The Design and Implementation of Transparent Application-Layer Filtering Platform

Yi-Shing Lee and Wei-Ru Lai*

Department of Electrical Engineering, Yuan Ze University, R.O.C., Taiwan

Abstract: For a long time, firewalls have played an important role in network security, protecting many of us against the attacks of malicious users. The implementation of firewalls can be classified into two categories: packet-filtering and proxy-based. Packet-filtering firewalls have gained enormous popularity because of the high performance and easy deployment features. However new generation of network attacks (worms, viruses etc.) have penetrated the protection of the traditional packet-filtering firewalls. Application-layer firewalls (traditionally called proxy servers) are given increasing attention recently. The two weaknesses, poor performance and complicated deployment procedures, have hindered the spread of application-layer firewalls. Powerful hardware can be adopted, like ASIC, to greatly improve the performance but complicated deployment roots in the congenital inability of many network protocols. To solve the deployment difficulty, the paper first discusses the concept of transparent deployment and implements a protocol-independent platform for illustration. On this implemented platform, firewall programmers simply focus on the development of application-specific filters while rest of the remaining hard task is taken care of by the platform.

Keywords: Packet-filtering firewall, proxy-based firewall, application-layer firewall, transparent deployment, application-layer protection, filtering platform.

1. INTRODUCTION

Global enterprises rely heavily on the Internet for business communication and information exchange. To balance the requirements of communications and the business privacy, firewalls are deployed between Internet and the corporate networks. Firewalls [1] protect the corporate networks from viruses, network attacks, or even malicious software. For example, the malicious programs corrupt important files, cause malfunctioned operation of business servers, and even take advantages of these servers to attack other public servers on the Internet. Firewalls are most oftenly adopted to block many known vicious attacks, notify system administrators of the latest alerts, and generate effective reports for analysis.

However, as more attacks occur on application layer, such as the design flaws of application protocols or the vulnerability of application software (malicious ActiveX and Java Applet code), traditional packet-filtering based firewalls are unable to detect viruses, worms, junk emails (SPAMs) and other application-layer attacks. The main reason is that packet-based firewalls perform inspection solely on a single packet. The capacity of a single packet (payload) is pretty limited. If malicious code is transmitted over multiple packets, for best accuracy, firewalls need to assemble the payload of multiple related packets, restore the original data stream and then perform inspection on the restored data stream. The payload assembly and inspection (Deep Packet Inspection) technique is commonly used in many commercial IDS/IDP (Intrusion Detection/Intrusion Detection and Prevention) products. The inspection task is even more challenging when many application protocols on the Internet rely on more than a single socket connection (e.g. multiple communication channels are created), such as FTP, instant messaging (IM) applications, multi-media streaming, popular peer-to-peer (P2P) protocols and even VoIP communications. In many cases, the application data is not transmitted in the original binary format (e.g MIME). Merely inspecting the payload of multiple related packets is not enough. That is the reason many commercial products with DPI technology have already reached the bottlenecks and result in the inaccuracy on the detection of many network attacks.

Technically, only a proxy is able to fully comprehend the operation logic (protocol) of an application. Since a proxy understands the format of the transmitted application data, for the best inspection accuracy on viruses, worms, malicious code and application-layer attacks, many commercial security gateway products have already integrated limited proxy-based filters on their platforms. These proxy-based security gateway products integrate many well-known technologies formerly adopted in packet-filtering firewalls, proxies and IDS/IPS.

As soon as connections are created, proxy-based firewalls [2] begin the inspection task on the transmitted data. Since a proxy fully comprehends the format of application data, the proxy-based filters have no difficultly in filtering application-layer attacks. Besides, quite a lot of information can be logged for later analysis, such as the URLs of HTTP requests, the sender (or recipients, attachments) of emails *via* SMTP and POP3, the complete conversation dialogue of MSN connections. The application-layer records are an essential part of business security information management. Therefore the demand for application-layer inspection devices is unbelievably huge on the market. For market segmentation or differentiation, firewall venders have created many marketing terms, such as application-layer firewall,

*Address correspondence to this author at the Department of Electrical Engineering, Yuan Ze University, Tao-Yuan 320, R.O.C., Taiwan; Tel: 886-3-4638800, Ext. 2431; Fax: 886-3-4639355;
E-mail: wrlai@saturn.yzu.edu.tw

1874-1339/08

application-layer gateway (ALG), unified threat management (UTM) and unified security management (USM) [3] etc.

Application-layer firewalls [4], just like traditional proxy servers, possess two obvious weaknesses, poor performance and complicated deployment procedures. The two weaknesses have hindered the spread of application-layer firewalls. Since proxy-based firewalls have to create a bi-directional communication channel, inspect the application data and log essential information, all these additional features and processing overheads result in the poor performance compared to traditional packet-filtering firewall. As the hardware development greatly improves over the years and more acceleration techniques are developed, such as ASIC, network processor and multi-core processor, the performance issue is easily solved. As for the second issue, complex deployment procedures, resubetting and configuration on client software settings are the two major tasks for network administrators. It requires tremendous efforts to deploy a proxy-based firewall in a large network with hundreds to thousands of users. For example, re-subnetting or configuring static routes on relevant neighboring devices are often unavoidable. In general cases, proxy deployment requires the support of client application software, such as SOCKS settings in many client software programs. For example, users must configure the proxy or SOCKS settings in their favorite browsers (e.g. Internet Explorer, Firefox) when an HTTP proxy server is deployed. It is even a nightmare if no automated tools or procedures are available and network administrators need to personally configure the settings of client software for each and every user.

