

國立交通大學

資訊科學系

碩士論文

整合式網路安全管理程式庫之實作



An Integrated Approach to
Network Security Management Library

研究生：蔡宗穎

指導教授：袁賢銘 教授

中華民國九十六年七月

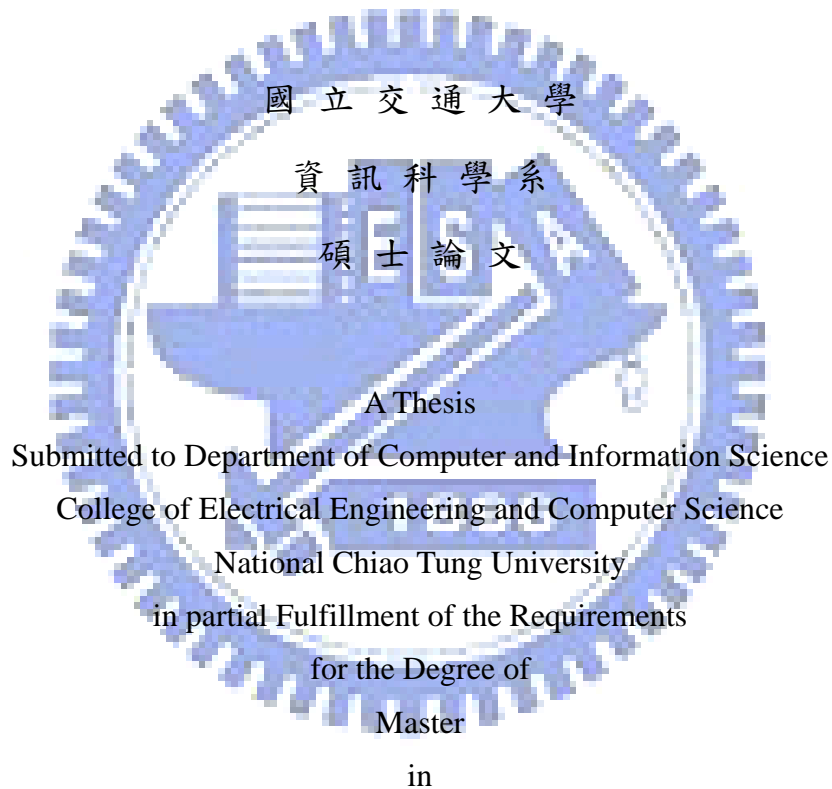
整合式網路安全管理程式庫之實作
An Integrated Approach to
Network Security Management Library

研究生：蔡宗穎

Student：Tzong-Yiing Tsai

指導教授：袁賢銘

Advisor：Shyan-Ming Yuan



Computer and Information Science

July 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年七月

中華民國 九十六 年 七月

整合式網路安全管理程式庫之實作

研究生：蔡宗穎

指導教授：袁賢銘



網路的應用在今日已相當普及。不同於個人應用，對企業而言，網路的管理和安全性議題顯得格外重要。每日都有新的安全性漏洞不斷的被發現，因此，很難找到一個一勞永逸的解決方案。

在本篇論文中，我們將著重於：流量控管，防火牆，和即時網路資料分析這三個方面，建構一個網路安全管理程式庫（NSML）。開發人員可以藉由使用本程式庫來縮短開發一個網路安全管理相關軟體之時程。亦可應用本文所提出的網路架構，保護企業中的重要資源，並確保其服務品質。

An Integrated Approach to Network Security Management


Student: Tzong-Yiing Tsai

Advisor: Dr. Shyan-Ming Yuan

Department of Computer Science

National Chiao Tung University

Abstract



Nowadays, Internet is so popular not only to individuals but also to enterprises. For enterprises, the management and security issues are very important. IDC estimates worldwide security software/services will grow 20% a year (to 43 billion US dollars in 2008). There are new security problems discovered every day. Meanwhile, hacker grows as the defender grows. Therefore, the security solutions will never be perfect.

In this paper, we focus on bandwidth control, application-firewall, and real-time traffic analysis to build a network security management library, NSML, which can acts as a stand alone library to saving the time to develop network security software through its API, and provides architecture to manage and protect the networks in an enterprise. NSML saves the time to develop a network security software and also reduces its complexity. By using our architecture, critical resources in enterprise network are protected and their quality of service are also guaranteed.

Acknowledgement

首先我要感謝袁賢銘教授給我的指導，在我研究的過程中不斷和我討論，並給予我很多意見，提供我新的思考方向，讓我的研究更加完善。此外，要特別感謝指導我的邱繼弘學長，在一起研究的過程中帶領著我，並幫助我解決了許多實作上的難題，感謝學長的熱心指導，也感謝其他博士班學長給我的幫助。另外要感謝實驗室的碩二同學們，和我一起渡過這兩年的時間，一起研究。此外，我想感謝通識中心師長和朋友們，在工讀的這兩年裡給我許多幫助，當我因論文而煩惱時，也及時地給我我最大的關懷。最後，我要感謝我的家人給予我的支持，讓我一直良好的求學環境中成長而無後顧之憂。謝謝你們。



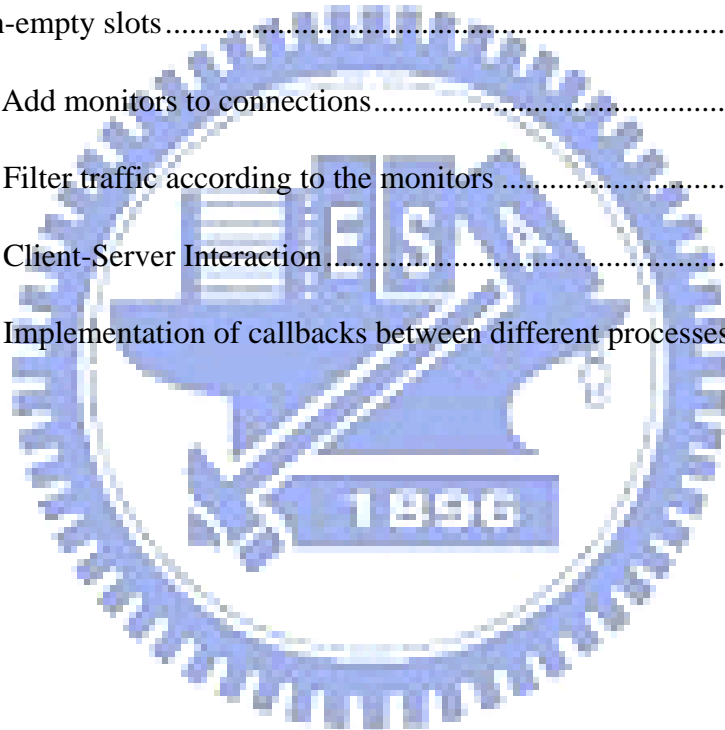
Table of Contents

| | |
|--|------------|
| Acknowledgement | iii |
| Table of Contents | iv |
| List of Figures..... | vi |
| List of Tables..... | vii |
| 1 Chapter 1 Introduction..... | 1 |
| 1.1 Preface | 1 |
| 1.2 Motivation | 1 |
| 1.3 Research Objectives | 2 |
| 1.4 Research Contribution..... | 3 |
| 1.5 Thesis Organization..... | 4 |
| 2 Chapter 2 Background | 5 |
| 2.1 Network Security Management Features | 5 |
| 2.1.1 Firewall and Application Firewall..... | 5 |
| 2.1.2 Bandwidth Control | 5 |
| 2.1.3 Real-time Traffic Analysis..... | 6 |
| 2.2 Microsoft Network Access Protection, NAP..... | 6 |
| 2.3 Microsoft Winsock Service Provider Interface, Winsock SPI..... | 8 |
| 2.4 Related Works..... | 9 |
| 3 Chapter 3 System Architecture..... | 10 |
| 3.1 NSDLL | 10 |
| 3.2 NS Software | 11 |
| 3.3 NSLib | 11 |
| 3.4 Shared Tables..... | 12 |

| | | |
|----------|--|-----------|
| 3.5 | CoreService | 12 |
| 3.6 | NSServer | 12 |
| 4 | Chapter 4 Implementation Details | 13 |
| 4.1 | Shared Tables..... | 13 |
| 4.1.1 | Shared Table for Rules | 13 |
| 4.1.2 | Shared Table for Logs | 15 |
| 4.1.3 | Avoid Race Condition | 16 |
| 4.2 | Rules..... | 17 |
| 4.2.1 | LegalConnRule and IllegalConnRule | 17 |
| 4.2.2 | BindingRule | 18 |
| 4.2.3 | SignatureRule..... | 18 |
| 4.3 | AlertLog and TrafficLog | 20 |
| 4.4 | Client-Server Interaction in Server Mode | 21 |
| 4.5 | The Performance Issue | 22 |
| 4.6 | Implementation of Callbacks between Different Process Spaces | 24 |
| 4.7 | NSServer and Database Schema | 25 |
| 5 | Chapter 5 Programming Interface and Performance Evaluation..... | 30 |
| 5.1 | NSLib Application Programming Interface | 30 |
| 5.2 | Performance Evaluation | 34 |
| 6 | Conclusion and Future Works..... | 35 |
| 6.1 | Conclusion..... | 35 |
| 6.2 | Future Works | 37 |

