A DRIVER-BASED RELIABLE DISTRIBUTED FIREWALL SYSTEM

MUMTAZ MOHAMMED ALI AL-MUKHTAR
College of Information Engineering, AL-Nahrain University
IRAQ

ABSTRACT

This paper proposes a Reliable Distributed Firewall System (RDFS), which is a client-server network paradigm. The system consists of two elements: a Distributed Firewall-Client, which captures every transmitted or received packet, originated from or received by the client machine, and then applies the filtering rules on these packets. The second element is the Firewall-Controller designed as a user friendly GUI, which manages the Firewall-Clients on all the machines. It can read, write or modify the rules for each client individually through authenticated and secured communication channels. Each Firewall-Client uses the Firewall-Hook Driver on Windows platform as the firewall application.
The proposed distributed system addresses the shortcomings of the conventional firewall as being the networks' bottleneck. This is done by decentralizing the packets filtering processes and making them work independently. This would increase the system availability and at the same time protect against internal attacks, which is unfeasible using the conventional firewall setup. In addition, the system can be configured for fail-over mode, by imposing a dual controller. This would enhance the overall system availability remarkably. Policy optimization techniques for rules and security policies are developed to reduce the processing requirement per packet, thus faster filtering speed can be achieved.

Keywords: *Firewall, Network Security, Security Management, Firewall-Hook Driver, Dual-Redundancy*

1. INTRODUCTION

Firewalls have become a very common technology to use for securing computers and networks. A firewall can be classified as any device that limits network access. It is a collection of components inserted between two networks that filter traffic between the networks according to a local security policy [1].

Conventional firewalls rely on notions of restricted topology and control entry points to function. More precisely, they rely on the assumption that everyone on one side of the entry point-the firewall- is to be trusted, and that anyone on the other side is, at least potentially-an enemy [2].

The assumption that all insiders are trusted no longer holds true as organizations try to safeguard themselves against other types of threats [3]. Conventional firewalls were never designed to solve the insider problem, and intrawalls, which move security enforcement closer to the user, ease the problem only slightly at the cost of significantly more complex management [4]. Due to the increasing line speeds and the more-computation-intensive protocols that a firewall must support, firewalls tend to become congestion points.

The distributed firewall concept, first introduced by Bellovin in 1999 [5], provides firewall protection at the network end-points via a centrally defined policy. Unlike conventional firewalls, which only provide protection at the network perimeter, distributed firewalls provide host protection for internal threats. Distributed firewalls are topology-independent, provide fine-grained access control, and reduce global performance bottleneck.

Some works [6,7] attempt to solve the problems using multiple firewalls. However, what comes into question is whether the added cost of hardware and delay is worth the added security. Ioannidis, Keromytis, Bellovin and Smith [8] described a distributed firewall for Open BSD hosts. In this scheme, security policy is still centrally defined using the Kenote trust management system [9] to specify, distribute, and resolve policy. Concepts of an embedded distributed firewall architecture that is implemented on the host's network interface card (NIC) are described in [10,11,12]. A possible way to implement a distributed firewall by the use of the agent technology is proposed in [13] where firewall rules sets are to be distributed to a set of controller agents scattered on some network nodes. In [14], the authors propose a distributed Internet security system called General Network Security Collaboration Framework (GNSCF) where firewalls are used as the basic network element of the network.

This paper introduces a distributed firewall architecture called Reliable Distributed Firewall (RDF). A firewall is placed at each host in the network to address the insider problem. The kernel-mode driver "Firewall Hook Driver" supported by windows 2000/XP platforms and later version is developed to manage a host's packet filter firewall. Policy management remains centralized. A dual-redundant controller manages all the hosts. However, each host can be configured in a way that does not affect the others, i.e., managed independently.

In conclusion, three main contributions are made in this paper: (1) the controller (policy manager) can support redundancy that enhances the overall system reliability; (2) the communication between the controller and clients is secure and authenticated; (3) the ruleset is optimized by discarding the overlapped or conflict rules.

2. FIREWALL-HOOK DRIVER

The basic structure of a Microsoft® Windows® 2000/XP related to Window Driver Model

(WDM) consists of a required set of system-defined standard driver routines, plus some number of optional standard routines and internal routines, depending on the type of a driver and the underlying device. The common set of standard routines allows all kernel-mode drivers to process I/O Request Packet (IRP) by calling system-supplied support routines [15].

