Asian Journal of Information Technology 6 (11): 1105-1109, 2007
ISSN: 1682-3915
© Medwell Journals, 2007

Implementing an Application Gateway for Client/Server Systems

A.J. Jegede and G.I.O. Aimufua
Department of Mathematical Sciences, Nasarawa State University, Keffi, Nigeria

Abstract: This study was carried in the computer science laboratory of the Department of Mathematical Sciences, Nasarawa state University, Keffi, Nigeria. The research is a continuation an earlier work titled 'A specification of an application gateway for client/server systems' published in Asian Journal of Information Technology, vol. 6 No. 8. The model of an application gateway proposed in the earlier work is implemented here as a totally decentralized distributed application consisting of three subsystems: Generate packet system, AppGateway system and Learning system. The system flowchart is used to illustrate how each of the subsystems works as well as their interactions in the overall system. Java is used as the language of implementation because it is very suitable for network communication and it has extensive routines for dealing with communication protocols. The existence of these routines provides java with the capability and flexibility to communicate with TCP/IP protocols such as HTTP, FTP, etc.

Key words: Application gateway, network, packet, packet generation protocols, system flow diagram

INTRODCUCTION

In a study titled 'X Through the Firewall and other Application Relays', Wolman (1998) referred to application gateway as an application proxy or application layer firewall. Unlike packet- filtering-based and stateful-packet-filtering-based firewalls which examine incoming and outgoing packets at only the network and session levels, application layer firewalls inspect traffic at the application level in addition to lower levels. Ranum (2001) posited that application layer firewalls can be used as network address translators since traffic goes in one side and comes out the other, after having passed through an application that effectively masks the origin of the initiating connection.

An application gateway works as follows. When a packet comes into an application layer firewall, it is handed off to an application specific proxy, which inspects the validity of the packet and the application level request. For example, if a web request (HTTP) comes into an application gateway, the data payload containing the HTTP request will be handed off to an HTTP proxy process. In the same vein, an FTP request would be handed to an FTP-proxy process, Telnet to a Telnet-proxy process and so on Anonymous (2001). This concept of a protocol-by-protocol approach is more secure than stateful and generic packet filtering because the firewall understands the application protocols (that is, HTTP, FTP, etc) themselves. This application gateway developed in here is meant to provide security for local area network connected in a client/server mode. The gateway stands between a local area network (connected in a client/server mode) and external networks and systems. It ensures that only authorized network packets are allowed to enter or leave the internal local area network.

MATERIALS AND METHODS

We used the system flow chart to illustrate how each of the program modules functions as well their interaction in the overall system. The system flow chart uses 5 basic symbols as shown in Fig. 1.

The local area network and the external network are designated as entities and are represented using the symbol shown in Fig. 2.

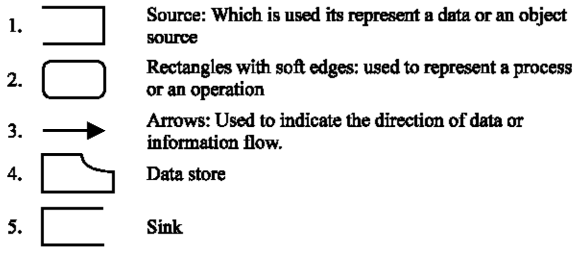

1.
Source: Which is used its represent a data or an object source
2.
Rectangles with soft edges: used to represent a process or an operation
3.
Arrows: Used to indicate the direction of data or information flow.
4.
Data store
5.
Sink

Fig. 1: System flow diagram symbols

Fig. 2: The entity symbol

Corresponding Author: A.J. Jegede, Department of Mathematical Sciences, Nasarawa State University, Keffi, Nigeria