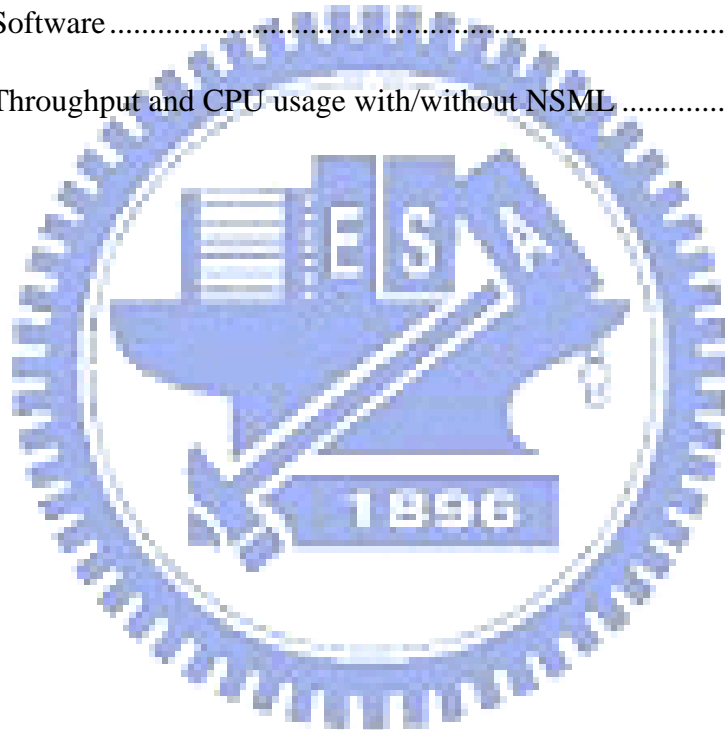
List of figures

| | |
|--|----|
| Figure 2-1 NAP States | 7 |
| Figure 2-2 NAP Server and Client Architecture | 8 |
| Figure 2-3 Winsock SPI..... | 9 |
| Figure 3-1 System Architecture | 10 |
| Figure 4-1 The memory arrangement of CLogTable, shaded slot stands for non-empty slots..... | 15 |
| Figure 4-2 Add monitors to connections..... | 19 |
| Figure 4-3 Filter traffic according to the monitors | 20 |
| Figure 4-4 Client-Server Interaction..... | 21 |
| Figure 4-5 Implementation of callbacks between different processes..... | 25 |



List of figures

| | |
|--|----|
| Table 4-1 BindingRule | 26 |
| Table 4-2 LegalConnRule and IllegalConnRule | 26 |
| Table 4-3 SignatureRule | 27 |
| Table 4-4 AlertLog | 28 |
| Table 4-5 TrafficLog | 28 |
| Table 4-6 Software | 29 |
| Table 5-1 Throughput and CPU usage with/without NSML | 34 |



Chapter 1 Introduction

1.1. Preface

Nowadays, Internet is so popular. We check our emails online, read news from the websites, shopping on internet, share files using P2P software, etc. Therefore, for an Enterprise, it is important to manage the Internet well and make it secure. IDC estimated worldwide security software/services will grow 20% a year (to 43 billion US dollars in 2008). There are new security problems discovered every day. Hacker grows as the defender grows. The security solutions will never be perfect.

1.2. Motivation

Network security contains many features such as traffic analysis, bandwidth control, firewall, reverse firewall, spy wares and Trojans detection, etc. There are so many software developed on each features, such as Spybot Search & Destroy [1], Bandwidth Controller [2], yet either they are commercial, or they just solve single problem. Also, those who want to develop their own network security applications have to build the applications step by step from the beginning. Almost all the security software monitor and control the network by means of hacking the operating system's network stack in certain level. Some of them are in driver level and some in application layer. The developer has to hook the network himself to get the information. We need a comprehensive solution. If there is a library which wrap the hooking details and provide simple APIs to let developers get the information and

control, the time is saved. On the other hand, enterprise needs a good network security solution to manage the usage of network, protect company critical resource, and monitor network events. Therefore, we have the idea to build a network security management library, NSML.

1.3. Research Objectives

There are three objectives in this paper: reliability, flexibility, easy-to-use, and performance.

Reliability

Reliability is surely an important issue for NSML. If it does not have high reliability, once it crashes all the network applications will be affected. It is inadmissible for enterprises to wait until the system is recovered. For enterprise, time is money. Moreover, because of the system hacking, if NSML is not stable enough, it is likely to crash the system in the worst situation. So only after the stability is guaranteed, NSML has its value.

Programmable Policy

NSML should not limit the action taken after we found an intrusion. NSML only decides whether the connection should be blocked or not according to the rules given by developers. All other detailed action is left to developers with structured

information about the violation. The developers are able to decide the reactions.

Easy-to-use

It is complex and not intuitive to hook the windows network through Winsock SPI [3]. The better way is wrap the hooking details and open up a series of simple APIs to let developers control the network through the APIs. The information we get using the API should be structured.

Performance

The performance of the client can be measured in two aspects: the network bandwidth and CPU usage. Though nowadays' network capability is strong enough to handle large amount of traffic, NSML still affects the network performance because we go through each connections and payloads. It takes CPU times to check the connections and payloads, so the system performance will be affected, too. Therefore, we should make the check as simple as possible while preserving the functionality.

1.4. Research Contribution

The contributions of this paper are:

1. We give a network security management library to saving the time and duplicated jobs to develop network security software. We also introduce a simple programming interface.

2. We craft a library to control the bandwidth, built up the firewall, and analyze the traffic in real-time.
3. We build a network security management system based on the client-server architecture.
4. We test NSML to guarantee the its performances.

1.5. Thesis Organization


In Chapter 2, we discuss the background of some intrusion detection/prevention frameworks such as Network Access Protection [4][5] and the core technology used to build our library. In Chapter 3, we show the system architecture of NSML. In Chapter 4, we explain the implementation details for each component, and how they cooperate. In Chapter 5, we introduce the programming interface of NSML and make performance evaluations. Finally, in Chapter 6, we give the conclusion and future works.

Chapter 2 Background

2.1. Network Security management Features

There are many different features in network security area, such as firewall, bandwidth control, traffic analysis, spy ware detection/removal, etc. In this paper, we discuss the former 3 issues. The last one, i.e. , spy ware detection/removal is not taken into consideration.

2.1.1. Firewall and Application Firewall



A firewall is a hardware or software designed to permit or deny data transmission to computers or devices with different trust levels. A simple firewall can only specify whether a connection from certain host port to another is legal or not. An application firewall (software) provides more information than a traditional firewall. It not only gives a way to monitor the payload in an application view rather than low level packet view, but also gives the relations of all incoming or outgoing messages with running applications. Therefore, NSML will build an application firewall.

2.1.2. Bandwidth Control

Though network cost is much lower than before, still large enterprises can afford the cost of unlimited network traffic. For smaller companies, maybe a 12M/2M ADSL is enough. However, due to the asymmetric download/upload bandwidth and the

popular P2P file sharing software such as eMule, BitTorrent, etc, the upload bandwidth often totally used by P2P software, which leads to the intolerably network speed. Therefore, the bandwidth should be controlled. This work is purposed last year by CH Chiu [6] in our lab (DCSLab of CIS NCTU).

2.1.3. Real-Time Traffic Analysis

Most virus and Trojans intrude the system through operating system's open service. For example, the Blaster worm [7] sends RPC request with buffer overflow and exploit code to TCP port 135 on Microsoft Operating Systems to open the backdoor. The message we received or we sent may contain malicious data. Hence, the monitor and filtering of traffic in real-time is necessary. There are many different real-time traffic analysis algorithm published like PAYL [8]. However, because the analysis is performed in real-time, it must lower the network performance and take CPU time. Therefore, it has always been a tradeoff to guarantee the system performance or to lower the false positive rate and false negative rate.