The Microsoft Windows 2000 Driver Development Kit (DDK) introduced the concept of a firewall-hook driver. The intent of a firewall-hook driver is to manage network packets that are sent and received across a firewall in the context of the TCP/IP protocol [16].

Implementing the firewall-hook driver to manage network packets constitutes a low down solution in the network stack that enhances the overall firewall performance significantly. The basic idea is simple; a DDK translates the policy language into driver internal format. The system controller distributes this policy file to all hosts that are protected by the firewall. All incoming packets are accepted or rejected by each host according to the policy without affecting the machine's performance or throughput.

3. SYSTEM FILTERING POLICY

The filtering function would start at the top of the ruleset's link list and work down through the rules. Whenever a rule that permits or denies the packet is found, one of the following actions is taken:

- Allow: the function will forward the packet to its destination as requested.
- Deny: the function will discard the packet without returning an error message to the source; this will hide the firewall presence for the outsiders. Also an entry will be generated in the log file; the entry consists of the dropped packet's information and the time of incidence.

The default rulesets for each firewall filter are predefined depending on the controller's IP address, the client's IP address and the communication port used. The system uses "the deny all" as access denial method that is all the inbound and outbound traffic is denied unless it is explicitly allowed by the ruleset. Therefore the communication ports are explicitly allowed at the setup time.

4. RELIABLE DISTRIBUTED FIREWALL ARCHITECTURE

The Reliable Distributed Firewall System (RDFS) consists mainly of the following modules:

- Controller Module: it manages (add, modify, delete, and optimize) the ruleset for the client modules. This module is also responsible for keeping track of each client module update time and history. The controller supports redundancy by implementing a fail-over mode. This mode requires the availability of a second controller in the network. So it is possible to configure a redundant controller to enhance system availability.
- Client Module: it implements the ruleset that the controller module sends for execution by the driver module. This module acts as the interface between the controller and the driver modules.
- Communication Module (Comm.): this module is responsible for securing the flow of filtering rules and control signals between the controller and client modules by adopting the data authentication and encryption techniques.
- Optimization Module: this module is responsible for optimization the ruleset before sending it to the client module for implementation. This module increases the firewall performance.
- The Diver Module: it manages the driver installation and uninstallation. In addition it manages the execution of the ruleset sent by the client module.

Figure 1 illustrates the RDFS architecture and the interaction among its modules that are shown in a shaded color. The upper part illustrates the controller application that based on an optional dual-redundancy. The system can run in a single mode where only one controller exists, or in a fail-over that requires the presence of two controllers.

The fail-over mode is the important insurance of controller application to operate continuously. The dual controller system adopts "heartbeats" method to keep connection between the primary and secondary controllers, and to show the present operation state of the system. The heartbeat is represented in figure 1 by a dotted line. The lower part of the figure represents the firewall-client application, which constitutes the packet filter.

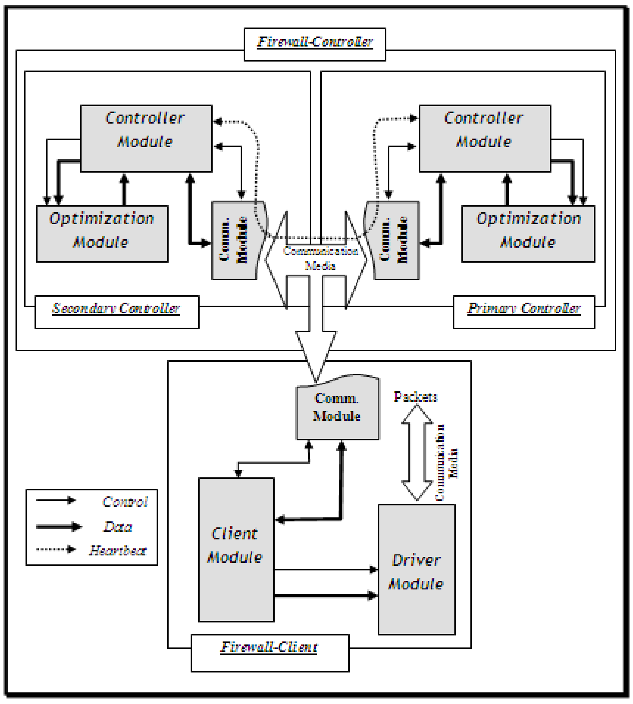

Figure -1- The RDFS Architecture

THE CONTROLLER MODULE

The controller module is the main part of the proposed system. Its main objective is the management of the distributed firewall-clients in terms of behavior and filtering rules. This would be done for each client independently of the others.