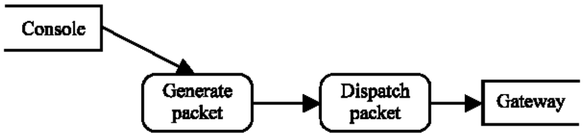


Fig.3: Flow diagram illustrating the operation of the packet generation module

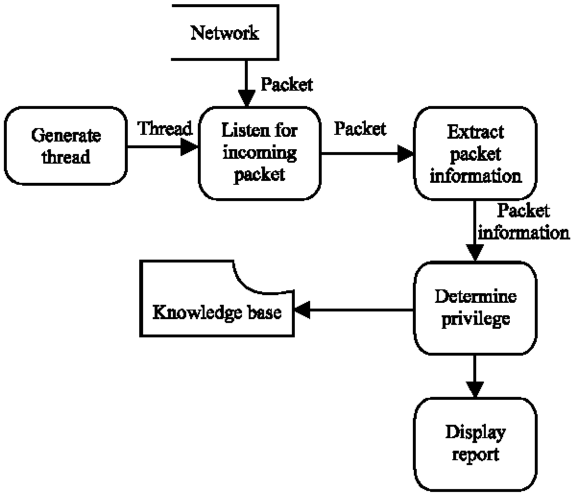


Fig. 4: System flow diagram illustrating the operation of the application module

The program modules: The program comprises of four modules. Each of these modules is presented as a class with states and methods that perform a particular function. These modules are:

Packet generation module (implemented as general packet class): This program module was necessitated by the need to overcome the constraint of generating the packets to the parsed (or filtered) by the application gateway. This class extends with the JFrame class of the javax.swing packet. It uses JTextfields and JComboboxes to capture the parameters of the packet to be generated. The packet (together with its parameters) is attached to the check event of the JButton. This check event is captioned OK. This module (or class) generates a packet object and serializes it on the network through the application gateway module (that is, the AppGateway Class). Its operation is illustrated by the flow diagram shown in Fig. 3.

Application gateway module (implemented as the appgateway class): This module accepts the packets generated by the packet generation module and parses (or filters) them based on the information obtained from the knowledge base.

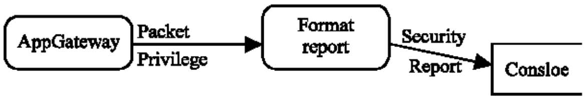
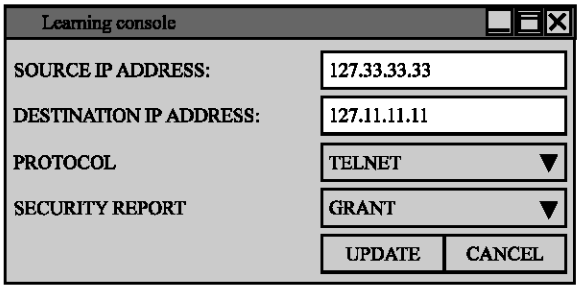


Fig. 5: System flow diagram illustrating the operation of the display report module



SOURCE IP ADDRESS: 127.33.33.33

DESTINATION IP ADDRESS: 127.11.11.11

PROTOCOL: TELNET ▼

SECURITY REPORT: GRANT ▼

Fig. 6: Graphical user interface depicting the (grant) operation of the learning system module

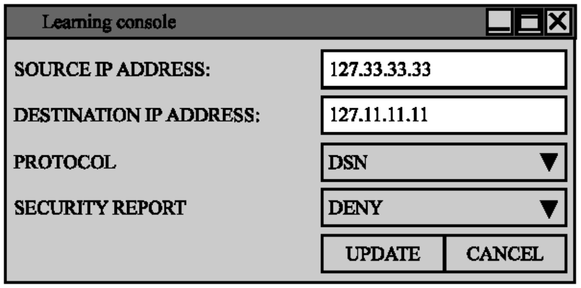
The knowledge base holds the following information about each packet:

- Source address
- Destination address
- Protocol
- Access rights

The AppGateway class (or module) performs this function by using the methods of the Java.net, java.sql, java.io and java.odbc packages. The class uses threads to manage the packet filtering operation. This enables the class to perform uninterrupted filtering function. The system flow diagram shown in Fig. 4 illustrates how the application gateway module works.

Display report module (implemented as display report class): The purpose of this module is to display the report generated by the application gateway module (that is, the AppGateway class). This class is designed as a modal Jdialog class. It is invoked by the AppGateway class and its operation is depicted in Fig. 5.