A proxy is positioned between a client and a server. The proxy sends a request to the server on behalf of the client therefore the server is not aware of the existence of the real client since all the requests come from the proxy. In some cases, that generates a security issue because the end-to-end "transparency" is compromised after the proxy is deployed. Network administrators often demand the transparent architecture of a firewall. The so-called "transparency" or "transparent architecture" [5] usually refers to the invisibility of a firewall device in the deployment stage. If a device (or a system) reduces the traditional complexity level of deployment task, the device is considered to satisfy some kinds of transparent requirements. For example, no more configuration task must be done on client software. Both sides of the communication should not be aware of the existence of the intermediate proxy. The transparent deployment also must not generate any security vulnerability. For example, once a proxy is deployed, the server loses the track of the real sources of the requests since all the connections are initiated by the proxy.

To our knowledge, there is no commercial product to solve the problem. Also, we have not found any research literature discussing on possible solutions. Therefore, this paper carefully studies the limitations of current transparent application-layer firewalls and proposes an intuitive solution. The idea is then realized on Linux platform as proof-of-concept. Besides ease-of-installation, the modified network platform provides a new set of system calls, based on the traditional BSD socket API [6], for firewall developers, so that developers can focus their efforts on the programming of proxy-based filters without worrying about other underlying transparent deployment issues. The network platform is deployed transparently, 1) No IP subnetting or configuration on neighboring network devices are required. 2) No configuration on the client software is required. 3) Both client and server are not aware of the existence of the intermediate application-layer filtering device. Besides that, software filters for any TCP application protocols are possible on this proposed network platform. Firewall developers simply devote the efforts on the programming of application-specific filters, such as HTTP filters for URL filtering, SMTP filters for anti-virus and anti-spam. The rest of the hassles are solved by the proposed network platform. The proposed network platform therefore offers complete transparency compared to current transparent techniques being used on the market.

The remainder of this paper is organized as follows. Section 2 describes related researches of firewalls. Section 3 details the architecture of a proxy-based firewall. Next, it presents the problems on current proxy-based firewall deployment. Section 4 provides solutions to the transparent deployment of proxy-based firewalls and demonstrates the implemented platform. Section 5 concludes this paper.

2. RELATED RESEARCHES

This section describes the related researches on firewalls. Firewall is the network device to separate two or more networks and provide security issues. It is often categorized as software-based system and hardware-based appliance. Whether it is a software firewall or hardware appliance, the physical location of firewalls is in-between internal (or trusted) networks, such as corporate networks, and external networks, such as the Internet. Legitimate users can access the public servers while invalid access attempts as well as malicious attacks are prevented.

The firewall categorization includes transitional packet-filtering firewalls, Stateful Packet Inspection (SPI) firewalls, proxy gateways, Intrusion Detection System/Intrusion Prevention (IDS/IDP) systems, ASIC-based firewalls and application-awareness L7 firewalls (Anti-Virus, Anti-Spam, URL-Filtering Gateway). Among them, packet-filtering firewalls, SPI firewalls and IDS/IDP systems are based on the technologies of packet-filtering, and are often called generic packet-filtering firewall. Others are categorized as proxy-based firewall [7, 8, 9:837-843, 10:46-50, 11:15-17, 12, 13].

Two common techniques are commonly adopted by packet-filtering firewall, signature-based and flow-based anomaly. Signature-based refers to the inspection on the packet payload against known signatures or patterns, such as Sasser, Code-Red or HTTP buffer overflow attacks. Flow-based anomaly refers to that normal behavior of network activities is first gathered and any abnormal activities are considered attacks. Many commercial firewall products contain a signature set of 2,000~6,000 patterns and the signature set is updated daily. To increase the accuracy, some products assemble on the payload of multiple packets to form a stream of data for inspection.

Packet-filtering firewalls do not provide intelligent application filtering. For example, an IPS device is able to determine if a certain string is contained in a packet. The string may be a signature of a malicious worm. If the worm dynamically modifies the signature or the signature is randomly

Table 1. Comparisons of Packet-Filtering and Proxy-Based Firewalls

Firewall Type	Packet-filtering	Proxy-based
Performance	Excellent	Poor
Deployment	Easy	Difficult
Inspection Scope	Packet header or payload	Application layer data
Client Support	Not required	Yes, SOCKS or proxy support is required.

separated in multiple packets, the IPS device most often fails in detection. Since a limited number of packets are assembled for maximum system performance, if the application data is compressed or decoded in other binary formats, signature-based filtering would also fail. Human judgments are often heavily required because an IPS device is likely to generate many false alarms. For example, a packet carrying worms definitely contains a certain string but if a packet containing the certain string is not necessarily a worm packet.

Different from packet-filtering (or SPI) firewalls, proxy-based firewalls do not perform inspection on a per packet basis. Proxy-based firewalls do not simply forward IP packets to the final destination, but forward the request to a server on behalf of a client. Therefore in proxy architecture, no direct communication is possible between a client and a server. A proxy intercepts a client's connection, extracts the requests, asks the server for resource, and then sends the response of the server back to the client.

Since a proxy-based firewall forwards the application request to a server, two separate communication channels are maintained for every application request. The first channel originates the client to the proxy itself, while the other channel is created from the proxy itself to the server. A proxy-based firewall, besides source/destination IP address and port numbers, acquires lots of other essential information, which makes possible enforcing tighter security policy. For example, an HTTP-filtering firewall can determine "who" has the access privileges instead of the source IP address of a client host, "which" website the user is allowed to access instead of the destination IP address of a web server. Proxy-based firewalls operate on the highest layer of OSI model (Application Layer). In terms of security policy enforcement, proxy-based firewalls offer the highest level of security control. For example, content filtering is available on proxy-based firewalls, such as virus email, spam, malicious programs, pornography pictures and websites. User-based authentication and authorization are also supported on proxy-based firewalls.

Three drawbacks are brought by proxy-based firewalls. First, commercial proxy-based firewalls support content-filtering on limited application protocols (e.g. HTTP, SMTP, POP3, IMAP, FTP) and some popular instant messaging applications. Since only specific kinds of applications are supported, proxy-based firewalls are sometimes called application-specific firewalls or application-layer firewalls. For any new application, a proxy program or application filter must be customized. Second, an application filter on proxy-based firewalls is in fact both a client and a server, and complex content-filtering tasks require high computation power and memory capacity, poor performance is an issue. Third, proxy-based architecture requires the support of client software during the deployment stage. The configuration task on client software of every user is inevitable. The problem is worse if the number of users is up to hundreds or even thousands and it is even worse if the client software does not support SOCKS or proxy setting.

