A Novel User Authentication and Logging System for IIS Web Sites

By

Major Farooque Azam

Dedications

Index

List Of Tables

List Of Figures

Abstract

Today many sites on the World Wide Web require identification of a user and have many reasons for such identifications.. Subscription oriented sites need to verify identity of the user or often corporate intranet sites need to secure access to propriety information, also many Web sites sometime wish to tailor services to the preferences of the individual users. Microsoft provides standard verification mechanism in their product Internet Information Server (IIS), which uses Windows NT security system to manage rights and identification scheme by utilizing familiar tools such as *NT User Manager* and *File Manager*. However, IIS is unsuitable for Internet use when large no of Internet users are required to be identified and provided the permissions individually. In the existing scenario the main objective of the project under consideration is to develop the authentication software that works independent of the IIS and NT interface and access control functions, but still be able to provide directory and file level security. In Army, keeping in view the requirements of handling the information safely, so that no unauthorized person should be able to access in confidential information, such system will provide a sound footing for developing any extra ordinary system as per the specific requirement of in department.

Chapter1
Introduction

In this chapter first of all the limitations of Windows NT and Internet Information server (IIS) in relation to the authentication of secure Web sites are discussed. Based on these limitation objectives of the proposed Novel Authentication Design are outlined. Then the benefits that are achieved by the objectives will be highlighted. Finally, the outline of the thesis is described.

1.1 Problem Definition

Authentication is the process of obtaining and verifying the identity of user and a key to building access control system for a Web site. Microsoft [1] has implemented its standard authentication mechanism in the Internet Information Server (IIS). The standard implementation of this mechanism in IIS uses the Windows NT security system to manage users, rights and identifications. This allows the administrators of IIS based systems to manage their server by using familiar tools such as the *NT User Manager* and *File Manager*. Unfortunately, using these familiar tools is not always the ideal solution, because of the reasons described in this section: -.

1.1.1 Adds To The Work Of System Manager

IIS uses the file base security of NT. Therefore, it adds to the work of system manager. Since a manager not only has to manage the local accounts of the users on a local Intranet but also uses the same *User Manager* to manage the accounts of users who access the server over the Internet. Therefore, system manager ambiguously differentiates between local and Internet users, as a result, the process of allocation of user rights becomes prone to many errors and most of the time security loop holes are detected in the Web server, which invite many hackers to experiment with that Web site.

1.1.2 Number Of Users

The number of users who access the server on Internet is generally very large, which puts extra burden on the NT database and effects the general performance of the operating

system. The IIS security system works well for small part of a Web site, accessed by few users, but incase of hundreds of users the IIS security system becomes extremely difficult if not impossible to use.

One solution in IIS to tackle such a problem is to make one account of a restricted area of Web site and let all users share the same *username* and *password*, but in this case, ability to revoke access from only one user without informing all others is lost. Moreover, each user looks the same, therefore identity of individual user is lost.

1.1.3 Dependence On NTFS File Format

IIS depends on *NTFS file format* of NT to control access to individual directories, i.e. it cannot protect a directory if NT's file format is FAT. It also means that non-NTFS partitions cannot contain directories that are part of a protected Web site.

1.1.4 Limited Ways Of Protection

IIS only protects directories by *username, password and IP address*, but for general Web site usage there are many cases when there is requirement to authenticate pages in some other way, e.g. *by Referrer. T*hat is. on the basis of the previous page that the user has visited, this enables the removal of requirement to authenticate by Username if the next page lies in some other directory. Another way to authenticate *by Domain name.* That means, a user can be authenticated by domain name through which a user is approaching that Web site

1.1.5 NTLM Method

IIS and Internet Explorer also provide another method that is NTLM(NT Logging Mechanism or NT Challenger/Response). This method is integrated with Internet Explorer and does not prompt user for username and password and gets the required information from the underlying operating system. But this method works only if there is *NT or LAN Manager Networking Domain* available, which is mostly in Intranet; however, it is not useful for Internet use.

1.1.6 NT Access Rights Dependency

The system is able to protect directories only if these are protected under the Windows NT, other wise if the directories are listed in the WWW access service, i.e. they are visible to the Web browsers then there is no way of stopping even an anonymous user to access these.

1.2 Objectives of the Project

As discussed in section 1.1 there are many problems related to the default implementation of the authentication scheme in IIS. This makes it unsuitable for internet use when the server has Web site accessed by a large number of users. The main objective of this project is, to develop authentication software that works independent of the *IIS and NT* protection methods and can still protect the directories on the Web site. The specific objectives of the project are summarized in this section:-

In IIS authentication process is only dependent on NTFS. The project is aimed to make the authentication process possible for NTFS file format as well as for FAT file format. This will make it possible to protect directories in all partitions on the hard disk and in NTFS or FAT file format.

Often, a facility in Web sites is required to map the Internet users to internal NT account users so that local NT users may enjoy the same permissions even if they are logged on through the Internet. Hence one of the objective is to map users of the external database to the users of internal database of NT. This method will allow the web site users to access the directories even if they are protected under NT on the criteria of external database.

IIS protects directories by username, password and IP address only An effort will be made to authenticate the user in more than three. These protection methods may be e.g. *by referrer or by domain name etc*. Also the methods will be custom developed leaving room for increasing no of ways, the directory can be protected. This method enables the users to access directories in a better and easy way and reduce the requirement of entering authentication credentials many times. Once a user has entered a protected domain by supplying the credentials the rest of authentication under such conditions is done transparently.