The multiple clients' management is done through an easy to use GUI where all the functionality of the controller can be carried out. All the administration is done using this module only; hence this module represents the only interface the system administrator will use to control the system and the clients. This module interacts directly with two other modules: the optimization module and the communication module.

4.1.1 MODULE OVERVIEW

This module consists of two parts; client part which controls the client behavior (add, delete, and start/stop service on the designated client). The second part is the ruleset or policy part, which controls the ruleset of the designated client. This part manages the addition, deletion, modification and rearranging of rules. Also it is responsible for interacting with communication module for fetching and sending the rules from and into the clients. This is illustrated in figure 2, which shows the controller module's components and its interaction with the other two modules.

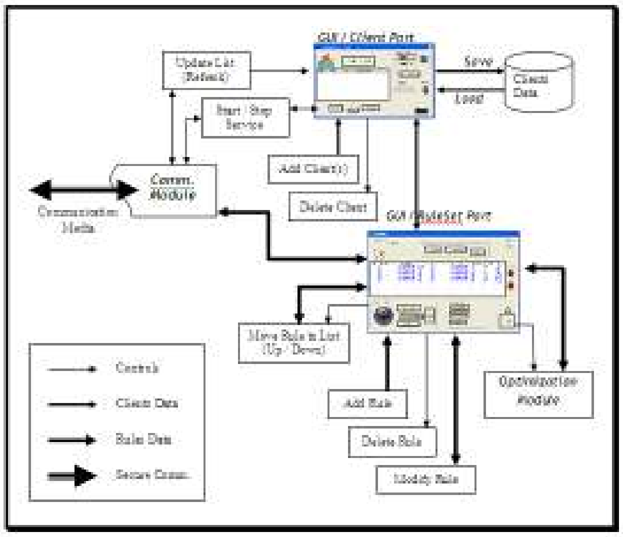

Figure -2- Controller Module

4.1.2 FAIL-OVER MODE

By installing a second controller into the network, a fail-over mode can be accomplished. It is possible to configure a redundant controller to provide a continued operation.

The following notes will summarize the behavior of this mode:

- This mode works only with two controllers located in the same subnet.
- The primary and the secondary controllers are determined during setup time.
- An indication is shown in the controller's GUI window to differentiate between the primary and the secondary controllers.
- All GUI functions of the secondary controller will be disabled except for the view functions, which enable only one active controller in the network.
- Each controller listens to other controller's heartbeat. Once after certain measure cycles, the secondary controller does not receive the heartbeat from the primary, it can mark the primary controller as faulty. Hereafter, the secondary controller takes over the control in place of the primary controller.
- When the faulty primary controller is put again into operation, it will still be considered as the secondary controller, and will be authorized only to update its client list from the active controller's list periodically.
- If the secondary is detected as faulty, the primary indicates the failure of the secondary in the controller's GUI window.

4.1.3 FIREWALL CONTROLLER'S FUNCTIONS

The controller module comprises the following functions:

- Functions of the Client Part: Add Client, Delete Client, Update Client List, Start/Stop Service, Save Client List, Load Client List, and Configure Client Firewall.
- Functions of the RuleSet Part: Add Rule, Delete Rule, Modify Rule, Clear List, and Arrange Rule.

These functions are summarized on table 1.

Table -1- Controller Functions Summary

Function Name	Function Summery
Add Client	Adds a new client or multiple clients into the client List.
Delete Client	Deletes a selected client from the client list.
Update Client List	Refreshes the client List and tries to find the status of the clients in the list (on-Line or off-Line).
Start/Stop Service	This toggles the filtering function ON or OFF, but keeps the rules installed.
Save Client List	Writes the client list into a file.
Load Client List	reads the client List from a file.
Configure Client Firewall	This functions loads up the ruleset window (if the Client is On-Line) and starts the ruleset management.
Add Rule	This pops-up the add rule window; it adds a new entry row (rule) in the rules list after entering the required data.
Delete Rule	Deletes a selected rule and updates the list
Clear List	Deletes all the rules and loads the default ruleset instead.
Modify Rule	Modifies a selected rule by enabling the change of any data in the add new rule window.
Arrange (Up/Down)	These two functions can move a rule up or down respectively one row each time activated.