Table **1** shows a simple comparison of packet-filtering and proxy-based firewalls. To provide 100% protection from any malicious attacks, especially attacks on application protocols, proxy-based firewalls are the only solution. Therefore it is believed that proxy-based firewalls are the trend for the future. In next section, the paper focuses more on the study of proxy-based firewalls.

3. STUDIES OF PROXY-BASED FIREWALL

Proxy-based firewalls [2] are the best candidates for the detection of application-layer attacks and complete content filtering. For example, an HTTP proxy inspects HTTP requests for URL filtering or the integrity of downloaded files, while an SMTP proxy checks if any virus or malicious attachments are contained in an email. HTTP and SMTP are two of the most popular protocols on the Internet over the years. It is not a surprise that HTTP and SMTP proxy filters are often included in commercial firewall products and certain techniques are developed for the transparent deployment. Thus, in this section, the operation of transparent HTTP and SMTP proxy is studied in details. Also, the section reveals the reasons why proxy operation is only available for limited application protocols, and drawbacks of current transparent deployment techniques.

3.1. The Operation of Transparent HTTP Proxy

The section describes the current transparent deployment of HTTP Proxy, how it works and limitations. The basic operation for any proxy is almost the same. The only difference is how each proxy program handles the application specific tasks.

Basic proxy operation is a three step process.

Step 1: Proxy listens to a special port and waits for a client request.

Step 2: Proxy handles the client request and analyzes for essential information or signatures.

Step 3: Proxy initiates a second connection to the real destination server, forwards the client request to the server, and sends the server response back to the client.

Fig. (**1**) illustrates the transparent deployment of HTTP Proxy. Four roles are present in this transparent architecture, Client, Firewall, Proxy and Web Server. Many commercial firewall products incorporate proxy-based application filters onto the products as a single system. For better illustration, Proxy and Firewall are separate components here.

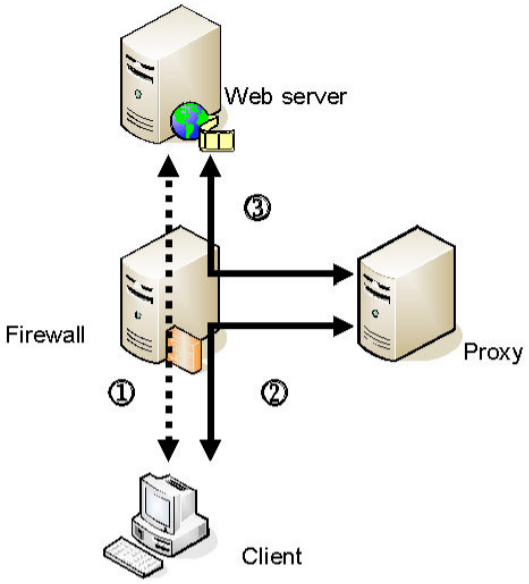

Fig. (1). Transparent HTTP proxy.

Consider that Client (denoted as C) wants to access Web Server (denoted as S) and then creates an HTTP connection to S (denoted as the dotted line 1, between C and S). However, Firewall (denoted as FW) intercepts this connection, modifies the destination address to P and then the connection is redirected (by routing decision) to P. Then, a socket is created between C and P (denoted as the solid line 2). FW restores the original address on all the packets replied from P. That is, after the packets from P are processed by FW, the source address of the packets is S. C believes a socket is created between S and itself while in fact the connection is created between C and P. The above technique is called Redirection. After three-way handshake is complete, C sends an HTTP request for a web page. In HTTP 1.1, the request header contains a "HOST" field, specifying the domain name of the web server. Based on this field, P realizes C is trying to connect to S. Then P would create a second connection to S (denoted as solid line 3). P sends another HTTP request to S (sometimes simply forwards the original HTTP request to S) and relays the replied HTTP Response back to C. The two HTTP connections (from C to P and from P to S) are remained open and are closed at the same time, which is sometimes called On-the-Fly.

The Redirection-based transparent deployment of HTTP proxy generates the following drawbacks. First, HTTP 1.1 must be utilized or the "HOST" field must be present in the HTTP request header, otherwise there is no way for the Proxy server to know the real destination address of the web server. Second, the end-to-end communication is not transparent. The Redirection technique makes it possible for the client believe it is communicating directly with the server, but from the server's point of view, all the connections are initiated from the proxy. If address-based access control is enforced by the server, the client would have problems accessing resources since the connection source is modified (e.g. from C to P). Third, the Redirection settings on the firewall must be carefully configured, or the Redirection Loop may occur. As shown in Fig. (**2**), if Firewall does not exclude the address of Proxy in the Redirection settings, the second connection initiated from Proxy would be redirected back to itself, forming a loop.

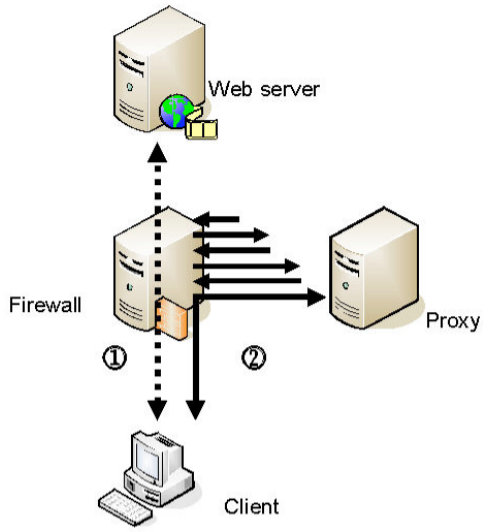

Fig. (2). Redirection loop of HTTP.