2.2. Microsoft Network Access Protection, NAP

Microsoft proposes the Network Access Protection [4][5] to provide an extendible framework for secure network environment. The framework periodically performs a series of "Health Check" on clients. Network administrator can define the policy to decide what client is said to be healthy. Once one of the health checks is not passed and violates the policy setting, the client is isolated from the network until it

goes through remediation process and then passes all the checks again. The check is made on client System Health Agents, SHAs, and checked on server side System Health Validators, SHVs. Both the development of component SHA and SHV is open to 3rd party companies. Therefore the network administrator can install the validators they need. The following is the state transition between different NAP client states.

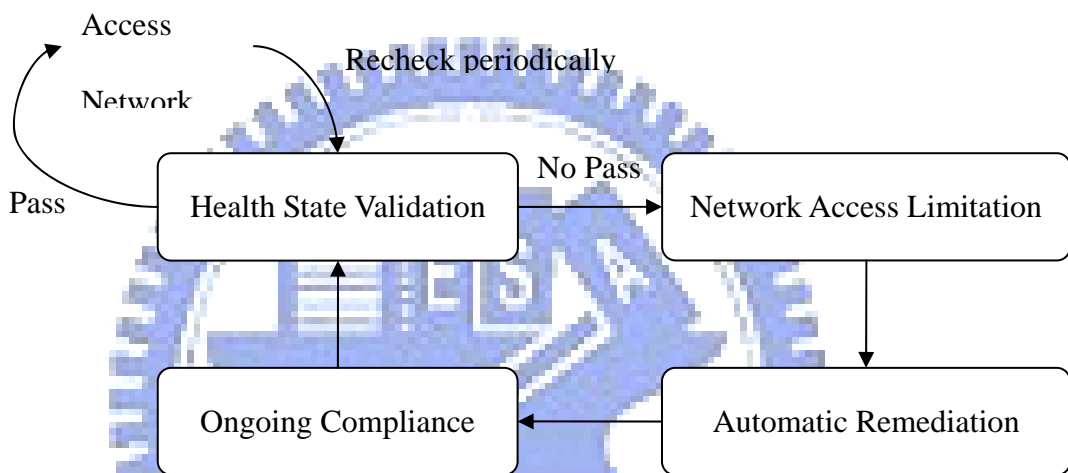


Figure 2.2.1 NAP States

Also, NAP defines two components: Quarantine Enforcement Client, QEC and Quarantine Enforcement Server, QES. Each QES in server side is corresponding to a QEC in client side. Each pair is defined for different type of network access. For example, there is a QES for DHCP configuration and a QES for VPN connections. The states of health gathered from SHA will be collect by QEC and transferred to QES, and then dispatched to corresponding SHV. The result gathered from SHVs will be applied to the policy setting to see if a client should be isolated or not. If it should be, QES will send a signal to the client’s QEC, then the client is isolated. The

following is figures from the NAP Architecture showing the relationship between the server and client. Currently, NAP supports DHCP, IPsec, and VPN connections.

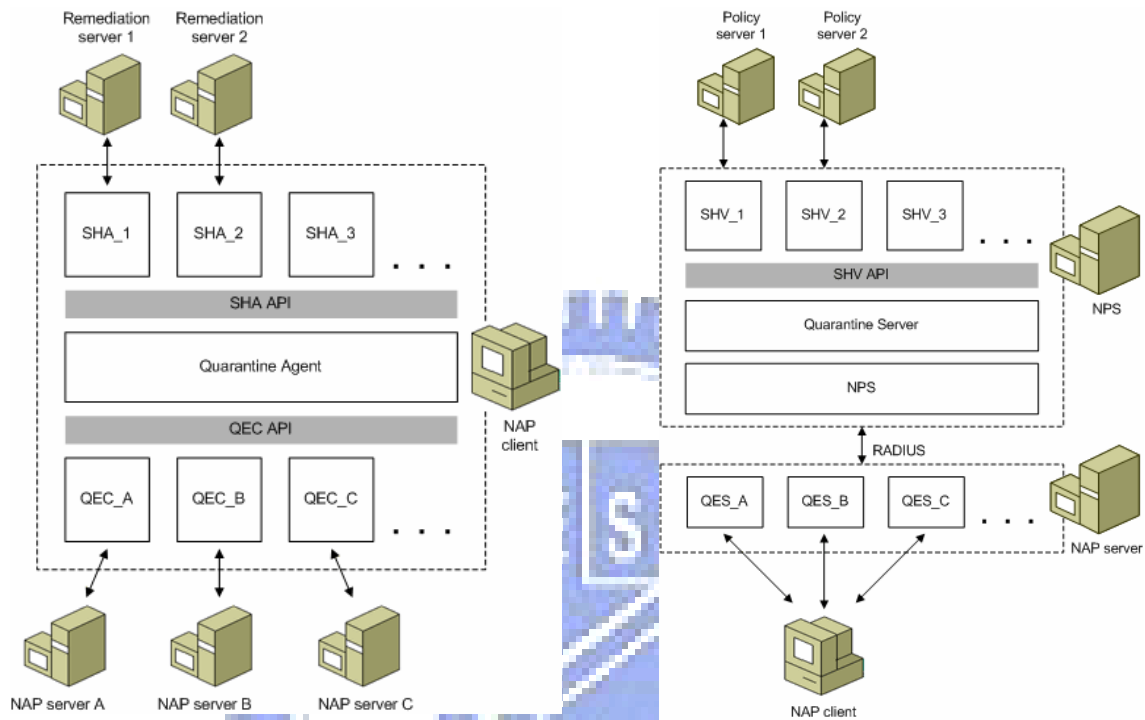


Figure 2.2.2 NAP Server and Client Architecture

NAP is a powerful framework to enhance the network security, but it cannot work alone, it needs the contributions of different validators.

2.3. Microsoft Winsock Service Provider API, Winsock SPI

The Winsock Service Provider Interface, or Winsock SPI, is a specialized interface of Winsock used to create providers. Traditional Winsock APIs have corresponding service provider APIs in SPI. On one hand, the network event/message will be passed to SPI before they are passed to Winsock Applications. On the other

hand, the event/message sent by the applications will also be passed to SPI before they are transmitted.

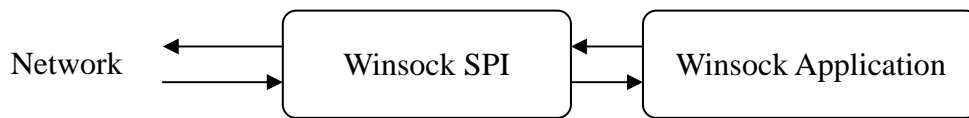


Figure 2.3 Winsock SPI

That is, we can hook Winsock through SPI functions. Each Winsock function has its corresponding hooking function in SPI. For example, a call to `send()` will be redirect to `WSPSend()` in SPI, developers can perform logics in `WSPSend()` to decide whether or not to let the `send()` call complete, or change its behavior. Because SPI is built on application layer, we can view connections in a high level rather than the packet level. Therefore, we know the application names, process ID, data buffers, etc, which common firewalls cannot do.

2.4. Related Works

Sygate [9] (acquisitioned by Symantec) is a complete solution about the network security issues, including personal firewall, real-time traffic analysis, secure remote desktop, etc. However, it lacks of bandwidth control and programmable policy. Also, it is commercial and not extendible. The detailed architecture and mechanism is not published.

Chapter 3 System Architecture

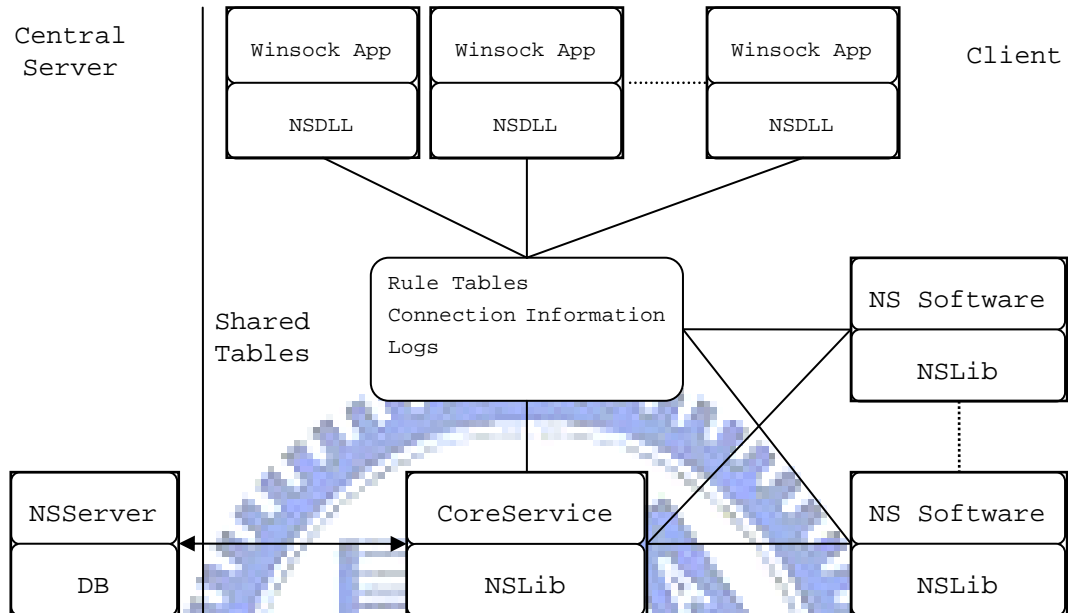


Figure 3.1 System Architecture

NSML runs in two modes: server mode and standalone mode. In server mode, new rules are pushed from server side to client side and applied immediately. The alert and traffic of each process will be logged. The log will be transferred to server side. In this mode, client does not take responsibility for storing the rules. In standalone mode, however, the rules should be stored and loaded locally.