THE CLIENT MODULE

This module acts as the main application in the client side. It does not have a GUI to be managed from, but its main function is to maintain the filter operations and keep the communication channels open with the controller for obtaining new set of rules.

This module works as an interface between the controller and the driver modules, because the controller has no ability to communicate directly with the driver module. The module relays the rules and control signals from the controller module into the driver module. Figure 3 illustrates the client module, its components and its interaction with other modules.

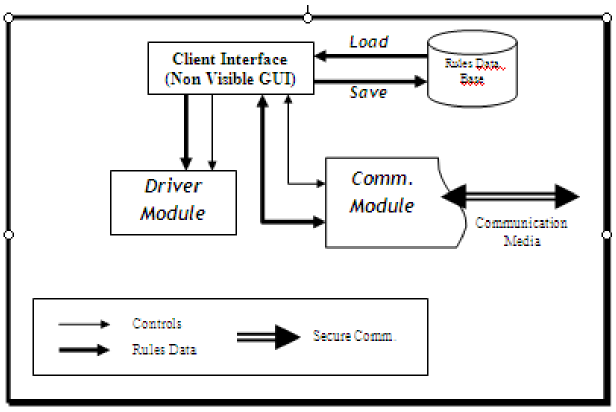

Figure -3-Client Module

THE COMMUNICATION MODULE

This module is responsible for sending and receiving data and controls between the controller and clients modules in secure channels. The security is accomplished by data authentication and encryption. Figure 4 illustrates the communication module and its components.

The communication module function breaks down to seven simple steps:

- Prepare the data for sending using authentication and encryption techniques.
- Open (create) a socket.
- Name the socket.
- Associate with another socket.
- Send and receive data between sockets.
- Authenticate then decrypt the received data.
- Close the socket.

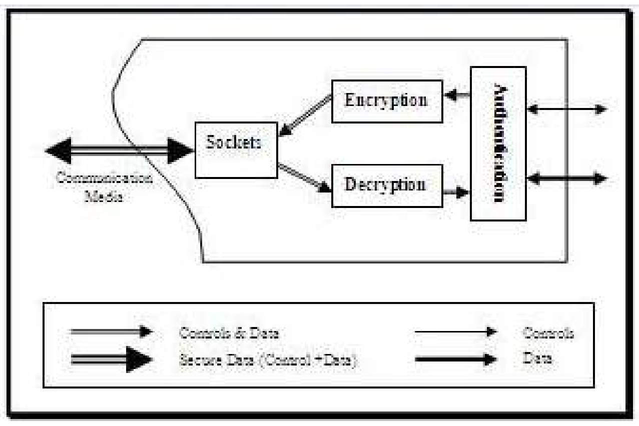

Figure - 4 - Communication Module

A network client-server model is implemented in this system. However, concerning the implementation of this module, the controller represents the client and the firewall-client represents the server. The controller would request a service from the firewall-client (pulling the rules or status and sending them back). Whereas The firewall-client manages the requested service.

The firewall-client application normally listens at a certain address (controller's address) and port address (port# 8148) for new control signals or data generated from the controller. When the controller requests a service, the firewall-client's server processes "wakes up" and services the controller, performing whatever appropriate actions the controller requested. Table 2 shows the controller's requests and their corresponding services obtained from the client-firewall.

Table -2- Requests and Services

Controller Requests	Firewall-Client Services
Is_Client_On-Line	Sends a status signal if on-line.
Retrieve_Rules	Send all the current ruleset to the controller.
Send_Rules	Replace the current rules in the filter with new rulesets.
Start_Filtering	Starts applying the rules on the incoming and outgoing packets.
Stop_Filtering	Stops the filtering actions and accepts all packets.

Before sending any data or control messages to the other communication module, this module runs the secure hash authentication algorithm (SHA), which will result in additional 4 bytes hash code. Thereafter it encrypts the data as well as the resultant code using Rijndeal Algorithm and a predefined key (agreed to by both parties). Now the data is ready to be sent via the communication media.

At the other party (the recipient), the process is reversed; that is, first the module will retrieve the encrypted data using the sockets, and then decrypts it using the predefined key. Thus the data now goes to the authentication process and where the hash code is extracted. Thereafter it runs the SHA hash authentication algorithm on the data and compares the computed code with the extracted code. If there were a match then this data is authenticated and should be passed into the upper modules, else the data is not authenticated and should be discarded.