3.2. The Operation of Transparent SMTP Proxy

The Redirection technique is also widely adopted for building transparent SMTP proxy for the application-layer filtering. The architecture is shown in Fig. (**3**). If Client (denoted as C) would like to deliver emails to Server (denoted as S), C creates a connection with S (denoted as the dotted line). The architecture is shown in Fig. (**3**) and the operation of SMTP proxy-based firewall is similar to HTTP proxy-based firewall. C would like to deliver an email and is creating a connection. Redirection is done on the Firewall (denoted as FW) and the connection from C is redirected to Proxy (denoted as P). The real connection, which is redirected to P, is the solid line 2. The dotted line 1 represents the connection viewed by C. Then C follows the SMTP protocol to finish the email delivery and terminates the connection. Based on the command "RCPT TO" in the SMTP transaction, P is able to locate the real destined mail server (which is Mail Server, denoted as S). P creates an SMTP connection to S (denoted as the solid line in Fig. (**3**) (**b**) for the email delivery. Email delivery does not require real-time processing, therefore on the contrary to the previously-mentioned HTTP proxy operation, only one connection is remained open at the same time. This kind of operation is sometimes called "Store-and-Forward".

Several drawbacks are present in this transparent deployment of SMTP proxy. First, if ESMTP Authentication is required for email delivery, because P does not have a complete valid credential list, sender identity fails to be verified. In most cases, SMTP proxy is configured to simply assume all email senders are legitimate users, and then just skip the authentication stage. A security breach is created and that

results in a lot of spam abusing cases. Second, end-to-end communication transparency is compromised since the email server fails to know the real source address of the email sender.

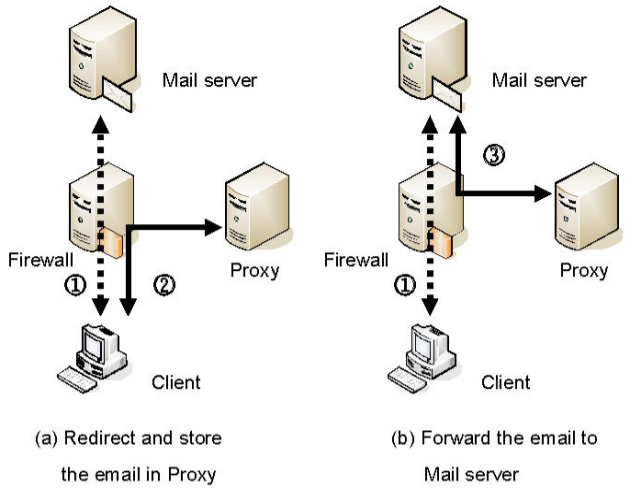

Fig. (3). Transparent SMTP proxy.

3.3. System Requirements

As shown in Subsections 3.1 and 3.2, the Redirection technique makes transparent deployment possible for both SMTP and HTTP protocols even though some drawbacks exist. However the Redirection technique is not suitable for most application protocols. If an application protocol does not contain any information regarding the real final destination, redirection technique fails. HTTP 1.1 requires "HOST" field is present in the request header for the proxy to locate the address of the original web server. The "RCPT TO" command in SMTP protocol makes it possible for the proxy to deliver emails to the final destination email server. If no destination address is available in the request message defined in an application protocol, the Redirection technique is not applicable for this application. Unfortunately many application protocols belong to this category, from the early application protocols POP3, IMAP, NNTP or FTP, to the latest IM, P2P and many online gaming protocols. Even though the information of destination address is available in the request messages, some problems remain to be overcome due to protocol limitations, e.g., ESMTP Authentication problem. Based on the above study, the Redirection technique is only applicable for limited applications. The Redirection technique fails to satisfy requirements of transparent deployment and thus is not a perfect solution. For developing a real transparent application-layer firewall, the following four system requirements are considered:

1. Transparency in Physical Network Installation

To install a new filtering device demands basic domain knowledge for network administrators, such as IP re-subnetting, routing principles. The installation of more complicated devices requires even more professional skills and advanced knowledge. To fulfill the "Physical Network Transparency" requirement, the device should possess the plug-and-play (PnP) feature for easier installation.

2. No Configuration Required on Client Software

Many proxy-based application firewalls require the reconfiguration on client software, such as the proxy setting in web browsers, the server setting in SMTP/POP3/IMAP client software, SOCKS setting in many IM software (MSN or Yahoo Messenger) and P2P software. In a large network with thousands of users, configuring all these client settings is a great challenge. If any client software does not support the use of proxy, then the proxy-based application firewalls would fail to operate.

3. Transparency in End-to-End Communication

By nature, the operation of proxy-based application firewalls is very similar to that of traditional proxy servers and possesses the same problem. The source address of the second connection is different from that of the first connection. Failure to identify a client's source address results in security vulnerability. Both sides of a communication (client and server) are supposed to be aware of the real addresses of each other.

4. Applicable for Almost Any Application Protocols

The traditional Redirection-based technique requires destination related information be included in request messages. But in fact few application protocols fulfill this prerequisite. That is the main reason why commercial firewall products only support transparent deployment for HTTP and SMTP protocols only.

4. IMPLEMENTATION OF PLATFORM

To fulfill the system requirements described above, the paper introduces a transparent application-layer filtering platform, on which software programmers can focus purely on the development of the application filters while the proposed platform solves most of the transparent deployment issues. Based on the system requirements discussed in Section 3.3, the proposed platform provides the corresponding solutions as follows:

1. The transparent application-layer filtering platform adopts the bridge architecture so that network administrators simply install the proposed platform in almost any preferred places with no need to modify any configuration of neighboring routing devices (such as subnetting, IP address or static route parameters). Also installation of any third-party client tools or any parameter settings (proxy or SOCKS etc) on the application software on the user side is not a requirement any more.
2. The proposed platform must not compromise the end-to-end transparency, which means both sides of the communication are aware of the real addresses of each other.
3. The platform provides an extended API, integrated with standard BSD Socket API, for firewall programmers to develop any add-on software filters with little extra efforts. Since most programmers are already familiar with BSD Socket API, the learning curve of the extended API is minimized as much as possible.