3.1. NSDLL

A Winsock SPI filter appears as a dynamic linking library (DLL). Once the filter is installed, NSDLL will be injected to all Winsock applications. Every running Winsock applications have an instance of NSDLL. NSDLL works as a filter between

the Winsock application and the operating system. Every Winsock function calls will be passed to the corresponding hook function in the DLL. The DLL perform logic to decide whether to let the calls complete or change its behavior. Also, the DLL collects the traffic information and produce logs. The logic and logs are stored in shared tables. In the worst case, once the DLL found a malicious operation, it can close the dangerous connection immediately to protect the system.

3.2. NS Software

Network Security Software, NS Software, is applications using NSML to manage the network, for example, an application firewall with graphical user interface. Though the firewall logic is done by NSML, yet we leave the action taken after we've got a warning to the NS software be means of registering callback functions. The graphical firewall above may decide to pop a message box to alert the user that there is a rule violation or so. NS software can read structured network information from shared tables. In server mode, the rules are automatically loaded into shared tables through CoreService. However, in standalone mode, NS software should take the responsibility to store the rules locally and load them itself.

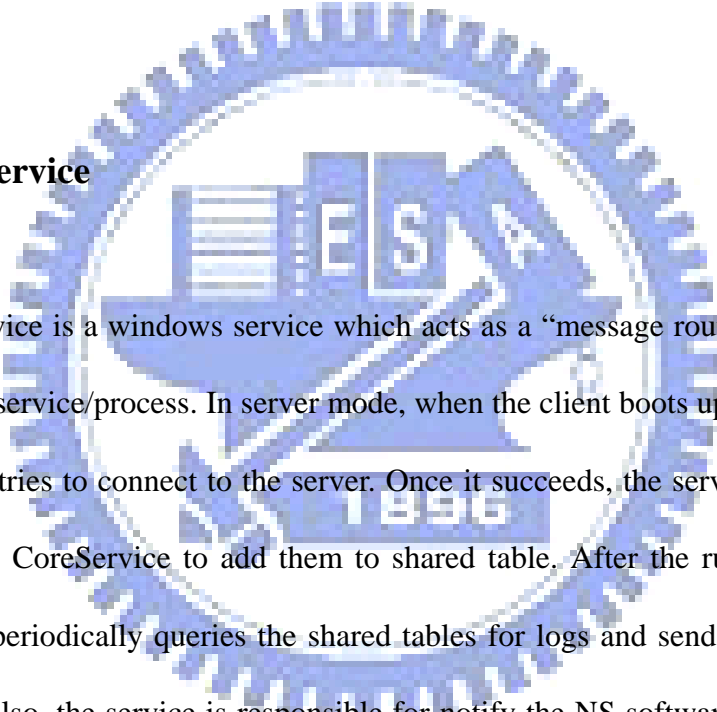
3.3. NSLib

NSLib provides the programming interface for NS software developers. Developers are able to read and manipulate the connections and rules. NSLib is also responsible for the communication between NS Software and CoreService.

3.4. Shared Tables

Shared tables are an essential part of NSML. Different kind of rules, current running connections, and logs are stored in the tables during runtime. NSDLLs look up the rules runtime to perform the logic. Therefore, the rules will be applied once they are added into the tables. Logs are generated by NSDLLs and stored here temporarily, waiting for CoreService to handle. Of course, NS software is able to read the logs, too.

3.5. CoreService



CoreService is a windows service which acts as a “message router” between the client/server, service/process. In server mode, when the client boots up and the service is running, it tries to connect to the server. Once it succeeds, the server will push the latest rules to CoreService to add them to shared table. After the rules are applied, CoreService periodically queries the shared tables for logs and sends the logs to the server side. Also, the service is responsible for notify the NS software that registered callbacks when corresponding rules are violated.

3.6. NSServer

NSServer is responsible for storing rules in Database and pushing rules to the clients in server mode. A tool is provided to manage the server and the clients.

Chapter 4 Implementation Details

In this chapter, we discuss the implementation details of NSML. We choose C++ as our programming language. Firstly, we discuss the design of different types of shared tables. Then, we introduce 4 kinds of rules: LegalConnRule, IllegalConnRule, BindingRule, and SignatureRule. Next, we give two kinds of logs: AlertLog and TrafficLog. Then, we talk over the Client-Server interaction in server mode. Moreover, we face the performance issue caused by the large amount of traffic logs, and propose the solution. Next, we discuss the implementation of callbacks between different process spaces. Finally, we introduce the server side data base schema.

4.1. Shared Tables

Because NSML hooks the applications in user mode, our system needs a method to share information between processes. Thus, shared memory is used to share rule and logs between In NSDLLs, NSLib, and CoreService. There are two types of shared table in our system: common shared table (for rules), and special shared table (for logs). They are alike in most ways, with slightly different policy for arranging elements.

4.1.1. Shared Tables for Rules

Due to the large size of rules and logs, we choose “Named File-Mapping Object” in Windows platform. Each shared table is assigned an unique ID. When a process

wants to access a table, it follows the following process to bind shared memory:

1. Try to open the file-mapping object with unique ID. If we succeed, go to step 3.
2. Because the object does not exist, we create a new file-mapping object with unique ID.
3. Map the object to acquire a memory reference.

Because the tables might be accessed by different processes concurrently, a unique mutex is opened to avoid the race condition for each shared table.

4. Try to open the mutex object with unique ID. If we fail, go to step 5.
5. Create the mutex object with unique ID.

Currently, the shared tables are implemented by fixed-length arrays. Each entry of the array is said to be a “slot”. The slot size is changeable for different tables. The shared memory we get from the system has logically continuous address. Therefore, we can treat the tables like a real array and the entry type can be any structure we defined. Each slot has a flag to specify whether the slot is used or not. To remove a rule, we just turn the slot flag off. To add a new rule, we need to find a slot with “off” flag. In this version, the array size is limited and the system needs to traverse the entire array to find certain element. The worst case to find certain element is $O(n)$.

To implement a new shared table, just define a new class inherit from CSharedTable class. The class handles the shared memory and mutex issue for the derived class.

4.1.2. Shared Tables for Logs

In order to reduce the load on the server side and improve the performance on the client side, we define another class CLogTable with the same shared memory technology as CSharedTable. The difference between CSharedTable and CLogTable is that entries in CLogTable must be contiguous while CSharedTable has no such restriction. The reason is due to the efficiency of reading logs. Logs are not like rules, they are generated constantly and should be ordered by time. If we apply the same policy as CSharedTable on CLogTable, the logs will be disordered and hard to analysis. Therefore, we arrange the slots to form a circular queue. In order to operate the circular queue, two indexes are used:

BeginIndex: point to the start of contiguous elements in the table.

EndIndex: point to the next available slot in the table.

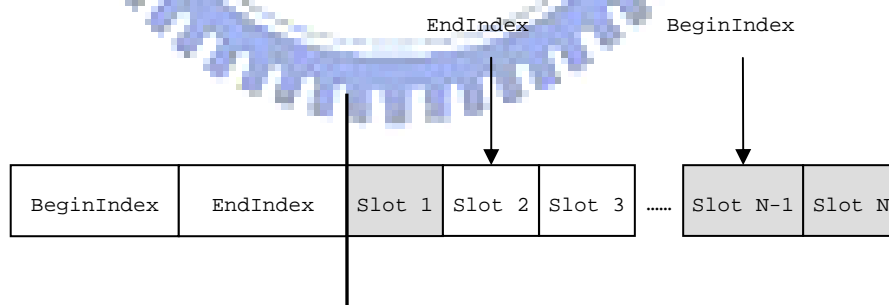


Figure 4.1.2 The memory arrangement of CLogTable,

shaded slot stands for non-empty slots

The element are pushed into the slot pointed by EndIndex and popped from

the slots BeginIndex.