Learning system module (implemented as the learning system class): This module is purposely for creating and updating the knowledge base. The knowledge base is used to provide preset information on which the application gateway module operates. The learning system module provides an interface through which rules are added to or removed from the knowledge base. That is, it serves as a means of specifying ahead of time which source is allowed to reach a particular destination based on which protocol. For example, the learning console graphical user interface in Fig. 6 provides a means


Learning console

SOURCE IP ADDRESS:	127.33.33.33
DESTINATION IP ADDRESS:	127.11.11.11
PROTOCOL	DSN ▼
SECURITY REPORT	DENY ▼

Fig. 7: Graphical user interface depicting the (deny) operation of the learning system module

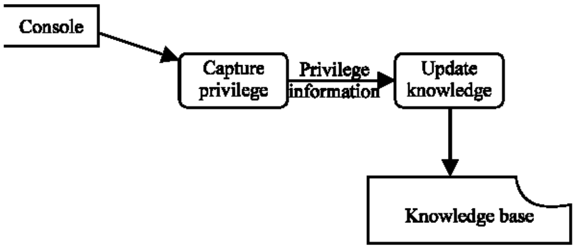


Fig. 8: System flow diagram illustrating the operation of the learning system module

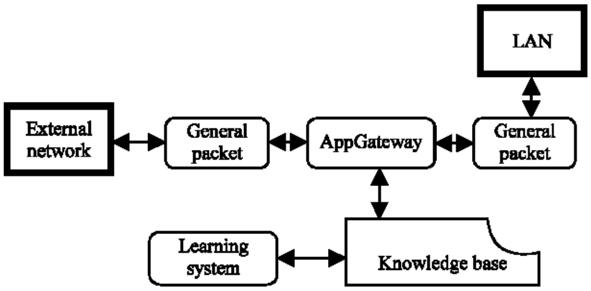


Fig. 9: System flow diagram depicting the overall operation of the gateway system

of specifying a rule (in the knowledge base) which allows packets with source address 127.33.33.33 to reach destination whose address is 127.11.11.11 using the TELNET protocol.

Conversely, we can specify another rule which does not authorize packets originating from a source whose address is 127.33.33.33 to reach a destination whose address is 127.11.11.11 using the DNS protocol. The GUI shown in Fig. 7 illustrates how this can be done.

The learning system class is designed as a Jframe class. It uses the java.odbc and java.sql packages to update the knowledge base. The learning system captures its input via a Graphical User Interface (GUI) powered by the javax.swing package and updates the knowledge base via the Jbutton event handler. The flow diagram in Fig. 8 depicts the operation of the learning system module.

All these modules interact to achieve the overall objectives of the application gateway system. This interaction is illustrated by the system flow diagram in Fig. 9.

DEVELOPMENT ENVIRONMENT

The development environment for the system consist of the hardware and the software platforms on which the system was implemented as well as the programming language and other tools (utilities) used for implementing the system.

Hardware requirements: These comprise of the following:

- A VGA/SVGA monitor
- Pentium II MMX processor and above
- A32 MB memory (RAM)
- A 101 enhanced keyboard or a compatible
- A hard disk of about 500 MB and above
- A mouse
- Network Card (for handling network connection)

Software requirements: The software requirements for the system includes

Java Virtual Machine (JVM): The JVM provides a further interpretation and execution of the byte code produced by the java complier. The java virtual machine adds operating system specific information to the java byte code so that the program can run on a specific operating system. When the byte code is run, the JVM interprets and checks the integrity and security of the byte code. Then it dynamically applies specifics based on the parameters found in systems configuration and environment variables.

Java Development Kit (JDK) version 1.3.1.2: The java Development Kit (JDK) consists of a number of executables which are command line driven. The tools available in the Java Development Kit are as follows:

- Javac i.e. the java complier
- Appletviewer i.e. java's applet viewer
- Java i.e. java interpreter
- Javap i.e. java disassembler
- Javah i.e. C header and stub file creator
- Javadoc i.e. java document generator
- Jbd i.e. java debugger

Microsoft access 2000: This is used for creating and managing the knowledge base.
A Microsoft Windows operating system

IMPLEMENTATION LANGUAGE

Java is chosen as the language of in domination because it has broader capacity and more power (Harold, 1997) Moreover, java is robust, secure architecture neutral and portable (Walsh, 1996).