Database hits are normally resource extensive and take much longer if simultaneous access to the database is made for the authentication purpose. One of the objectives is to make the

authentication process fast by *caching* the authenticated credentials in memory and reduce database hits

Logging of user activity as per custom requirements is also one of the most demanding requirements. Hence another objective is to create a *log* according to the custom authentication and covering those aspects of logging, which are not covered by the system log of NT.

1.3 Benefits of Proposed Objective

The proposed project will relieve the system manager from the extra burden of the *User Manger* tool of Windows NT and provide a separate interface to manage the protection methods. Development and maintenance of username and password will be independent of the Access Control List (ACL) of NT so the manager can easily handle and distinguish the local and remote users. It will remove extra load on NT database by keeping authentication credentials in a separate external database.

The project will improve the efficiency of NT operating system by keeping the Web site security separate and independent of NT operating system.

The package will also remove unnecessary requirement to protect the directories of the Web site on the host server. Thus remove the overhead and requirement of granting explicit access to all the existing users in NT. However, if directory is protected under NT then explicit permissions are required even for NT users in that local domain to access the protected directory.

1.4 Thesis Outline

First chapter has identified limitations of IIS and based on those limitations, objectives of this project and benefits of implementing these objectives have been highlighted

Second chapter will provide .conceptual background of client and server model and based on that, different phases of authentication process are discussed in detail. After that different authentication schemes currently available on Internet technology are briefly outlined and finally flow of HTTP information in IIS has been explained with a view to build the discussion for third chapter.

Chapter 2
Authentication Background

2.1 Introduction

In this chapter the concept of client–server model is discussed with emphasis on application of authentication in this model.. Then HTTP (Hypertext Transfer Protocols) and mechanism of challenge and response is described in detail. There are generally five steps involved in the process of challenge and response and authentication design needs to incorporate these basic steps. There are generally two types of authentication schemes namely Basic and NTLM. Finally their merits and demerits have been presented briefly

2.2 Clients and Servers

The Web is based on a client–server model. Client–server computing distributes the basic components, user interface, program logic and data between the client and server computers [2]. In client–server model, the client can request data from the server or can post data back to the server for storage. Client can also request a process to run on server and may provide any specific kind of functional logic.

In client-server model, the server can send data to the client or may provide access to data storage and like client may also provide functional logic.

In traditional client-server model, client and servers can be "*thin*" or "*fat*". The terms indicate a functional relationship rather than a physical characteristic of the computer. The standard components of an application are a user interface, the programmer's logic or business rules and data storage.

Client-server computing is a division of labour that is typically needed by most applications. The server is optimized to provide data to multiple clients and the client application is optimized to interact with the end user. The one task that may reside on one or both sides of the model is the functional logic.

Thin clients are generally limited to a user interface. All processing and program logic resides on a fat server. The server is called “fat” because the functional logic resides on the server. This is the most common model on the Web today [2].

The client here is typically a Web browser such as Microsoft’s Internet Explorer. The Web browser is simply a user interface to many kinds of data that are retrieved from the Web server. Here ISAPI application provides the functional logic to the HTTP server and the client displays various types of data from a specific URL.

2.3 Authentication in HTTP

Authentication is the process of obtaining and verifying the identity of a user. The authentication process is not responsible for determining whether or not the user has access to a particular resource, it is only responsible for establishing the user’s identity. However, authentication is the key to building an access control system for a Web site. After the server has authenticated a user’s identity, that identity might be used by the server to decide whether the user has right to access a given resource.

The HTTP standard defines a mechanism for user authentication. The process defined by the standard is simple, flexible and extensible. The protocol defines a challenge and response process for verifying user’s identity and right to access the system.

The process is very straightforward: It comprises of following sequence of events:

- An anonymous user requests a secured or restricted resource.
- The server responds with an HTTP 401 “Access Denied” message challenging the users right to access the resource.
- The browser displays a dialog box, prompting the user for a user name and password.
- The browser resubmits the original request to the server, this time including the credentials derived from the Id and password.

- The server both accepts the credentials and responds with the requested data or responds with another HTTP 401 “Access Denied”.

This sequence of events is discussed in more detail in this section.

2.3.1 The Initial Request

Generally people work with Web sites as anonymous user. The user remains in an anonymous state until a request is made to a protected resource. At this instance, the authentication process is initiated.

HTTP specification itself does not dictate the circumstances that initiate the process. HTTP standard only dictates how the server should indicate to the remote client that authentication is required.

E.g. the initial request might appear as something like shown in Table 2.1: -

Table 2.1-Sequence of initial request

Line No	Types of Requests
01	GET /bin/secure/sample.dll? HTTP/1.0
02	Accept : image/gif, image/ jpeg, */*
03	Referrer : http//localhost/Warehouse/home.html
04	Accept-Language : en
05	UA-pixels : 800*600
06	UA-color : color8
07	UA- OS : Windows NT
08	UA-CPU : *86
09	User-Agent : Mozilla/2.0 (compatible; MSIE 3.0; Windows NT)
10	Host : localhost
11	Connection: Keep-Alive

Now if the directory secure requested in line 01 is protected than this initial request will trigger the authentication process.

2.3.2 The Challenge

After the server has determined that authentication is required for access to the resource requested by the user, it generates a standard HTTP 401 “Access Denied” message.

This response indicates the remote browser that its request has been denied but may be resubmitted with authenticated credentials.

A sample response from a server to browser in this case is shown in Table 2.2: -

Table 2.2-Sequence of server **response**

Line No	Types Of Response
01	HTTP/1.0 401 Access Denied.
02	WWW-Authenticate : Basic realm = “local host”
03	Content-Length : 24
04	Content-Type : text/html
05	Error : Access is Denied

This message indicates that the request has been denied by HTTP 401 code in the message header. This means that the browser can again request the page but with complete credentials. It is different from HTTP 403 – state code which shows that the access is completely forbidden.