Linux is a very popular network operating system on which many commercial application-layer firewalls are based. The proposed platform is implemented on Linux for mainly two reasons. First, Linux is open source software under GPL. The source code of Linux kernel, system utilities and technical documentations are easily available on the Internet, free of charge [14-25]. Second, Linux has gained its popularity over the years. Many software developers already have experiences on Linux programming. The proposed platform modifies several modules (layer 2/3/4/7) of Linux kernel and offers an extended API (based on BSD Socket API) for firewall programmers in the development of application filters. Finally an HTTP application filter is developed as an example on this proposed platform for the demonstration purpose. The proposed platform is referred as transparent application-layer filtering platform throughout this paper.

4.1. Operations of the Platform

The software filters on the proposed transparent application-layer filtering platform is by nature a proxy-based architecture. If the second connection presents the same source port and source address as those of the first one, both two connections would seem identically the same. Both sides would not know the existence of an intermediate proxy server, thus the requirement of end-to-end transparency is reached. For transparent proxy (filters) developers, as long as the original socket pair information (source port, source address, destination port, destination address) of the first redirected connection is easily available, and the second connection can be masqueraded as the first one connection before Redirection takes place, the requirement of end-to-end transparency is fulfilled.

In terms of internal operation, the system architecture of the proposed application-layer filtering platform is shown in Fig. (**4**). The proposed platform is composed of three system modules: Bridge, Stateful Packet Inspection firewall (SPI) and Application Filters.

The bridge module (Fig. (**4**) (**a**)) retrieves the hardware address of an incoming Ethernet frame, according the internal forwarding MAC table, forwards the frame *via* the correct interface. In the example of Fig. (**4**), Ethernet frames (i.e., the Data Flow) received from the left network interface are forwarded through the right network interface. SPI module (Fig. (**4**) (**b**)) is a general firewall module, which provides connection tracking, determines the fate of packets (accepting or denying), and performs network address translation when necessary. The Application Filter (Fig. (**4**) (**c**)) is by nature a proxy daemon, which accepts a client connection, creates a server connection and performs content filtering or attack inspection upon the transmitted data.

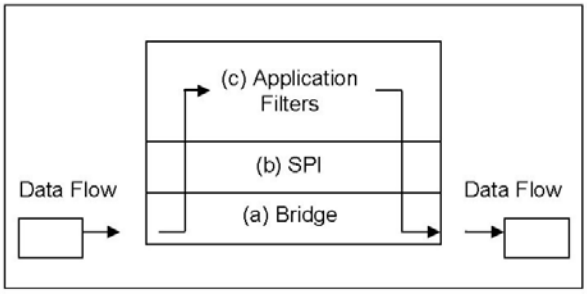

Fig. (4). Transparent application-layer filtering platform.

The process to create connections between client and server *via* this transparent proxy is simply described as follows. Many of the details are ignored here for easy understanding.

Step 1: A client would like to create a TCP connection to the server and sends out the first TCP packet (initial SYN packet). The packet, now in the form of an Ethernet frame, is received from the left interface and is passed to the Bridge module. The Bridge module realizes the frame contains an IP packet and passes the packet to the upper SPI module for further inspection.

Step 2: SPI module inspects the header, as well as the payload, of this packet. If Redirection is enabled, the destination address of the packet is rewrited and the packet is passed to the upper layer program (proxy daemon). If three-way handshake is complete, a TCP connection is successfully established between a client and the proposed platform.

Step 3: The proxy daemon retrieves the socket pair information of the original connection (before Redirection is performed), such as original source address (i.e., the IP address of the client), original source port and original destination address (i.e., the IP address of server) etc. The above operation, formerly impossible, is now available for application-filter developers with the help of the extended API set.

Step 4: The proxy daemon instructs the SPI module to masquerade the second connection so that the second server connection would have the same socket pair as the original client connection.

Step 5: Following the instructions, SPI module performs the network address translation and passes the packet to the lower Bridge module.

Step 6: Bridge module fills the correct destination hardware address and forwards the frame to the destination or the router for relaying.

Step 7: Bridge module passes the replied packet from the server, such as the second SYN+ACK packet or any other date packets, to the upper SPI module.

Step 8: SPI module performs the connection tracking and realizes that this packet belongs to an existing masqueraded connection.

4.2. System Description of the Platform

The proposed platform is implemented on Linux. Though bridging module, as well as firewall module, (known as netfilter) are already provided by Linux, much integration work remains to be done. The modified kernel modules and system utilities are briefly listed here:

1. Several software bugs in bridging and netfilter code are corrected, so that bridging module can forward packets the upper layer netfilter module for further process.

2. System utility tools, such as iptables, are modified so that application-layer content filtering can be dynamically enabled and disabled.

3. netfilter code is modified, so not only original socket pair information of any redirected connections is preserved, but complex NAT function is also available.

4. BSD Socket API is extended. Upper applications can retrieve the original socket pair information, instruct netfilter to perform address translation on the self-initiated connections. So that the socket pair of the second server connection is the same as that of the first client connection.

The proposed application-layer filtering platform provides:

1. Bridge mode deployment for plug-and-play installation.

2. By the help of netfilter, a proxy daemon intercepts any preferred connection through the platform (Redirection technique).

3. Software developers can retrieve the original socket pair information, such as the original source address and the original destination address. A program is able to change the socket pair of any self-initiated connections as wished.

