculates the results and generates the final report.

B. Analysis of the results

Each normally terminated TCP connection is composed of at least seven packets [1]. In the kernel proxy, the first packet and also the packets that contain application layer content are analyzed and inspected. If they are sent to the application layer daemon, they need re-analysis by the user level rule manager. Therefore, the number of communications with the daemon and the number of rules, have a direct effect on the firewall performance.

The FireWall ToolKit (FWTK) from TIS is a popular user level application proxy [16]. The comparison between the in-kernel proxy and FWTK shows that the transfer of the firewall input traffic to the application layer causes a severe decline in the efficiency of the firewall (the

 ISBN: 978-1-61208-125-0

performance drop in the last part of the yellow line in figure 5).

In the case of the kernel HTTP proxy, only the packet that contains the URL is sent to the application layer daemon, while the remaining packets will pass through the kernel and be sent directly to the destination. Therefore the process is much simpler for many TCP control packets. FWTK on the other hand sends all the input data traffic to the application layer, thereby decreasing the efficiency.

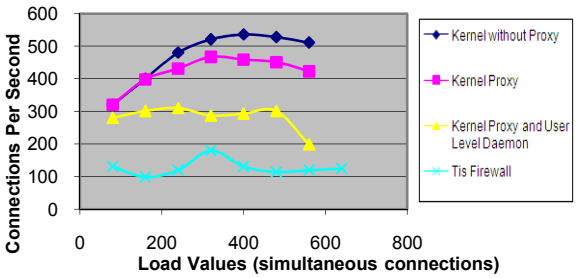

Figure 5. Comparison of the results (Connections per Second)

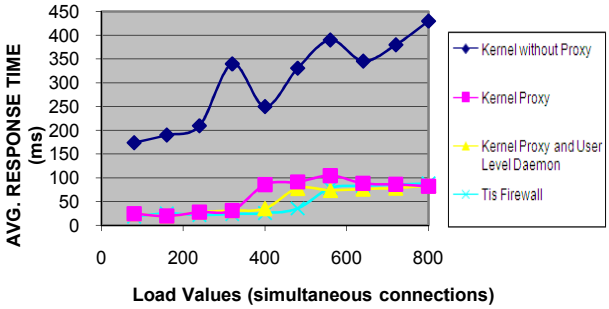

Figure 6. Comparison of the results (Average Response Time)

In the in-kernel proxy, a high efficiency is reached because of the pass-through of most TCP control packets.

Of course, the number of rules in the in-kernel proxy also affects the efficiency of the firewall directly. However, the influence of the number of rules is much less than that of the extra copies between kernel and user-level and associated the context switches. Figure 6 compares the average response time of kernel proxies (with/without user level daemon) and FWTK application proxy.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, the architecture and implementation model of an in-kernel proxy to increase the efficiency of the firewalls is presented. Results of the evaluation show that a proper division of labor between the application and kernel levels could yield substantial savings in terms of reduction of OS system calls and copied data. The efficiency increase compared to a kernel without any proxy and also compared to the efficiency of FWTK application layer proxy is shown. The main reason for the increase in efficiency of the proxy is due to the decreased number of contexts switches between the kernel and application layer and also the reduction of unnecessary copies among the different layers.

A complete URL filter proxy requires an application layer URL categorizer. However, in this project, a basic URL categorizer has been implemented; the implementation of a kernel URL categorizer could be a future extension.

The most important future enhancement will be implementing of a high performance kernel level packet modifier which inspects and modifies packets content, removing potentially malicious content (e.g. viruses and other malwares) from the packets. The important issues here are controlling the packets sequence number and the TCP sliding windows. This would lead to the creation of a light packet content filtering TCP daemon in the IP layer.

Finally, the kernel proxy should provide an application-layer interface to aid users and administrators in the configuration, rule editing and policy management of the firewall.

VII. REFERENCES

[1] Y. Zhang, Z. M. Mao, and J. Wang, “A Firewall for Routers: Protecting against Routing Misbehavior,” Proc. 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2007.

[2] J. Lee, P. S. Jean, T. Rick, M. G. Jack, and B. A. Bavier, “Network Integrated Transparent TCP Accelerator,” Proc. 24th IEEE International Conference on Advanced Information Networking and Applications (AINA 10), 2010.

[3] A. Rousskov, “On Performance of Caching Proxies,” North Dakota State University Fargo, 1999.

[4] R. Jain and T. J. Ott, “Design and implementation of split tcp in the linux kernel,” Doctoral Dissertation, 2007.

[5] S. Sahu, “Design Considerations for Integrated Proxy Servers,” Department of Computer Science, University of Massachusetts, 1999

[6] R. Knobbe, A. Purtell, and S. Schwab, “Advanced Security Proxies: an Architecture and Implementation for High Performance Network Firewalls,” Proc. DARPA Information Survivability Conference and Exposition 2000

[7] S. K. Adhya and S. Ganguly, “Asymmetric TCP Splice: A Kernel Mechanism to Increase the Flexibility of TCP Splice,” master thesis, Department of Computer Science & Engineering Indian Institute of Technology, April 2001.

[8] S. E. Schechte and J. Sutaria, “A Study of the Effects of Context Switching and Caching on HTTP Server Performance,” Available: http://www.eecs.harvard.edu/stuart/Tarantula/FirstPaper.html, [Oct. 29, 2010].

[9] K. N. Gopinath, “Kernel support for building network firewalls,” M.S. thesis Department of Computer Science & Engineering Indian Institute of Technology, April 1997.

[10] W. Shroder, Firewall and Internet Security, Prentice Hall, 1994.

[11] Standard Performance Evaluation Corporation (SPEC): SPECweb2009 Release 1.10, Oct 2009.

[12] KeyLabs Corporation: Test Final Report, Firewall Shootout Networld+Interop, 1998.

[13] K. Fall, “A peer to peer I/O system in support of I/O intensive workloads,” Ph.D. thesis, 1993, Department of Computer Science U.C.San Diego.

[14] S. D. Purkayastha, “Symmetric TCP Splice: A Kernel Mechanism For High Performance Relaying,” Department of Computer Science & Engineering Indian Institute of Technology, April 2001.

[15] The Netfilter Project site, Available: http://www.netfilter.org/ [Oct. 29, 2010].

[16] TIS FWTK Firewall site, Available: http://www.fwtk.org [Oct. 29, 2010].

 ISBN: 978-1-61208-125-0