The access- denied message on line 05 is part of HTTP message body and can be used to indicate the reason of denial if the browser does not attempt to authenticate. The line 02 indicates the methods of authentication supported by the server.

2.3.3 Acting on the Challenge

After the browser receives the access denied message it has the option to obtain the credentials or display the access denied message. Credentials might be obtained by prompting the user for a user name and password or some other method. For example some browsers obtain the security credentials from the underlying OS and the current logged on user account on the LAN.

After the browser has obtained the required credentials it responds to the challenge of the server automatically.

2.3.4 Responding to the Challenge by Resubmitting the Request

After the browser has obtained some notion of user-id or password, it resubmits the original request with an additional peace of information: authentication credentials. These are supplied in request in the form of additional request header: for example the request described in Table-2.1 will be sent again with an additional 12^{th} line as:

12 Authorization: basic c2hrbWM6eHh4

HTTP is a stateless protocol so it needs to supply these credentials with a new request for the resource. The server does not keep track of what resources were specified in the original request.

2.3.5 Result

After the request is submitted to the server it is processed normally. The server finds the authentication header in the request and attempts to use these credentials to determine the users identity.

The credentials might be incorrect and in that case the server again responds with 401 “Access Denied” message that started the whole authentication process. The browser than has the option to retry authentication by allowing the user to reenter user name and password or shift to another authentication scheme. Many browsers allow only three attempts to authenticate for a given resource before displaying the access denied message to the user. If the credentials supplied with the new request are valid, the server returns the requested resource to the user.

2.3.6 Impact on Future Requests

After the browser has successfully completed this authentication process for a given server, it generally caches the credentials it used for the log on. It than silently adds these credentials to all requests to the authenticating servers for the duration of the users current

session. This behavior means that each individual request for a resource for the server after the initial request will include the credentials accepted in the original request.

2.4 Authentication Schemes

Authentication schemes are mechanisms, used to pass credentials from a browser to a server.

The browser and servers can innovate new authentication schemes and still negotiate to find a set of protocols supported by each.

The data governing the behaviour of authentication system is carried in a set of new message headers: The WWW-authentication records. A WWW-authentication record should consist of the following.

a. *The WWW-authentication variable name*.

b. *The name of authentication scheme*: examples are “Basic” or “NTLM”.

c. *A realm specification*: This consists of a string of the form realm = <realm name>. The realm name is used to define the space on a server in which a set of credential is valid. IIS generally sets a realm name to the host name, defining a single authentication space spanning the site.

d. *An optional list of authentication parameters*: This is a comma delimited list where each variable takes the form name = value. These values are dependent on the authentication scheme in use

The authentication schemes allow growth and innovation in the mechanisms used to pass credentials to the server. The HTTP protocol does define one scheme that should be implemented by most browsers, the basic authentication scheme. Web servers are free to implement other authentication schemes that improve upon this basic one.

2.4.1 The Basic Authentication Scheme

The basic scheme transmits encoded username and password across the network. A server indicates that it supports the basic authentication scheme by indicating a WWW-authentication record in the response header, as: - *WWW-authentication*: Basic realm = “localhost”.

A request carrying authentication credentials for the basic scheme will contain an authorization record, as: - Authorization: Basic c2hrbWM6eHh4

The authorization record contains the scheme to be used to check the credentials and the encoded form of the credentials themselves. In the HTTP basic scheme, this consists of the username and password (separated by a colon), encode in the MIME base-64 format.

This format consists of a set of 65 human readable ASCII characters. Each character of the encoded data string represents 6 bits of the data originally used to represent the input. Its goal is not to encrypt data but to transmit it in a format that can be safely reproduced by almost any machine.

Basic authentication is not a secure method of transmitting data, since MIME-64 encoding is only a method of encoding and not encrypting of data. This method can prevent a casual user from reading the user name and password from a network monitor but a technical person cans easily determine it.

There is a method to overcome this problem: combining the basic scheme with SSL (secure socket layer). Sessions running under SSL encrypt the data in the packet in a quite secure way. This method transforms the basic authentication into a quite secure mechanism. IIS and most commercial browsers support SSL.

2.4.2 The NTLM Authentication Scheme

The IIS and Internet Explorer implement an additional scheme called NTLM authentication. This scheme does not prompt the user to enter a username and password or transmit them across the network. Instead, it obtains security credentials of the currently logged on user from the underlying operating system.. The browser then transmits a

synthesized value across the network to the server, representing the user's security credentials.

This authentication scheme works only if an NT or LAN Manager networking domain is available. This makes it ideal for use in the settings where the underlying network is based on Microsoft server of some kind. It is not useful in Internet applications.

2.5 Authentication and Windows NT Security

IIS integrates the authentication with Windows NT. In Windows NT, authentication is basically a process of resolving incoming WWW username to NT logons. In the default IIS authentication process, the Windows NT user pool is used to authenticate Web users. The pool of username available for authentication in IIS is the set of users defined in the '*NT User Manager*' and the user's Windows NT password are used to validate their rights to use the identity.

A subject of some confusion is to determine that, which NT security database will be used to authenticate a particular request when the server is member of Windows NT domain. If the Web server is installed on a primary or backup domain controller, the domain's security database will be used as the default source for username and password. If the Web server is hosted on a server that is not a domain controller, the server's local security database will be used by the Web service when authenticating users.