4.3. Pseudo-Example for Developers

For firewall developers, as long as the original socket pair information is available and the masquerading action is possible on the second self-initiated connection, building a complete application-layer filtering firewall on the proposed platform is a piece of cake. The migration of original proxy-based filters does not require any huge modification. A pseudo example is provided below for reference:

```
/*Waiting for a client connection after Redirection*/
while (client_sock = accept_connection()) {
    orig_dst_addr = getOrigDst(client_sock);
    /* get the original destination address of this redirected
    connection, which is the real server address */
    orig_src_addr = getOrigSrc(client_sock);
    /* get the source address of this client */
    dst_socket = getSocket();
    /* prepare the socket for the second server connection
    */
    setSockOrigAddr(dst_sock, orig_src_addr);
    /* specify the source address of this socket to that of
    the client */
    dst_sock = connect(orig_dst_addr);
    /* Open the second connection to the server */
    proxy_action (client_sock, dst_sock);
    /* this function performs most of the content filtering
    work. */
}
```

4.4. Experiments on the Platform

The subsection proves the achievability of the proposed application-layer filtering platform. The platform is able to support proxy-based filters on any existing TCP protocols. To prove the concept as well as the extended API set, a generic proxy-based HTTP filter is written. The generic HTTP filter running the proposed platform, presents a full-featured application-layer HTTP firewall. The following experiments are conducted to simulate user browsing activities. Packet sniffing tools are utilized to collect experiment results.

Fig. (**5**) shows the basic network diagram and components used in the lab. Among them, Gateway is the proposed platform. Two proxy daemons are running on the Gateway, Squid and App_Filter. Squid is the most popular web cache proxy software, mostly on the Unix-like systems while App_Filter is the generic HTTP proxy developed based on the extended API. In the lab, many advanced features of Squid are temporarily disabled for simplicity. Therefore both Squid and App_Filter are served as the role of simple HTTP proxy only. The Gateway is equipped with two network interfaces (eth0 and eth1) and is running in bridge mode. A virtual bridge device fr0 is created with an IP address 192.168.6.123, whose member devices are eth0 and eth1. Client C simulates users. Client C with IP 192.168.6.238 would create HTTP connections to Web Server, whose IP is 192.168.16.199, to download the default web page. The packet-sniffing tool, tcpdump, is running on system interfaces of Gateway (both eth0 and eth1) and that of Client C (eth1). The captured packets are carefully studied and compared, mostly on the socket pair information and the sequence/acknowledge numbers in TCP header.

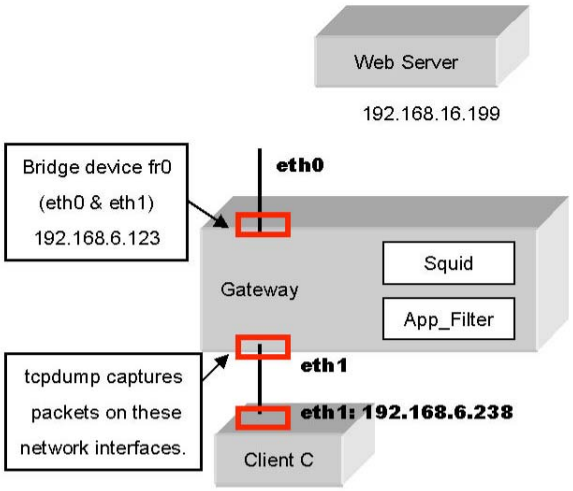

Fig. (5). Lab network diagram.

Three scenarios are considered as follows.

1. Direct Connection: Squid and App_Filter are stopped and Redirection rules are disabled, as shown in Fig. (**6**).

2. Common Transparent Proxy (Squid): Only Squid is running on Gateway and Redirection rules are enabled, as shown in Fig. (**7**).

3. Proposed Transparent Proxy (App_Filter): Only App_Filter is running on Gateway and Redirection rules are enabled, as shown in Fig. (**8**).

In scenario 1, no proxy daemons are running on Gateway and Redirection rules are disabled. Gateway serves as a pure

bridge, simply forwarding packets according to the internal forwarding table. When Client C opens an HTTP connection to Web Server, only one connection is created between Client C and Web Server. The result of tcpdump, running on several interfaces, displays the connection information. For simplicity, only socket pair information (denoted as *tcp@*) and sequence/acknowledge numbers (denoted as *seqno*) of the third packet are shown here. The information gathered at each system interface is shown as follows:

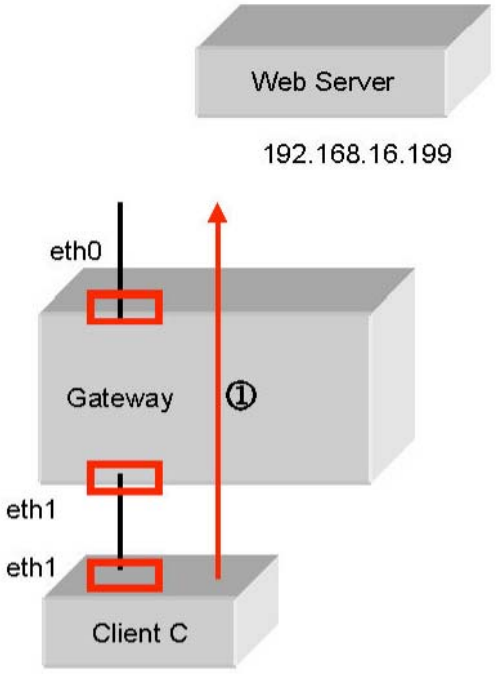

Fig. (6). Direct connection.

In Client C, the summary of the third packet is:
tcp@ [192.168.6.238:39486⇔192.168.16.199:80]
seqno 1520362652⇔2262076546

In Gateway LAN (eth1), the summary of the third packet is:
tcp@[192.168.6.238:39486⇔192.168.16.199:80]
seqno 1520362652⇔2262076546

In Gateway WAN (eth0), the summary of the third packet is:
tcp@[192.168.6.238:39486⇔192.168.16.199:80]
seqno 1520362652⇔2262076546

The three packets gathered from three different network interfaces have the same socket pair and sequence/acknowledge numbers. It is proved that the three packets are in fact the same packet, and are belonging to the same connection (1) as shown in Fig. (**6**).