If $\text{EndIndex}+1=\text{BeginIndex}$ the table is full. Also, because the elements are always pushed into EndIndex and popped from BeginIndex, the worst case to push or pop a log is $O(1)$.

CLogTable supports two protected functions:

`void *pushLog():` return a pointer to next available slot

`void *popLogs(int *size):` return a pointer to a contiguous block of logs and its size

To create a new log class, define a new class extend CLogTable and pass the table name, table size, and entry size to constructor of CLogTable. Also, define the specific function of pop and push, which invoke the original pushLog() and popLogs() internally and cast the pointer to specific type. Because the logs are stored in a circular queue and for performance concern, the logs should be returned may be divided into 2 blocks. 1 block ranges from the BeginIndex to the real end of the table, and 1 block ranges from the head of the table to EndIndex.

4.1.3. Avoid Race Condition

To avoid race condition of shared tables, we create unique mutex for each table. We lock the table before reading and writing, and unlock the table after the action is done. To avoid overhead, locking is not recommended if the rare condition never

happens according to the usage.

4.2. Rules

We define 4 kind of rules for different purposes. They are LegalConnRule, IllegalConnRule, BindingRule, and SignatureRule. Because we perform the rule logic in application layer, we know the Process ID, Process Name, etc, of the related Winsock Application. This gives us more power to write more specific rule according to different applications. For example, the browser IE should not connect to a destination port rather than 80. If it does, it might be intruded.

4.2.1. LegalConnRule and IllegalConnRule

LegalConnRule specifies what connections are legal and safe. It may be a rule for incoming message or outgoing message. On the other hand, IllegalConnRule specifies what connections are illegal. It also tells if the illegal connections should be blocked or not, and if we would like to get a warning or not. LegalConnRule is a “white list”, while “IllegalConnRule” is a “black list”.

To enhance the safety, we choose not to run the risk of being intruded. Therefore, the policy for LegalConnRule and IllegalConnRule is as following:

1. Does the connection appear in the “black list”? If it is, the connection is blocked or warned according to the rule. Both blocking and warning produce

AlertLogs. If it is not, go to step 2.

2. does the connection appear in the “white list”? If it is, it is regard as a safe connection. If it is not, we still let the connection pass the filter, and an AlertLog is produced to indicate it.

4.2.2. BindingRule

BindingRule is used to indicate what applications should not listen to certain ports. For example, the telnet.exe program should not bind on any port, if it does, we know it is infected. This is helpful in some cases, especially for IE. IE supports Browser Helper Object to let 3rd party developers add more functions like toolbar, etc. However, the user cannot usually tell whether a helper object is safe or not. They just install the modules they interested without knowing if there is a backdoor opened! If we set up a rule for IE so that it cannot bind on any ports, the problem is solved.

4.2.3. SignatureRule

SignatureRule specifies the pattern might be malicious. Usually, it takes much CPU time and lowers the network performance to perform complex analysis on the traffic in real-time. Therefore, we perform simple analysis on the traffic to guarantee the performance. Every SignatureRule defines 2 important parameters: pattern and maxOffset. Once a connection is established, NSDLL scans the SignatureRule Table for rules matching the source and destination. If the matching rules are found, two traffic monitors are added into the incoming monitor list and outgoing monitor list,

individually. Then, the connection begins to transfer data. Both the data received and the data sent will be checked for the pattern. Also, there is a offset counter initialized to zero and increase during the check. Once the counter exceeds the maxOffset of the rule, the monitors are removed from the incoming and outgoing monitor lists.

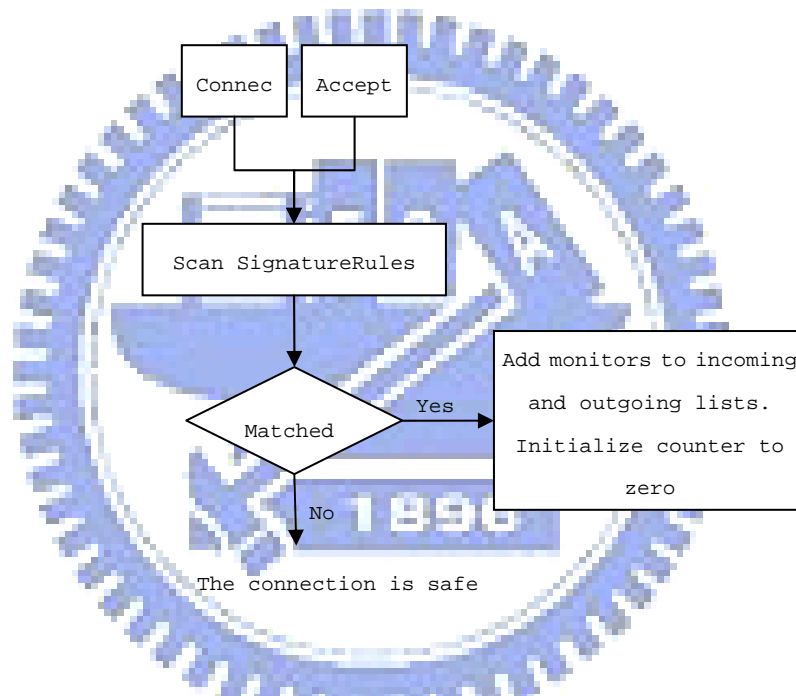


Figure 4.2.3.1 Add monitors to connections

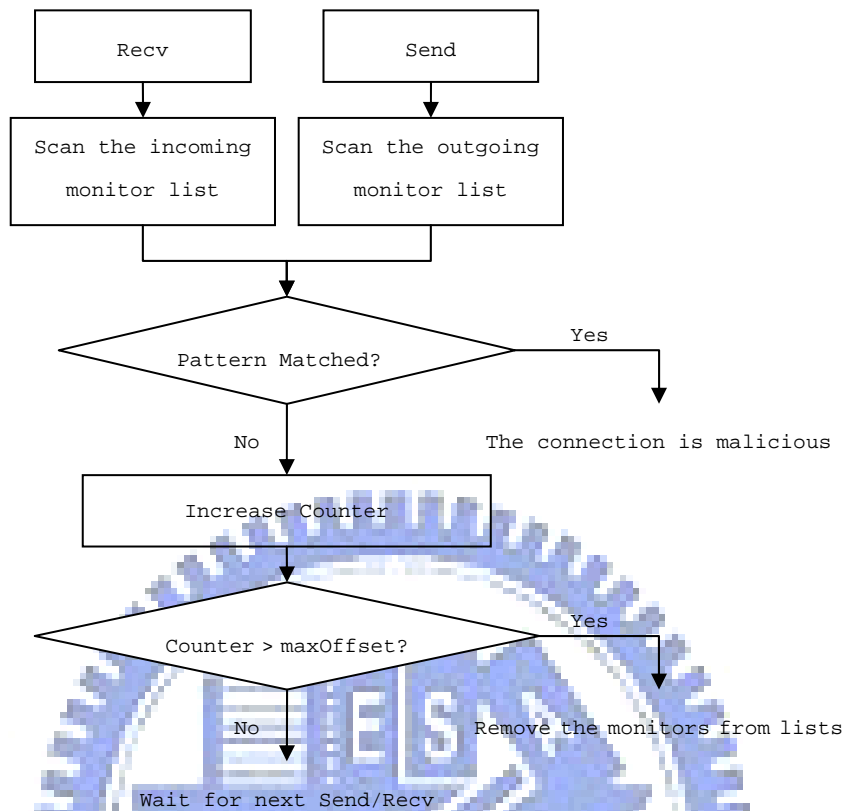


Figure 4.2.3.2 Filter traffic according to the monitors

4.3. AlertLogs and TrafficLogs

There are two types of pre-defined logs in NSML: AlertLogs and TrafficLog. Both AlertLogs and TrafficLogs are generated by NSDLL of the Winsock Application.

AlertLogs are generated when there are rules violated. For example, if we have a BindingRule specifies the browser IEXPLORER.EXE cannot bind on any port. Once IEXPLORER.EXE really binds on certain port, the AlertLog is produced to record this abnormal behavior.

TrafficLogs are produced when the application invokes one of the following Winsock function: socket(), closesocket(), accept(), connect(), bind(), send(), recv(), sendto(), recvfrom(). TrafficLogs record the detailed actions of Winsock applications in time order. If the system is intruded, we have clues to analysis the system's behavior.