In authentication process the server processes the information flow in seven steps, as shown graphically in Fig 2.1: -

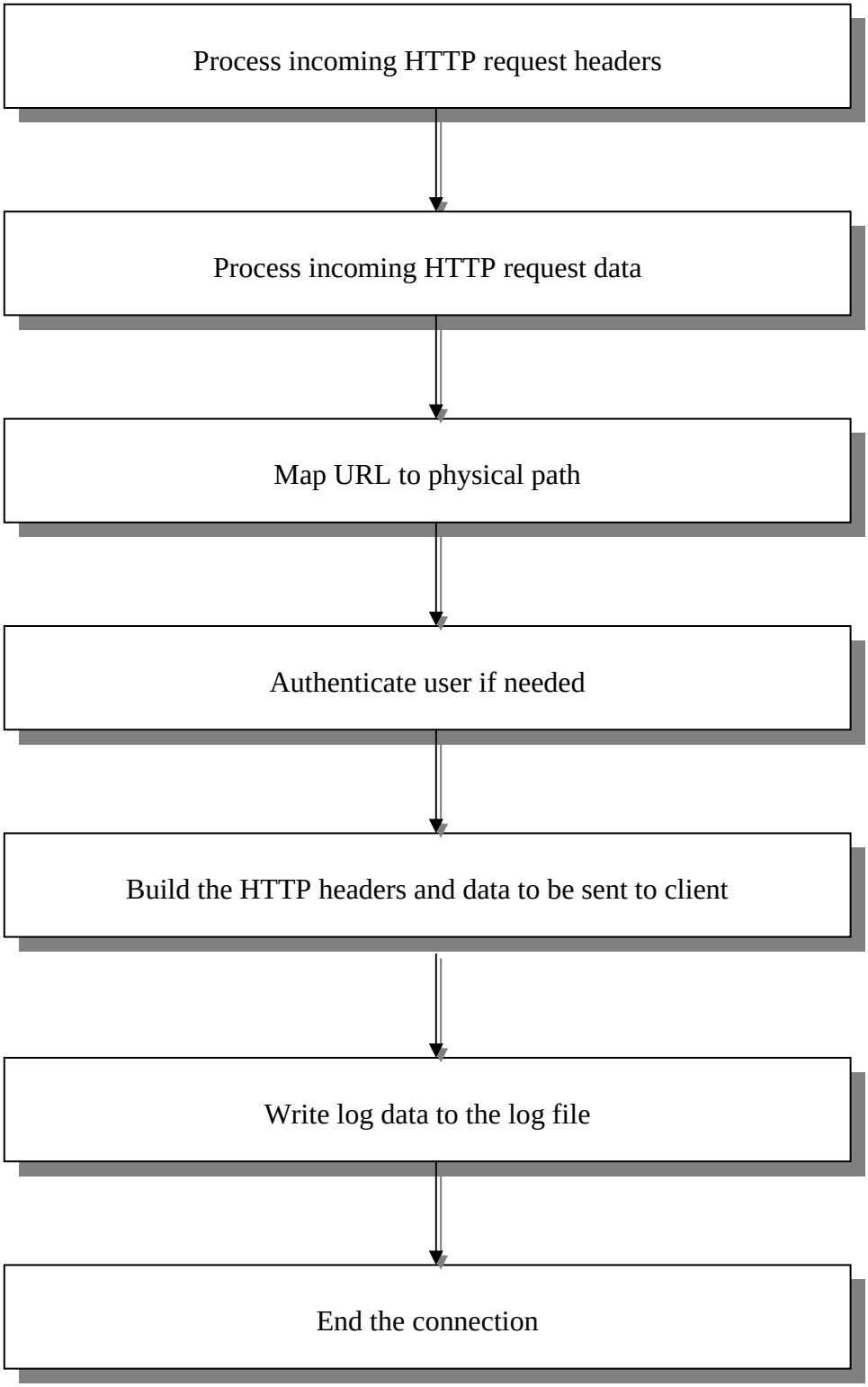

Fig 2.1-*Process of information from a client by a server*

2.5.1 The Anonymous User

When configuring Windows NT it is possible to allow anonymous users to access the resources on our Web site. In such case the Windows NT assigns a password to the anonymous user account that is used by the system to grant access to the resources allowed to the anonymous user. [3]

By default on installation, NT creates an anonymous account by the name of IUSER_ <machine name>. The system administrator can change this username and password at any time.

2.5.2 NT Access Rights and Authentication

Every request is resolved by using the context of some Windows NT user. For anonymous users, this is the Windows NT user associated with the WWW user during the authentication process. This is true for files and execution of code

The IIS security system is managed by using the standard Windows NT access rights. A user's right to access a resource is determined by checking the rights of the users engaged in requesting resource. If the NT user has access to the requested resource in NT, it will generally have access to it on the Web; if it does not, the server will not allow access.

The anonymous user is treated differently. If the anonymous user requests a protected resource, it triggers the authentication process and the browser pops up the authentication dialog to obtain the required credentials as explained in section 2.3.

2.6 Requirement for Custom Authentication System on IIS

The existing IIS authentication system works efficiently as long as the number of potential users is less but with increase in the number of users it become more and more difficult to handle the mechanism.

Taking into consideration the limitations of '*IIS default authentication system'* it seems appropriate to develop a custom authentication system that can remove the limitations of the 'IIS authentication system' and make the life of Web site administrator easier.

By the custom authentication scheme, limitations and drawbacks of the IIS authentication system can be removed and Web site account can be kept external to the NT user accounts. Also a room for advancement in the methods to authenticate can always be made as per the specific requirements of any user managing there own Web site. .