THE OPTIMIZATION MODULE

The main objective of the optimization module is to assist in firewall policy editing that will lead to speeding up the firewall operation and this is done by:

- Detecting and removing all the redundant rules in the ruleset.

A redundant rule performs the same action on the same packets as another rule. When a large number of filtering rules exist in a policy, the possibility of writing conflicting redundant rules is relatively high. A redundant rule may not contribute in making the filtering decision, however, it adds to the size of the filtering rule table, and might increase the search time and space requirements.

- Arranging the rules according to their use i.e., the common functions first.

The ordering of filtering rules in a security policy is very important in determining the firewall policy because the firewall-hook packet filtering process is performed by sequentially matching the packet against filtering rules until a match is found (linked list is adopted). When a large number of filtering rules exist in a policy, the possibility of writing conflicting or redundant rules is relatively high.

- Detecting the shadow rules.

A rule is shadowed when a previous rule matches all the packets that match this rule, such that the shadowed rule will never be evaluated. Shadowing is a critical error in the policy, as the filtering rule never takes effect. This might cause a permitted traffic to be blocked and vice versa.

- Detecting the general rules.

A rule is a generalization of another rule if the first rule matches all the packets that the second one could match but not the opposite.

- Maintaining the adopted access denial method at all times.

However, optimization the search time complexity is not a concern. Whereas the clarity and simplicity of the resultant ruleset are main issues because the optimization is done at the controller's ruleset window which will not affect the client operations. Initially no relation is assumed. Each field in Rule 1 is compared to the corresponding field in Rule 2 starting with the protocol then source address and port number, and finally destination address and port number. The relationship between the two rules is determined based on the result of subsequent comparisons. If every field of rule 1 is a subset or equal to the corresponding field in rule 2 and both rules have the same action, rule1 will be redundant to rule 2, while if the actions are different, rule 1 will be shadowed by rule 2.

THE DRIVER MODULE

Firewall-Hook Driver is not a network driver; it is a kernel mode driver. Basically, in this Firewall-Hook driver a callback function should be implemented, and then, the function should be registered with the Firewall-Hook driver. Now whenever a packet is being sent and received, the firewall driver will call the callback function each time. Figure 5 shows the driver module and its functions that are summarized on table 3.

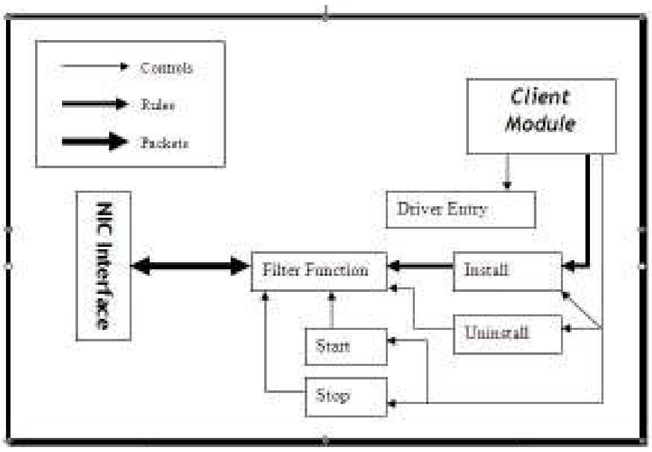

Fig -5- Driver Module

Table -3- Driver Routine Summary

Routine Name	Function Summary
DriverEntry	Initializes driver and sets entry points for other standard routines.
Dispatch	Handles IRPs with one or more major function codes.
Install Filter	Adds the rules to the filter function.
Firewall-Hook Filter	Filtering function.
Start_Firewall_ Hook	Starts the filtering function.
Stop_Firewall_ Hook	Stops the filtering function.
Uninstall Filter	Cleans up so that the driver can be unloaded.

5. CONCLUSIONS

An approach to the integration of security, reliability, and performance has been devised. It is based on the observation that a firewall system could be described in distributed and protective terms. A distributed viewpoint is related to the reliability and availability aspects. A protective viewpoint describes how to protect the network from inside as well as outside. Also implementing the firewall-hook driver to manage network packets constitutes a low down solution in the network stack that could enhance the overall firewall performance significantly.