4.4. Client-Server Interaction in Server Mode

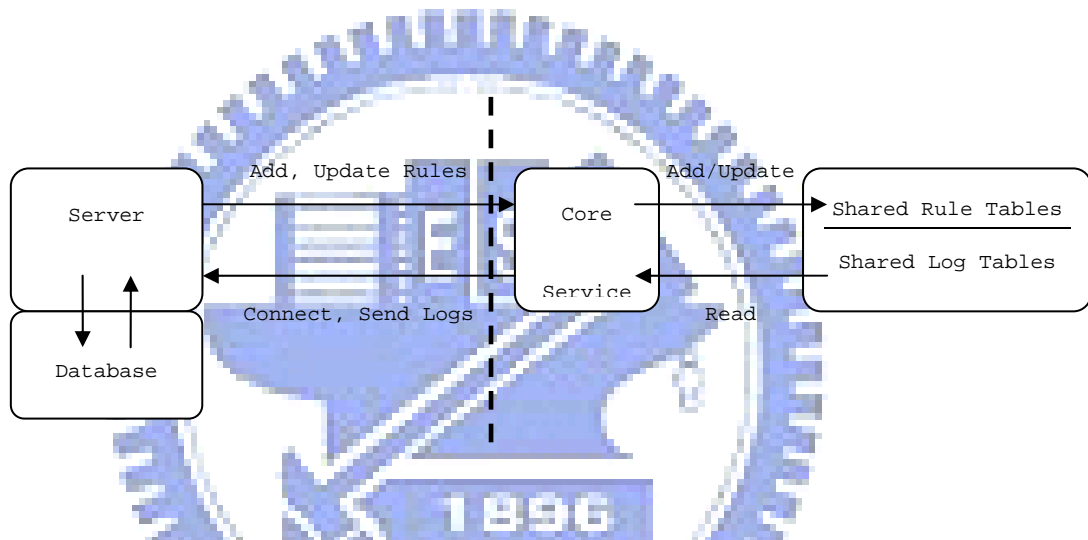


Figure 4.4 Client-Server Interaction

In server mode, the server is responsible for pushing and updating rule to each client. When the client computer boot up and the CoreService is running, CoreService will try to connect to NSServer. Once it succeeds, NSServer sends the RULE_ADD message and starts pushing rules to the client. The rules received by CoreService will be added to shared rule tables and applied by NSDLLs immediately. After the rules are all sent, NSServer sends a RULE_END message to tell the client it is over. If a existing rule is modified, NSServer sends RULE_UPDATE message to client. Then the updated rule is sent. After that, a

RULE_END message is sent. The NSServer maintains a list of online computers with NSML installed. Network administrator can use tool to monitor the network status of the clients.

4.5. The Performance Issue

As mentioned in Section 3.1, Inject DLL is responsible for producing traffic logs and alert logs to record the behavior of a Winsock Application. Those logs are necessary because the logs might be helpful if the system is intruded. The logs give the clues to find out what's wrong with our system. For AlertLogs, that is fine because AlertLogs are seldom produced. Only when the system finds a malicious connection the AlertLogs are produced. For TrafficLogs, However, a problem arises. Due to the high frequency of networking, NSDLL produces a large amount of TrafficLogs. To transfer all the logs to server side lowers the network performance significantly.

To solve the performance issue, we decide to shrink the size of TrafficLogs before they are really sent to the server side. As mentioned in Section 3.5, CoreService periodically queries Shared Log Tables for latest logs and sent them to the server side. After the TrafficLogs are popped from the TrafficLog table, we shrink the data size in the following steps:

1. TrafficLogs with types rather than SEND/RECV will not be shrunked.
2. Two TrafficLogs are said to be the same if they are both SEND type or both RECV type, generated by the same application, and the two logs have the

same source-destination pair.

3. for same TrafficLogs, the data transferred will be summed up.

For example, we have the following TrafficLogs:

| | | | |
|-------------|----------------------------------|-------|------|
| firefox.exe | localhost:4884-203.72.66.5:80 | RECV | 151 |
| firefox.exe | localhost:4884-203.72.66.5:80 | RECV | 180 |
| firefox.exe | localhost:4884-140.113.23.101:80 | RECV | 180 |
| telnet.exe | localhost:1477-140.113.23.101:23 | CONN | 0 |
| telnet.exe | localhost:1477-140.113.23.101:23 | SEND | 1010 |
| telnet.exe | localhost:1477-140.113.23.101:23 | SEND | 1030 |
| firefox.exe | localhost:4884-203.72.66.5:80 | RECV | 100 |
| telnet.exe | 140.113.23.101:23-localhost:1477 | RECV | 2010 |
| telnet.exe | localhost:1477-140.113.23.101:23 | SEND | 100 |
| firefox.exe | localhost:4884-203.72.66.5:80 | RECV | 150 |
| telnet.exe | 140.113.23.101:23-localhost:1477 | RECV | 510 |
| telnet.exe | localhost:1477-140.113.23.101:23 | CLOSE | 0 |
| firefox.exe | localhost:4884-140.113.23.101:80 | RECV | 100 |

After shrinking:

| | | | |
|-------------|----------------------------------|-------|------|
| firefox.exe | localhost:4884-203.72.66.5:80 | RECV | 581 |
| firefox.exe | localhost:3321-140.113.23.101:80 | RECV | 280 |
| telnet.exe | localhost:1477-140.113.23.101:23 | CONN | 0 |
| telnet.exe | localhost:1477-140.113.23.101:23 | SEND | 2140 |
| telnet.exe | 140.113.23.101:23-localhost:1477 | RECV | 2520 |
| telnet.exe | localhost:1477-140.113.23.101:23 | CLOSE | 0 |

After shrinking, the size of TrafficLogs significantly reduced and the network performance is therefore improved.

4.6. Implementation of Callbacks between Different Process Spaces

It is easy to implement callback functions in single process because all the functions are in the same logic memory space. However, each process owns its own addressing space, so it is hard to register callback functions between different processes. To solve this problem, we add 1 more shared table: SoftwareTable. When CoreService starts running, it creates a special named event object. Then it creates a new thread to wait the object to be signaled. When a NS Software starts running, NSLib will automatically add an entry to the SoftwareTable. Also, NSLib will record the callback functions locally and create a named event object. The event object name will be stored in the table, too. Next, NSLib creates a thread which waits for the event object to be signaled. Once a rule is violated, NSDLL will signal CoreService event object to inform CoreService to look up the SoftwareTable to find the corresponding name of event object to signal. After the thread in NSLib is signaled, the thread invokes the callbacks and then waits for another signal.

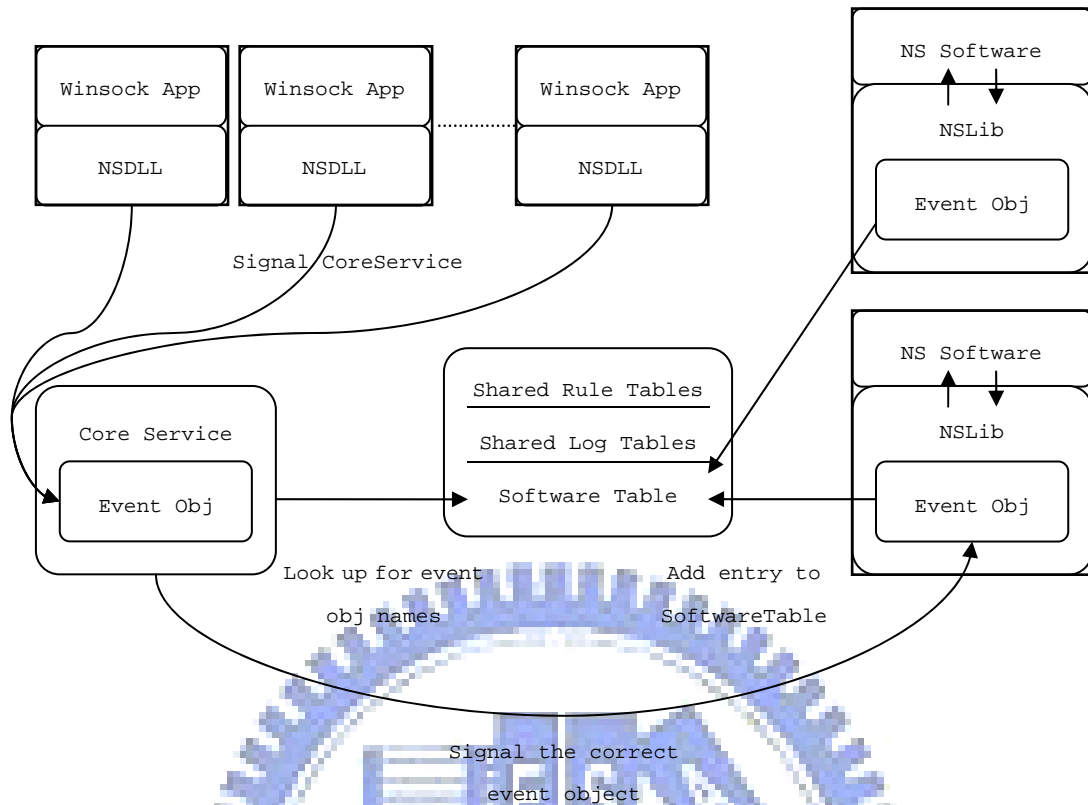


Figure 4.6 Implementation of callbacks between different processes

4.7. NSServer and Database Schema

In server mode, rule data are stored in a centralized database. In this section, we propose the database schema used in NSML. Currently, we choose MySQL [10] as our database. In the future, when the system is deployed to a large network environment, maybe some more powerful database like MSSQL would be a good choice. To access MySQL through C++, we also take the advantage of mysql++ [11], which is a powerful library to manipulate MySQL in C++.