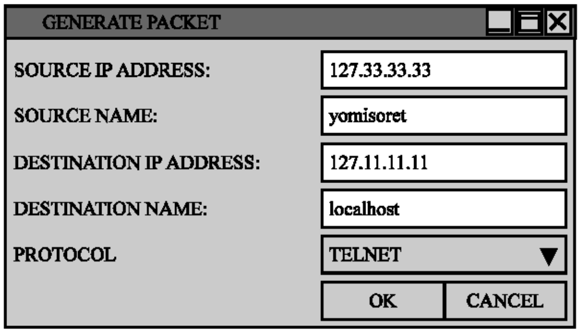
RESULTS AND DISCUSSION

The byte code resulting from the compilation of the Java code for the system is packaged into a self-extracting executable that places the entire system in the folder C:\Gateway. To start the packet generation application, it is ensured that the system is switched to the C:\Gateway directory. This is followed by typing "Java Generate Packet" or by double-clicking the Generate Packet system bat file. This leads to the coming up of the window based GUI shown in Fig. 10. As contained in the figure, packets are generated based on the input supplied by the user. Here we intended to generate a packet with an IP source address 127.33.33.33 which originates from a terminal named yomisore1. The packet which uses the telnet protocol is to be dispatched to another machine named localhost with an IP address 127.11.11.11.

On clicking the OK button, this subsystem creates an object of the serializable packet class which is passed to the application gateway for the filtering (or parsing) operation. Starting of the application gateway requires the typing of the "Java AppGateway" at the command prompt in the C:\Gateway directory or the double clicking of the AppGateway system bat file. At startup, this subsystem listens for incoming packets while displaying the GUI shown in Fig. 11.

On detecting an in-coming packet, the application gateway automatically extract the parameter contained in the packet and compares them with the information available in the knowledge base. A dialog box is displayed to report the access right allowed between the source machine and the destination machine based on the protocol used. The dialog box is as shown in Fig. 12.

The learning system is used to create and update the knowledge base of the application gateway. To update the knowledge base, the learning system is started by typing Learning system at the command prompt in the C:\Gateway directory or by double clicking the LearningSystem bat file. An ODBC driver is created for the database with the name Gateway via the administration tools of the operating system. On lunching this subsystem, the GUI shown in Fig. 13 comes up. This interface allows the user to grant or revoke privileges between the machine on the client/server LAN and those on the external network.



SOURCE IP ADDRESS:	127.33.33.33
SOURCE NAME:	yomisoret
DESTINATION IP ADDRESS:	127.11.11.11
DESTINATION NAME:	localhost
PROTOCOL	TELNET ▼

Fig. 10: Graphical user interface illustrating the packet generation process

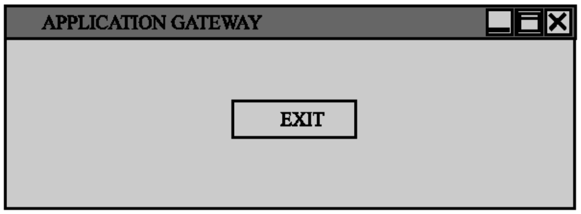
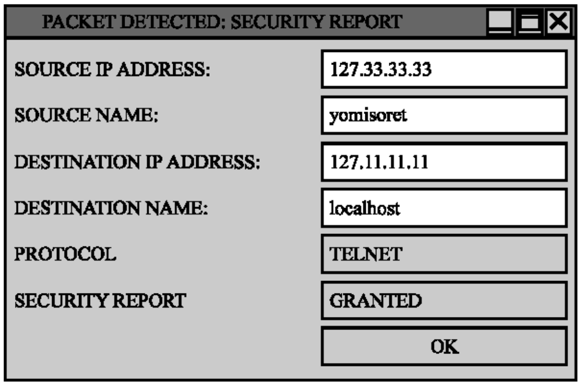
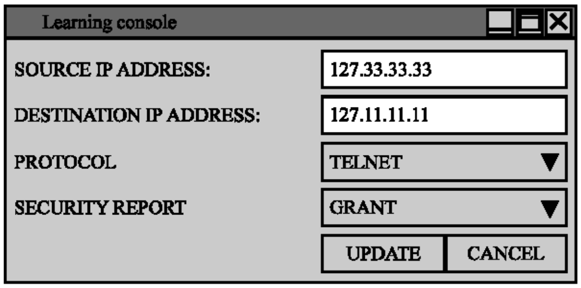