In scenario 2, Squid is running on Gateway (see Fig. (**7**)) and listens to TCP port 3128. Redirection rules are configured to redirect connections of tcp@80 to a local process running on tcp@3128. Connections from Client C are handled by the local process Squid on Gateway. By parsing the "HOST" field of an HTTP Request Header, Squid is able to retrieve the fully qualified domain name (FQDN) of the real destination web server. Then Squid opens a second connection to the web server, which is 192.168.16.199 in this case. Packet-sniffing tool, tcpdump, is running on each interface to display the connection information. For simplicity, only socket pair information (denoted as tcp@) and sequence/acknowledge numbers (denoted as seqno) of the third packet are shown here. The information gathered at each system interface is shown as follows:

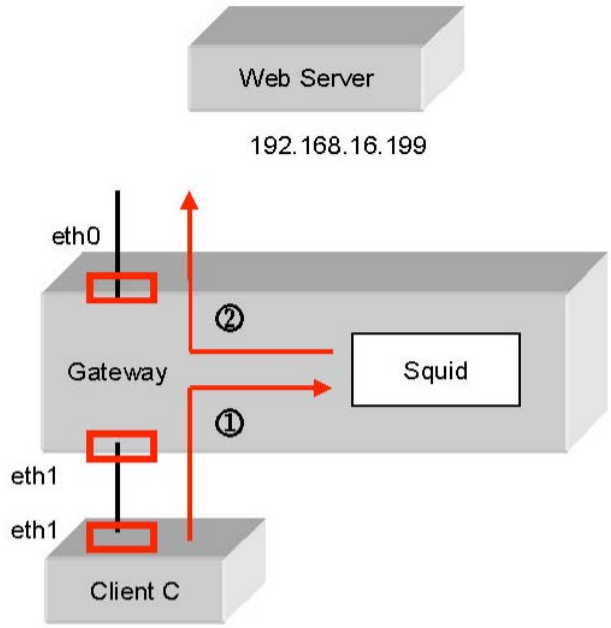

Fig. (7). Common transparent proxy (Squid).

In Client C, the summary of the third packet is:
tcp@ [192.168.6.238:64880 ⇔192.168.16.199:80]
seqno 314078504 ⇔1225491954

In Gateway LAN (eth1), the summary of the third packet is:
tcp@ [192.168.6.238:64880 ⇔192.168.16.199:80]
seqno 314078504 ⇔1225491954

In Gateway WAN (eth0), the summary of the third packet is:
tcp@[192.168.6.123:39643⇔192.168.16.199:80]
seqno 121085213 ⇔ 1053742436

The socket pairs as well as the sequence/acknowledge numbers viewed from Client C and Gateway LAN (eth1) are exactly the same. That is, by the use of Redirection technique, Squid successfully hides itself to Client C. Client C believes it is communicating directly with Web Server. However, in Gateway LAN (eth0), the result of a different socket pair indicates Web Server knows the source address of the connection is Gateway, instead of Client C. Therefore, it is concluded that Squid does not satisfy the requirement of end-to-end transparency.

In scenario 3, App_Filter, is running on Gateway (see Fig. (**8**)) and listens to TCP port 9000. Note that App_Filter is simply a generic proxy developed with the extended API set on the proposed platform. It does not need to parse the HTTP Request Header but simply forwards data received from one connection to the other connection and vice versa. Therefore, App_Filter is not protocol specific but suitable for almost any TCP applications. The program is so simple that the actual code is less than 150 lines. Because of enabled Redirection rules, all HTTP connections (tcp@80) sent from Client C are redirected to tcp@9000 and handled by

App_Filter. This is, the first connection (1) is opened between Client C and App_Filter. App_Filter uses the extended function call getsockopt() to retrieve the original source as well as destination socket information (before redirection), and then opens a second connection (2) to Web Server. Upon the creation of the second connection, App_Filter uses the extended function call setsockopt() to instruct netfilter to perform masquerading so that the source socket pair of the second server connection is the same as that of the first client connection. The result of tcpdump, running on several interfaces, displays the connection information. For simplicity, only socket pair information (denoted as tcp@) and sequence/acknowledge numbers (denoted as seqno) of the third packet are shown here. The information gathered at each system interface is shown as follows:

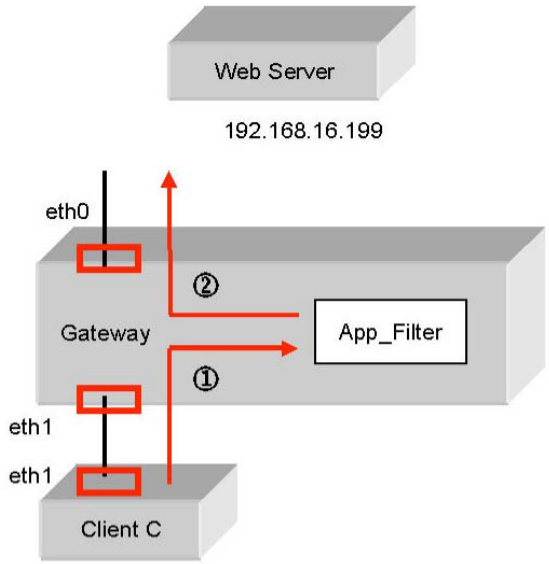

Fig. (8). Proposed transparent proxy (App_Filter).