2.7 Summary

The Web is based on a *client–server* model. Client–server computing distributes the basic components, user interface, program logic and data between the client and server computers

The HTTP standard defines a mechanism for user authentication. The process defined by the standard is simple, flexible and extensible. The protocol defines a *challenge and response* process for verifying user's identity and right to access the system. This process comprises of a number of sequences of events, discussed in detail in this chapter.

Authentication schemes are mechanisms, used to pass credentials from a browser to a server. Two methods have been discussed namely *basic authentication* and *NTLM authentication*. *Basic authentication* has a limitation that user credentials can be hacked easily by some experience users while on there way over communication channels and therefore SSL (Secure Socket List) is used in conjunction with *basic authentication*. However, NTLM is safe but can only work among Microsoft products.

Chapter 3
Goal Achieving Mechanisms

3.1 Introduction

Currently three alternatives namely CGI, ASP and ISAPI are available for creating custom designed Web applications on server side. These three technologies are briefly discussed in this chapter. Moreover, reasons for selecting one of the methods will be outlined and finally detailed architectural background of selected method will be explained with a view to discuss its implementation in this project.

3.2 Methods Available for Achieving Objectives

The Windows NT provides a platform for building Web applications for the Internet and Intranet environments.. It can be delivered as combination of Web pages that provide the user interface to the application and ActiveX components that encapsulate business logic and provide access to the databases where critical business information is stored. Currently three alternatives namely Common Gateway Interface (CGI), Active Server Pages (ASP) and Internet Server Application Programming Interface (ISAPI) are available for creating custom designed web applications on server side. These three technologies are briefly discussed in this section.

3.2.1 Common Gateway Interface (CGI)

CGI is a way WWW servers on most operating systems (such as UNIX and the Mac OS) are extended to support dynamically created Web pages. CGI is less secure as well as slower than both ISAPI applications and ASPs (Active Server Pages), but they can be easier to develop for individuals who are already familiar with CGI programming methods. CGI executables run in address space separate from the IIS program. CGI executables can be written in any language that produces Windows .exe files. The Windows CGI environment (WIN-CGI) is also very much like the UNIX CGI environment, but the execution environment is not identical. CGI executables are often program interpreters, which means that they read in and execute a scripting language such as Perl, TCL or BASIC. The term

CGI script describes a text containing commands that are interpreted by such a CGI executable.

3.2.2 Internet Server Application Programming Interface (ISAPI)

ISAPI applications are very much like CGI executables except for two things. ISAPI applications can run in the same address space as IIS or in a separate memory space and can therefore be faster than CGI executables, and they are written to a specialized interface on a compiled programming language such as C++. ISAPI applications like CGI scripts can perform complex operations on their own or can be interpreters for scripting languages such as Perl and TCL. ISAPI can also be used to implement *filters,* which extend the functionality of IIS and can affect how all communications streams in and out of IIS.

3.2.3 Active Server Pages (ASP)

ASPs are HTML pages that contain scripts written in *VB script* or *J scripts,* as well as regular HTML text. ASPs utilize an ISAPI DLL (asp.dll) as a scripting host to implement the *VB script* and *J script* scripting languages on the server side. These scripts are used to create HTML files before the HTML is sent to the browser. ASP scripts are used to provide many of the HTML text generation functions that you can do with CGI executables and ISAPI applications. In fact, ASPs are implemented as ISAPI applications. Moreover ASP provides any easy interface for programming but it loses execution speed and depth to handle many complex situations.

3.3 ISAPI-The Selected Method

Tradition method to develop Sever side applications, [4] is the use of Common Gateway Interface (**CGI)** because of its natural development environment dependency on time tested operating of UNIX. By keeping in view the limitations of IIS, Microsoft® developed the ISAPI model as an alternative to the CGI. The ISAPI model provides a number of advantages over the CGI model, including *low overhead, fast loading* and *better scalability*.

The two general types of ISAPI DLLs namely *ISAPI extensions* and *ISAPI filters* can be developed in IIS with the help of any system language like Visual C++. *ISAPI extensions*

can be assembled into Web applications that provide all of the functionality of an application running on client's desktop. *ISAPI filters* are created to customize the functionality of IIS. ISAPI *filters* provide a slightly different processing model than *extensions*. A client activates an extension upon request, whereas a *filter* is activated by IIS directly when a particular request event occurs. Extensions are *request-driven* and filters are *processing event-driven*.

3.4 Comparison of ISAPI Vs Traditional CGI

The chief difference [5] between the CGI programming model and the ISAPI programming model is that CGI creates a unique process for every request, while ISAPI does not. With CGI, every time an HTTP server receives a request it must initiate a new process, which, along with maintaining processes, is very resource intensive. This inherent limitation in CGI has made it difficult to develop responsive applications on the Internet. Fig 3.1 Illustrates differences between the CGI model and the ISAPI model: -