Fig. 11: Graphical user interface depicting the listening process of the AppGateway system



SOURCE IP ADDRESS:	127.33.33.33
SOURCE NAME:	yomisoret
DESTINATION IP ADDRESS:	127.11.11.11
DESTINATION NAME:	localhost
PROTOCOL	TELNET
SECURITY REPORT	GRANTED

Fig. 12: Graphical user interface depicting the packet detection and security reporting processes



SOURCE IP ADDRESS:	127.33.33.33
DESTINATION IP ADDRESS:	127.11.11.11
PROTOCOL	TELNET ▼
SECURITY REPORT	GRANT ▼

Fig. 13: Graphical user interface depicting the use of the learning system to update the knowledge base

CONCLUSION

This study provides a good background to other interesting and challenging areas of studies in network security. Although, we attempted to implement only a model of an application gateway to take care of perimeter (external) security in client/server environments, the work can be extended to accommodate further study/research in the following areas:

- Extension of the tools to support more protocols. This made possible by streamlining the implementation of the system with the TCP/IP protocol suite, which in the standard protocol for network communication. Although, the system currently supports only four members of the TCP/IP suite; that is, DNS, FTP, HTTP and Telnet, there exists the possibility of upgrading the tool to support more members of the TCP/IP suite such as SNMP-Simple Name Management Protocol and SMTP-Simple Mail Transfer Protocol. All that is required is to develop modules to process the packets using the new protocols as well as the incorporation of additional rules in the knowledge base. The allowance for extension is necessary because network environments are not static in real life. There is always the need to support a new service or to deactivate an existing one based on the objectives of the environment and/or security requirements or considerations.
- The need to continue the benefits of traditional packet filtering with these of an application gateway. This requires the development and verification of a system to implement the combined approach. The question of combining the strengths of the traditional packet filtering approach with those of the application gateway is currently at the forefront of research in firewall security.
- Exploration of the possibility of merging the tool with other network security strategy(ies) to obtain an enhanced security suite. This arises from the fact that application gateway is a reactive model (that is, it waits for a security breach before it responds) while other approaches such as intrusion detection system and scanners are reactive models. That is, they do not wait for a security breach before responding. Instead, they periodically examine the network for security holes and then react accordingly. Merging the two approaches (reactive and proactive) definitely provides a better means of managing security. However, there exists the need to define, design, develop and verify a system to implement the combined approach to see if and how it will work in a practical situation.

REFERENCES

Anonymous, 2001. Maximum Security. 3rd Edn. Sams, Indianapolis. United States, pp: 194-195.

Harold, E.R., 1997. Java Secrets, IDG Books Worldwide Inc., United States, pp: 196.

Jegede, A.J., G.I.O. Aimufua and H.O. Salami, 2007. A Specification of an Application Gateway for Client/Server Systems. Asian J. Inform. Technol., 6: 847-853.

Ranum, M.J., 2001. Internet Firewalls: Frequently Asked Question, http://www.intehack.net/pubs/fwfaq.

Walsh, A.E., 1996. Fundamentals of Java Programming for the World Wide Web, IDG Books Worldwide Inc., United States, pp: 170-191.

Wolman, T., 1998. X Through the Firewall the other Application Relays, Digital Equipment Corp., Cambridge Research Lab., ftp://crlssdoc.com./pub/DEC/CRL/tech.