The proposed Distributed Reliable Firewall system provides:

- Independent firewalls (represented by the clients), which can provide a security within the network by filtering directly at the client.
- Centralized management for the distributed firewalls with redundancy capability also enhances the system reliability and ensures that each client carries its work without any interruption even if the management (controller) fails.
- There is no longer a single chokepoint or a single point of failure that can isolate an entire network from both performance and availability standpoint. This is done by using the distribution and redundancy (controller's fail-over mode) approaches.
- An increase in the overall system speed is achieved by increasing the speed of clients' filtering operations by implementing rules optimization.
- Secure channels for transferring controlling signals and rules between the controller and the clients; these channels are created by implementing message encryption and authentication.

Future works will focus on the following directions:

- Implementing a content filtering, that can read the content of the packets and base their decisions upon specific contents (word(s), whole line(s) and URL(s)).

- Installing the client part automatically from the Server side without the need for local installation.
- Storing the entire clients' rules in an SQL database server. This would facilitate retrieving and can be used as a filtering rules knowledge base.

REFERENCES

[1] Cheswick W. R., Bellovin S. M., and Rubin A. D., *Firewalls and Internet Security*, 2nd Edition, Addison-Wesly, 2003.

[2] Pan Chi-Chien, Yang Kai-Hsiang, and Lee Tzao-Lin, " Secure Online Examination Architecture Based on Distributed Firewall", *IEEE International Conference on e-Technology, e-Commerce and e-Service*, pp. 533-536, March 2004.

[3] Verma Paven, and Prakash Atul, "FACE: A Firewall Analysis and Configuration Engine", *Proceedings on Applications and the Internet (SAINT"05)*, pp. 74-81,31 Jan.-4 Feb. 2005.

[4] Markham Tom, and Payne Charlie, "Security at the Network Edge: A Distributed Firewall Architecture", *Proceeding of the DARPA Information Survivability Conference and Exposition (DISCEX II*), Anaheim, CA, June 2001.

[5] Bellovin S. M., "Distributed Firewalls", *login: magazine, special issue on security*, pp.37-39, November 1999.

[6] Smith N. Robert, " Cascade of Distributed and Cooperating Firewalls in a Secure Data Network", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 15, No. 5, pp. 1307-1315, September/October 2003.

[7] Meredith Lynn M., "A Summary of the Autonomic Distributed Firewalls (ADF) Project", *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX'03)*, Washington DC, April 2003.

[8] Ioannidis, S., Keromytis, A. D., Bellovin S. M., and Smith J. M, "Implementing A distributed Firewall", *7th ACM Conference on Computer and Communications Security*, Athens, GREECE, pp. 190-199, November 2000.

[9] Blaze M., Feigenbaum J., Ioannidis J., and Keromytis A. D., "The Keynote Trust Management System Version 2", *Internet RFC 2704*, September 1999.

[10] Payne Charles, "Architecture and Applications for a Distributed Embedded Firewall", *Proceeding of the 17th Annual Computer Security applications Conference (ACSAC'01)*, IEEE Computer Society, December 2001.

[11] Liu Alex X.and Gouda Mohamed G.," Diverse Firewall Design", *Proceedings of the International Conference on Dependable Systems and Networks (DSN'04)*, IEEE Computer Society, September 2004.

[12] Ihde Michael and Sanders W. H., "Barbarian in the GATE: An Experiment Validation of NIC-based distributed Firewall performance and Flood tolerance", *Proceedings of the International Conference on Dependable Systems and Networks (DSN'06)*, pp. 209-216, 2006.

[13] Boughaci D., Drias H., Oubeka B., Aissioui A., and Benhamou B.," A Distributed Firewall Using Autonomous Agents", *Proceedings of the International Conference on Dependability of Computer Systems (DEPCOS-RELCOMEX"06)*, pp. 256-263, May 2005.

[14] Thomas J. Lane, and Abler Randal, " Implementing Distributed Internet Security Using A Firewall Collaboration Framework", *Southeast Conference*, IEEE, pp.680-685, March 2007.

[15] Smirnov V.Vadim, " Firewalls for Windows 9x/NT/2000", 2001. http://www.ntkernel.com/articles/firewaaeng.shtml

[16] Microsoft Windows Driver Development Kits, 2003. http://www.microsoft.com/whdc/ddk/winddk.mspx