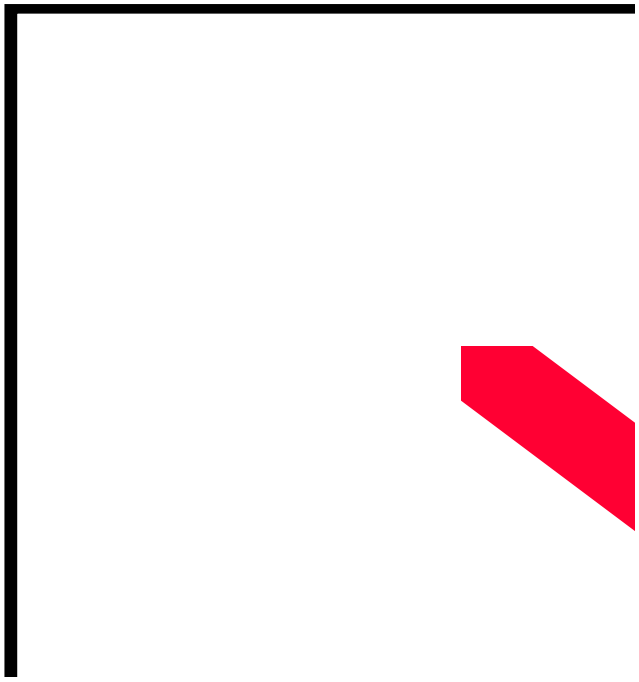

Fig 3.1-ISAPI vs. CGI model

In the ISAPI model, each request received by an HTTP server initiates the creation of an **EXTENSION_CONTROL_BLOCK** (ECB) data structure. Creating and maintaining a data structure is much easier and faster than initiating a new process. In addition, since the ECB and the extension are usually both running in the same process as IIS, the server can process requests faster and accommodate a higher volume of requests.

Finally, rather than using process isolation, the ISAPI model uses threads to isolate processing work items. Using multiple threads to synchronize work allows IIS to make more efficient use of system resources than is possible with the CGI model, or other models based on process isolation.

IIS 4.0 supports process isolation for ISAPI DLLs and scripts. IIS uses custom high-speed methods to establish communication between the server process and the surrogate process housing, ISAPI DLLs thus provide robustness with high performance

3.5 ISAPI Filter Architecture[5]

ISAPI filters are DLLs that are loaded into the IIS process when the Web service is started, and stay in memory until the Web service shuts down. Once loaded, ISAPI filters can be configured to receive a number of special filter event notifications that occur with each HTTP request that IIS receives, and each response that IIS generates in return.

When an ISAPI filter is loaded, the filter passes a structure to IIS that contains, in addition to other information, a bit field that specifies for which types of filter event notifications the filter should be notified. Each time one of those events occurs, an event notification is sent, and every ISAPI filter that has specified interest in that event is notified. The Fig 3.2 depicts the ISAPI filter architecture:

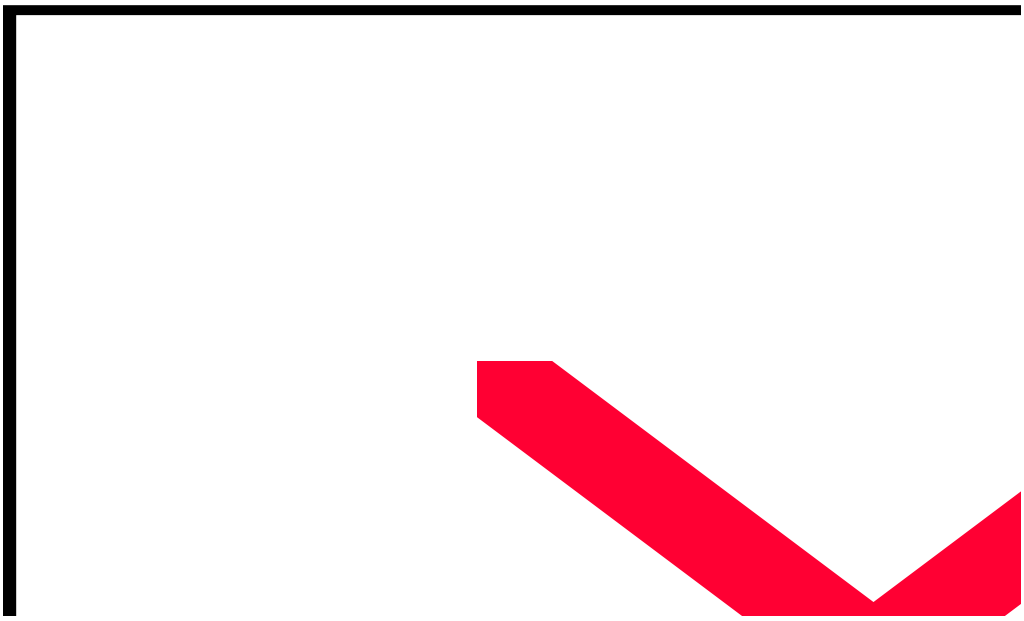

Fig 3.2- ISAPI filter architecture

3.5.1 Uses of Filters

ISAPI filters are very powerful and can be used to facilitate applications that perform a number of different tasks, including:

- Custom authentication schemes.
- Compression.
- Encryption.
- Logging.
- Traffic and other such like request analysis.

The ability to examine, and if necessary modify, both incoming and outgoing streams of data makes ISAPI filters very powerful and flexible.

3.5.2 Filter Event Processing

Every filter is always contained in a separate DLL that must export two entry-point functions, *GetFilterVersion* and *HttpFilterProc* and optionally export the *TerminateFilter* exit function.

A metabase property, *FilterLoadOrder*, contains a list of all filters that should be loaded by IIS when the Web service is started. Table .3.1 further highlights the phases of filter processing: -

Table 3.1 Phases in filter processing

Phase	What Happens
Registration	IIS loads the filter and calls the *GetFilterVersion* () entry.
Event processing	IIS calls the filter's *HttpFilterProc* () entry as it processes HTTP requests.

3.5.2.1 Phases of Processing [6]

This section describes the mechanism of two phases as summarized in Table 3.1 in greater depth.