There are 7 tables in the database schema, 4 tables for rules, 2 tables for logs, and 1 more table to record the NS Software information.

| Column Name | Datatype | NOT NULL | AUTO INC | Flags |
|-------------|-------------|-------------------------------------|-------------------------------------|--|
| rindex | INTEGER | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| hostip | INTEGER | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| protocol | SMALLINT(5) | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| serverport | SMALLINT(5) | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| processname | CHAR(20) | <input checked="" type="checkbox"/> | | <input type="checkbox"/> BINARY <input type="checkbox"/> ASCII <input type="checkbox"/> UNIC |
| sid | INTEGER | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |

Table 4.7.1 BindingRule

rindex: rule index, every rule has a unique rindex number. Rindex is used when there is an update about certain rule.

hostip: apply the rule only on certain host? Zero stands for all host.

protocol: what protocol is used, currently TCP only for BindingRule.

serverport: which port the application should not bind. Zero stands for all ports.

processname: which application should apply this rule. Null string stands for all applications.

sid: which NS software does this rule belong to.

| Column Name | Datatype | NOT NULL | AUTO INC | Flags |
|-------------|-------------|-------------------------------------|-------------------------------------|--|
| rindex | INTEGER | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| hostip | INTEGER | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| protocol | SMALLINT(5) | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| srcip | INTEGER | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| srcport | SMALLINT(5) | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| destip | INTEGER | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| destport | SMALLINT(5) | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| action | SMALLINT(5) | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| processname | CHAR(20) | <input checked="" type="checkbox"/> | | <input type="checkbox"/> BINARY <input type="checkbox"/> ASCII <input type="checkbox"/> UNIC |
| sid | INTEGER | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |

Table 4.7.2 LegalConnRule and IllegalConnRule

protocol: may be TCP or UDP

srcip: the connection source ip

sreport: the connection source port

destip: the destination ip of the connection

destport: the destination port of the connection

action: BLOCK or ALERT. Both BLOCK and ALERT produces AlertLogs, but ALERT only generate the alert and still let the connection there.

| Column Name | Datatype | NOT NULL | AUTO Inc | Flags |
|-----------------|-------------|-------------------------------------|-------------------------------------|--|
| rindex | INTEGER | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| hostip | INTEGER | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| protocol | SMALLINT(5) | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| port | SMALLINT(5) | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| signaturelength | INTEGER | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| maxoffset | INTEGER | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| action | SMALLINT(5) | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| processname | CHAR(72) | <input checked="" type="checkbox"/> | | <input type="checkbox"/> BINARY <input type="checkbox"/> ASCII <input type="checkbox"/> UNIC |
| signature | BLOB | <input checked="" type="checkbox"/> | | |
| sid | INTEGER | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |

Table 4.7.3 SignatureRule

signaturelength: length of the signature

maxoffset: the max offset where the connection should be monitored

signature: the signature

| Column Name | Datatype | NOT NULL | AUTO INC | Flags |
|-------------|-------------|-------------------------------------|-------------------------------------|---|
| idx | INTEGER | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| type | TINYINT(3) | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| ruletype | TINYINT(3) | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| rindex | SMALLINT(5) | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| processname | CHAR(20) | <input checked="" type="checkbox"/> | | <input type="checkbox"/> BINARY <input type="checkbox"/> ASCII <input type="checkbox"/> UNICODE |
| message | CHAR(100) | <input checked="" type="checkbox"/> | | <input type="checkbox"/> BINARY <input type="checkbox"/> ASCII <input type="checkbox"/> UNICODE |
| time | TIMESTAMP | <input checked="" type="checkbox"/> | | |
| sid | INTEGER | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |

Table 4.7.4 AlertLog

type: BLOCK or ALERT.

ruletype: the type of the violated rule

rindex: the rule index of the violated rule

message: additional message of this alert

time: when this alert is generated.

| Column Name | Datatype | NOT NULL | AUTO INC | Flags |
|-------------|-------------|-------------------------------------|-------------------------------------|---|
| idx | INTEGER | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| hostip | INTEGER | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| protocol | TINYINT(3) | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| srcip | INTEGER | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| srcport | SMALLINT(5) | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| destip | INTEGER | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| destport | SMALLINT(5) | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| length | INTEGER | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| action | TINYINT(3) | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| processname | CHAR(20) | <input checked="" type="checkbox"/> | | <input type="checkbox"/> BINARY <input type="checkbox"/> ASCII <input type="checkbox"/> UNICODE |
| time | TIMESTAMP | <input checked="" type="checkbox"/> | | |
| sid | INTEGER | <input checked="" type="checkbox"/> | | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |

Table 4.7.5 TrafficLog

length: the data length of bytes transferred

action: SOCK_CREATE,SOCK_CLOSE,SEND,RCV,SENDTO, RECVFROM, BIND, etc.

| Column Name | Datatype | NOT NULL | AUTO INC | Flags |
|-------------|----------|-------------------------------------|-------------------------------------|--|
| sid | INTEGER | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL |
| name | CHAR(20) | <input checked="" type="checkbox"/> | | <input type="checkbox"/> BINARY <input type="checkbox"/> ASCII <input type="checkbox"/> UNIC |

Table 4.7.6 Software

sid: the unique id of this software

name: the software name



Chapter 5

Programming Interface and Performance Evaluation

5.1 NSLib Application Programming Interface

NSLib

- NSLib(char *softwareName);
- AddSignatureRule(SianatureRule*);
- AddBindingRule(BindingRule*);
- AddLegalConnRule(LegalConnRule*);
- AddIllegalConnRule(IllegalConnRule*);
- RemoveSignatureRule(int rindex);
- RemoveBindingRule(int rindex);
- RemoveLegalConnRule(int rindex);
- RemoveIllegalConnRule(int rindex);
- UpdateSignatureRule(SianatureRule*,int rindex);
- UpdateBindingRule(BindingRule*,int rindex);
- UpdateLegalConnRule(LegalConnRule*,int rindex);
- UpdateIllegalConnRule(IllegalConnRule*,int
rindex);
- SetSignatureRuleHandler(SIGNATURE_RULE_HANDLER);
- SetBindingRuleHandler(BINDING_RULE_HANDLER);
- SetLegalConnRuleHandler(LEGALCONN_RULE_HANDLER);
- SetIllegalConnRuleHandler(ILLEGALCONN_RULE_HANDL

```

        ER);
-   SetAlertLogHandler(ALERTLOG_HANDLER);
-   SetTrafficLogHandler(TRAFFICLOG_HANDLER);

typedef void (*SIGNATURE_RULE_HANDLER)(SignatureRule*);
typedef void (*BINDING_RULE_HANDLER)(BindingRule*);
typedef void (*LEGALCONN_RULE_HANDLER)(LegalConnRule*);
typedef void (*ILLEGALCONN_RULE_HANDLER)(IllegalConnRule*);
typedef void (*ALERTLOG_HANDLER)(AlertLogEntry*,int size);
typedef void (*TRAFFICLOG_HANDLER)(TrafficLogEntry*,int size);

```

NSLib API are designed as simple as possible. These functions can be used to develop network security software either in standalone mode or in server mode. Now let's go through them:

NSLib(char*):

The constructor is responsible for initializing the environment. For example, it creates the event object which we mentioned in Section 4.6, and register itself to the Software Shared Table. Also, it creates another thread to wait for signals to execute the callback functions. The only one parameter is the name of the software.