In Client C, the summary of the third packet is:
tcp@ [192.168.6.238:9186 ⇔192.168.16.199:80]
seqno 778822275 ⇔ 1681053179

In Gateway LAN (eth1), the summary of the third packet is:
tcp@ [192.168.6.238:9186 ⇔192.168.16.199:80]
seqno 778822275 ⇔ 1681053179

In Gateway WAN (eth0), the summary of the third packet is:
tcp@[192.168.6.238:9186 ⇔192.168.16.199:80]
seqno 1688383135 ⇔ 15533100426

The socket pairs as well as the sequence/acknowledge numbers viewed from Client C and Gateway LAN (eth1) are the exactly same. Redirection technique successfully helps App_Filter hides itself to Client C. Client C believes it is communicating directly with Web Server. From the result gathered on Gateway LAN (eth1) and Gateway WAN (eth0), the two packets have the identical socket pair but different sequence numbers. It is proved that the two packets are different packets, and are belonging to two different connections respectively. However because of the identical socket pair, Client C and Web Server believe that they are in direct communication with each other without noticing the existence of Gateway.

Based on the above experiments, the platform has successfully demonstrated itself to be a truly application-layer filtering firewall. Firewall programmers can focus most of their time on the study of protocol specifications, and the development of respective software filters without worrying about any issues in actual transparent deployment.

5. CONCLUSION

Traditional packet-filtering firewalls only examine packet integrity on a per packet basis, while proxy-based firewalls analyze the application-layer data to provide 100% complete security. Besides deployment issues, current transparent application firewalls only support a limited number of protocols and the products all posses the same problem – the source address is always modified. To our knowledge, there is no commercial product to solve the problem. Also, we have not found any research literature discussing on possible solutions. Therefore, the paper studies the concept of transparency, presents the drawbacks of current transparent HTTP and SMTP deployment, and proposes a feasible solution. The realization of the idea is based on Linux OS. The bridge mode is suitable for plug-and-play installation and an extended API set is provided for firewall developers. The extended API is compatible with standard BSD Socket API so that the migration of existing proxy-based filters is extremely easy. The operation of proxy-based filters is to intercept incoming/outgoing connections, acquire the original connection information, and then analyze the data content to achieve application-layer protection. There is no need to adjust any proxy/SOCKS settings of client software. The proposed platform supports almost every kind of TCP application protocols, from the early Internet protocols HTTP, SMTP, POP3, NNTP or FTP, to the latest IM, P2P and many online gaming protocols.

REFERENCES

[1] A. Habtamu. "An Overview of Firewall Technologies." Internet: heim.ifi.uio.no/~abie/FirewallTechnologies.pdf, January, 2000 [May 9, 2007].

[2] G. Mark. "Firewall and Proxy Server HOWTO." Internet: tldp.org/HOWTO/Firewall-HOWTO.html, February 26, 2000 [May 9, 2007].

[3] Check Point. "Securing Networks With Next-Generation Unified Threat Management." Internet: www.mark-ent.com/documents/WhitePaper.pdf, June, 2006 [May 9, 2007].

[4] D.G. Nelly. "Patterns for Application Firewalls." Internet: hillside.net/plop/2004/papers/ndelessygassant0/PLoP2004_ndelessygassant0_0.doc, June, 2004 [May 10, 2007].

[5] Cedric. "Layer 2 filtering and transparent firewalling." Internet: sid.rstack.org/pres/0307_LSM03_L2_Filter.pdf, July 11, 2003 [May 10, 2007].

[6] Constantinos. "BSD Sockets API." Internet: pages.cs.wisc.edu/~lhl/cs740/assignments/references/sockets_const.ps, April, 1999 [May 3, 2007].

[7] J. Moscola, J. Lockwood, R.P. Loui etc. "Implementation of a Content-Scanning Module for an Internet Firewall," presented at the 11th FCCM, Napa, C.A., 2003.

[8] W. Sebastian. "Investigating large-scale Internet content filtering." M. Sc. thesis, University of Dublin City, Ireland, 2006.

[9] P. Akritidis, K. Anagnostakis and E.P. Markatos. "Efficient Content-Based Detection of Zero Day Worms." in *IEEE International Conference*, 2005, pp. 837-843.

[10] Shannon and D. Moore. "The spread of the Witty worm." *Security & Privacy Magazine, IEEE*, vol. 2, pp. 46-50, July-August 2004.

[11] Z. Chen and C. Lin. "AntiWorm NPU-based Parallel Bloom Filters for TCP/IP Content Processing in Giga-Ethernet LAN." in *Local Computer Networks IEEE Conference*, 2005, pp. 15-17.

[12] K. Wang and J.S. Salvatore. "Anomalous Payload-based Network Intrusion Detection." presented at the 7th RAID, France, 2004.

[13] Mike and V. George. "Fast Content-Based Packet Handling for Intrusion Detection." Internet: cse.ucsd.edu/Dienst/UI/2.0/ Describe/ncstrl.ucsd_cse/CS2001-0670, May 7, 2001 [May 14, 2007].

[14] LXR community. "Linux Cross Reference." Internet: lxr.linux.no/, 2007 [May 15, 2007].

[15] R. Rusty and W. Harald. " Linux netfilter Hacking HOWTO." Internet: www.netfilter.org/documentation/HOWTO/netfilter-hacking-HOWTO.html, July 2, 2002 [May 17, 2007].

[16] R. Rusty. "Packet Filtering HOWTO." Internet: www.netfilter.org/documentation/HOWTO/packet-filtering-HOWTO.html, January 24, 2002 [May 17, 2007].

[17] R. Rusty. "Networking Concepts HOWTO." Internet: www.netfilter.org/documentation/HOWTO/networking-concepts-HOWTO.html, The netfilter webmaster, July 29, 2001 [May 17, 2007].

[18] R. Rusty. "NAT HOWTO." Internet: www.netfilter.org/ documentation/HOWTO/NAT-HOWTO.html, January 14, 2002 [May 22, 2007].

[19] M. Fabrice. "Netfilter Extensions HOWTO." Internet: www.netfilter.org/documentation/HOWTO/netfilter-extensions-HOWTO.html, October 22, 2006 [May 22, 2007].

Received: January 01, 2008 Revised: February 06, 2008 Accepted: February 07, 2008