3.5.2.1.1 Phase I-The GetFilterVersion: -

When the server starts up, it checks a special registry entry to find out which DLL it should load as filters. As each DLL is loaded, the special–purpose *GetFilterVersion ()* entry is called. *GetFilterVersion* () does the registration phase of the filter. It serves the following purpose: -

a. *It specifies the filter's version information.* Using the HTTP_ FILTER_VERSION structure, GetFilterVersion returns a string to IIS with a version of ISAPI that the filter conforms to and a short description of the extension. This ensures that the filter and IIS are compatible. It also supplies descriptive information to identify a particular filter in the Web server or operating system (OS) event logs.

b. *It specifies the filter's priority.* Each filter must show its priority. The priority sets the order in which the filter is called to handle the events that apply to it. IIS may have to call many filters, so this is how the filter assigns the processing importance of any applicable events. Table 3.2 shows four priority levels. They are listed according to the priority by which IIS calls them.

Table 3.2-The four filter priorities

Priority	**What It Means**
SF_NOTIFY_ORDER_HIGH	Will load the filter at a HIGH priority.
SF_NOTIFY_ORDER_MEDIUM	Will load the filter at MEDIUM priority
SF_NOTIFY_ORDER_DEFAULT	Will load the filter at the DEFAULT priority. This is recommended.
SF_NOTIFY_ORDER_LOW	Will load the filter at LOW priority.

c. *It tells the server what events it will process*. The filter has the option of processing a defined set of events. It must tell the server which event it will process. This saves on the overhead needed to call each filter for every request that comes in. When GetFilterVersion () executes, IIS knows which events the filter will process and only calls it for one of those events. Section 3.5.3.3 explains in detail the sequence of notification events.

3.5.2.1.2 Phase II –The HttpFilterProc () Entry

Once the filter is loaded and registered, it is ready to get notifications. This is done when the server calls the HttpFilterProc () entry in the filter. Once again, this entry must be entered exactly as IIS expects it. This is shown in Fig 3.3.

```
DWORD WINAPI HttpFilterProc (PHTTP_FILTER_CONTEXT pfc,
                             DWORD notificationType, LPVOID pvNotification);
```

Fig 3.3-Declaring HttpFilterProc ()

The first parameter *pfc*, is a pointer to an HTTP_FILTER_CONTEXT structure. This structure holds information about the HTTP request itself. Appendix A explains the definition of this structure.

The second parameter *notificationType*, is a DWORD that represents one of the notification events the filter registered with the server. Section 3.5.3.3 explains possible values of this parameter.

The third parameter, *pvNotification*, is a void pointer to a server-supplied data area. This data area is the vehicle by which IIS shares the data to be processed with the filter. It is different for each notification type.

Table 3.3 summarizes the data type of this parameter for each notification type. Appendix B defines the structures *pvNotification* points to: -

Table 3.3-pvNotification: Events and Data Types

When the Notification Type Is	**pvNotification Points to a Structure of This Type**
SF_NOTIFY_READ_RAW_DATA	HTTP_FILTER_RAW_DATA
SF_NOTIFY_SEND_RAW_DATA	HTTP_FILTER_RAW_DATA
SF_NOTIFY_PREPROC_HEADERS	HTTP_FILTER_PREPROC_HEADERS

SF_NOTIFY_AUTHENTICATION	HTTP_FILTER_AUTHENT
SF_NOTIFY_URL_MAP	HTTP_FILTER_URL_MAP
SF_NOTIFY_LOG	HTTP_FILTER_LOG
SF_NOTIFY_ACCESS_DENIED	HTTP_FILTER_ACCESS_DENIED

The declaration of HttpFilterProc () specifies a return type of DWORD. *HttpFilterProc ()* is allowed to return a predefined set of values. IIS uses these values to determine how to continue processing the event. Table 3.4 shows these possible values and when they should be used.

Table 3.4-Return Codes from the HttpFilterProc ()

Type of Return Code	**When To Use It**
SF_STATUS_REQ_FINISHED	Use if the filter has handled the HTTP request. This tells the server to disconnect the session.
SF_STATUS_REQ_FINNISHED_ KEEP_CONN	Use if the filter has handled the HTTP request and TCP session is required to keep open if the option was negotiated.
SF_STATUS_REQ_NEXT_NOTIF ICATION	Use if it is desired to call next filter in the notification chain by the server
SF_STATUS_REQ_HANDLED_N OTIFICATION	Use this return code if the filter has handled the notification and no other filter should be called for notification.
SF_STATUS_REQ_ERROR	Use to tell the server that an error has occurred. The server will call *GetLastError ()* and indicate the error to the client.

SF_STATUS_REQ_READ_NEXT	Used for raw-read notification only. It is used when the filter is an opaque stream filter and the session parameters are being negotiated.

3.5.2.2 Details of Request Processing Sequence

The following steps outline how IIS and ISAPI filters generally interact with each other during process of the request:

a. When IIS initially loads an ISAPI filter, IIS creates and partially populates a *HTTP_FILTER_VERSION* structure. IIS then calls the filter's *GetFilterVersion* function, passing a pointer to the new structure as a parameter.

b. The ISAPI filter populates the *HTTP_FILTER_VERSION* structure with some version and descriptive information. More importantly, the filter also uses HTTP_FILTER_VERSION to specify which event notifications it should receive, and to declare the general priority level for the filter. In addition, the filter also indicates whether it is interested in events from secure ports only, unsecured ports only or both.

c. Each HTTP transaction between IIS and a client browser triggers several distinct events. Each time an event occurs for which an ISAPI filter has registered (by using *HTTP_FILTER_VERSION*, as described in step b), IIS calls the filter's *HttpFilterProc ()* entry-point function.