SetSignatureRuleHandler(SIGNATURE_RULE_HANDLER):

SetBindingRuleHandler(BINDING_RULE_HANDLER):

SetLegalConnRuleHandler(LEGALCONN_RULE_HANDLER):

SetIllegalConnRuleHandler(ILLEGALCONN_RULE_HANDLER):

The four functions are used to set up the callback functions. Each of them accepts a callback function pointer as the parameter. When a rule violation is encountered, according to its type, the corresponding callback function is invoked, and the rule violated is passed in.

AddSignatureRule(SianatureRule*):

AddBindingRule(BindingRule*):

AddLegalConnRule(LegalConnRule*):

AddIllegalConnRule(IllegalConnRule*):

The four functions are used only in standalone mode. In standalone mode, NS software is responsible for storing the rules physically on the disk. Therefore, when NS software is initialized, it should load the rule from the disk and add them into the shared tables.

RemoveSignatureRule(int rindex):

RemoveBindingRule(int rindex):

RemoveLegalConnRule(int rindex):

RemoveIllegalConnRule(int rindex):

These functions remove rules from the shared rule tables with given rindex. Every slot of rule tables has a flag to indicate if this slot is used or not. The remove operation simply sets the flag off. Therefore, it's efficient to remove the rules.

UpdateSignatureRule(SianatureRule*,int rindex):

UpdateBindingRule(BindingRule*,int rindex):

UpdateLegalConnRule(LegalConnRule*,int rindex):

UpdateIllegalConnRule(IllegalConnRule*,int rindex):

These 4 functions are used to update a existing rule in rule tables. The developer needs to specify the rindex to indicate which rule is going to be overwritten.

SetAlertLogHandler(ALERTLOG_HANDLER):

SetTrafficLogHandler(TRAFFICLOG_HANDLER):

Because the memory space is limited, we cannot store all the logs in shared memory. We only keeps the latest logs in shared memory. To prevent the tables from being full, the logs need to be consumed periodically in standalone mode. Therefore, the developer should set two callbacks which will be invoked periodically. There are two parameters passed into the callback functions: an array of log entries, and the size of the array.

5.2 Performance Evaluation

Because we need to process the rule logic in real-time, NSML causes some performance overhead. We test its performance by download a big file (697 MBs) using FTP. There are 3 condition tested: 1) without NSML, 2) with NSML but without rules, and 3) with NSML and signature rules.

| | CPU Usage | Throughput |
|-------------------------------------|-----------|------------|
| (1) Without NSML | 16.5% | 7.61 MB/s |
| (2) With NSML, without rules | 17.1% | 7.22 MB/s |
| (3) With NSML, with signature rules | 24.2% | 6.58 MB/s |

Table 5.2 Throughput and CPU usage with/without NSML

With NSML but without rules, the throughput is lower the original. This is because NSML produces and transfers real-time traffic logs. With NSML and signature rules, the throughput becomes 6.98 Mbps due to real-time traffic analysis and traffic logs. From the result, we can see the impact to the network performance caused by NSML is small and tolerable.

The difference of CPU usage between (1) and (2) is 1.6%, which is small. The CPU usage in (3) arises to 24.2% because the CPU needs to perform algorithms to analysis the traffic. Smaller signature leads to smaller CPU usage.

Chapter 6 Conclusions and Future Works

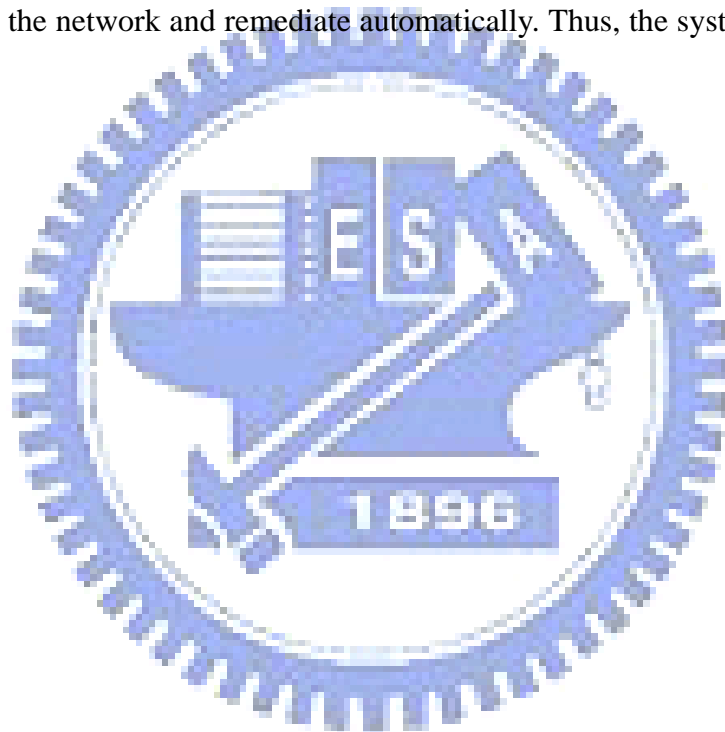
6.1 Conclusion

It is hard to propose a comprehensive solution for network security management problem. We try to find out a possible way to find an integrated solution for it. We focus the design of NSML in 4 aspects: reliability, programmable policy, easy-to-use, high performance. In this paper, NSML introduced an open framework to provide application firewall, bandwidth control, real-time traffic analysis, and client-server architecture to solve the security and manage of network. With the help of NSML, developers can easily develop the network security software they want. This significantly reduces the hardness and time-to-market of network security software development. Also, with the help of the framework, the improper use of network by users or malicious programs in the office can be detected and managed. The quality of service of critical network applications can be guaranteed.

6.2 Future Works

Currently, we use a simple algorithm to perform traffic analysis. However, to make the analysis more powerful, we must change to a more powerful analysis method while preserving the high performance. We need help of statistic algorithms. PAYL [8] may be one of our choices. Besides, current version of NSML does not take spy ware detection / removal into consideration although their strange behavior might be found by the rules. It is a good idea to add spy ware detection feature into the next version of NSML.

Microsoft and Cisco announced the NAP-NAC[12] Interoperability Architecture in Sep. 2006, which takes the advantage of NAP's software framework extensibility and NAC's hardware capability. This paper focused on developing a stable, flexible, easy-to-use, and high performance open network security management library. However, to enhance the security further, it is necessary to port NSML into NAP framework. With the help of NAP framework, all intruded computers can be totally isolated from the network and remediate automatically. Thus, the system will be more secure.



Bibliography

- [1] Spybot Search & Destroy,
<http://www.safer-networking.org/ct/index.html>
- [2] Bandwidth Controller,
<http://bandwidthcontroller.com/>
- [3] Winsock Service Provider API,
<http://msdn2.microsoft.com/en-us/library/ms741418.aspx>
<http://msdn2.microsoft.com/en-us/library/ms741424.aspx>
<http://msdn2.microsoft.com/en-us/library/ms741424.aspx>
- [4] Microsoft Cooperation, Network Access Protection Introduction, April 2007 Updated
<http://download.microsoft.com/download/8/d/9/8d9b3e54-6db7-4955-9e36-58a3f0534933/NAPIntro.doc>
- [5] Microsoft Cooperation, Network Access Protection Architecture, April 2007 Updated
<http://download.microsoft.com/download/3/9/f/39ff0ca3-56d1-4d93-af46-98f92134d040/NAPArch.doc>
- [6] CH Chiu, SM Yuan, Managing the Network Usage of Applications,
- [7] M Bailey, E Cooke, D Watson, F Jahanian, J Nazario - IEEE Security & Privacy, 2005
- [8] Ke Wang, Salvatore J. Stolfo, Anomalous Payload-Based Network Intrusion Detection, Recent Advances in Intrusion Detection, 2004
- [9] Symantec Sygate Enterprise Protection,
http://www.symantec.com/zh/tw/enterprise/products/overview.jsp?pcid=1001&pvid=1303_1
- [10] MySQL, <http://www.mysql.com/>
- [11] MySQL++, <http://tangentsoft.net/mysql++/>

- [12] Microsoft Cooperation and Cisco System, Inc, Joint Architecture for NAC-NAP Interoperability
<http://www.microsoft.com/presspass/press/2006/sep06/09-06SecStandardNACNAPPR.msp>
- [13] Microsoft Windows Internals, Fourth Edition: Microsoft Windows Server(TM) 2003, Windows XP, and Windows 2000 (Pro-Developer) by Mark E. Russinovich and David A. Solomon (Dec 8, 2004)
- [14] Winsock 2.0 by Lewis Napper (Nov, 1997)
- [15] Network Programming for Microsoft Windows, Second Edition by Jim Ohlund, Anthony Jones, and James Ohlund (Feb 13, 2002)
- [16] Seyed M.M., Tahaghoghi, Hugh Williams, Learning MySQL, O'Reilly, Nov 2006