d. If more than one ISAPI filter has registered for a given event, then IIS will notify filters marked as high priority first, medium priority second, and low priority last. If more than one ISAPI filter has declared the same general priority level, IIS will use the order in which the filters appear in the FilterLoadOrder property to resolve the tie.

e. The ISAPI filter uses the notification type information, passed by IIS as a parameter to *HttpFilterProc*, to determine what particular data structure is pointed to by the

other *HttpFilterProc* parameter, *pvNotification*. The ISAPI filter can then use the data contained in that data structure, as well as in the context structure HTTP_FILTER_CONTEXT, to perform any custom processing.

f. Once processing is complete, the filter returns one of the SF_STATUS status codes to IIS and IIS continues processing the HTTP request or response—at least until another event occurs for which ISAPI filters have registered.

g. When the Web service is stopped or unloaded, IIS will call *TerminateFilter* () in all ISAPI filters, as part of its shutdown sequence, for any filters that have implemented and exported the function. *TerminateFilter* () is typically used to perform cleanup and de-allocation of allocated resources.

During request processing, *GetFilterVersion* is called exactly once, when the ISAPI filter is initially loaded. If it is required to perform some per-connection initialization, one has to manage it internally within the context of the *HttpFilterProc* function call. Moreover, the priority setting for ISAPI filters is per filter, not per notification. For example it is not possible to assign a low priority rating for one type of notification, and a high priority rating for another type.

3.5.2.3 Typical Event Notification Sequence

In general, the events that occur during the processing of a typical IIS request and response are regular and predictable. The following steps outline the most common ordering of events:

a. **SF_NOTIFY_READ_RAW_DATA:** When a client sends a request, IIS will send SF_NOTIFY_READ_RAW_DATA, initially, only once. Data will be read until the client has sent the entire HTTP headers associated with the request. If there is more data available from the client (such as in a POST operation), it will not be read until step f.

b. **SF_NOTIFY_PREPROC_HEADERS**: A single SF_NOTIFY_PREPROC_ HEADERS notification will occur for each request. This notification indicates that

the server has completed pre-processing of the headers associated with the request, but has not yet begun to process the information contained within the headers.

c. **SF_NOTIFY_URL_MAP**: An SF_NOTIFY_URL_MAP notification will occur after the server has converted the virtual URL path contained in the HTTP request into a physical path on the server. Note that this event may occur several times for the same request.

d. **SF_NOTIFY_AUTHENTICATION**: An SF_NOTIFY_AUTHENTICATION notification occurs just before IIS attempts to authenticate the client. This notification will occur only on the first request made in a particular session, if Keep-Alives are actively being used by both server and client. If Keep-Alives are not in use, each request will cause IIS to send notification of this event.

e. **SF_NOTIFY_ACCESS_DENIED**: This event occurs if IIS has denied the client access to the requested resource (returning an HTTP 401 status code).

f. **SF_NOTIFY_READ_RAW_DATA**: As mentioned in step a, if the client has more data to send one or more SF_NOTIFY_READ_RAW_DATA notifications will occur here. Each read event notification will indicate that IIS has read another chunk equal in size to either the value of the *UploadReadAheadSize* metabase property (in kilobytes; usually 48 KB); or the remaining number of bytes available, if on the last chunk. Additional raw read events are not always completely predictable, because many factors can force IIS to adopt a different chunking scheme. Therefore, ISAPI filter should not rely on the exact behaviour described here. At this point in the request, IIS will begin to process the substance of the request. This may be done by an ISAPI extension, a CGI application, a script engine (such as ASP, PERL, and so on), or by IIS itself for static files.

g. **SF_NOTIFY_SEND_RESPONSE**: The SF_NOTIFY_SEND_RESPONSE event occurs after the request is processed and before headers are sent back to the client.

h. **SF_NOTIFY_SEND_RAW_DATA**: As the request handler returns data to the client, one or more SF_NOTIFY_SEND_RAW_DATA notifications will occur.

i. **SF_NOTIFY_END_OF_REQUEST**: At the end of each request, the SF_NOTIFY_END_OF_REQUEST notification occurs.

j. **SF_NOTIFY_LOG**: After the HTTP request has been completed, the SF_NOTIFY_LOG notification occurs just before IIS writes the request to the IIS log.

k. **SF_NOTIFY_END_OF_NET_SESSION**: When the connection between the client and server is closed, the SF_NOTIFY_END_OF_NET_SESSION notification occurs. If a Keep-Alive has been negotiated, it is possible that many HTTP requests occur before this notification occurs.

3.5.2.4 Exceptions to the Event Sequence

The sequence in section 3.5.2.3 is accurate enough for most purposes. However there are often such factors that can change the order of events. IIS or ISAPI filters may interrupt or modify the sequence of event notifications.

Due to the complex and dynamic event model used by IIS, it is important not to rely on the exact event notification sequence described in section 3.5.2.3. A well-designed ISAPI filter must be prepared to deal with events that occur in a non-standard order, without failing and, more importantly, without causing IIS to fail. Following are some of those situations in which the order or context of an event notification is nonstandard:

a. If ISAPI filter returns SF_STATUS_REQ_FINISHED from an SF_NOTIFY_ PREPROC_HEADERS handler, SF_NOTIFY_URL_MAP will not be called for that request, although some subsequent notifications, such as SF_NOTIFY_ END_OF_NET_SESSION, will still be called.

b. If access has been denied the client for the request resource, the next event after SF_NOTIFY_ACCESS_DENIED will be SF_NOTIFY_END_OF_REQUEST.

c. The SF_NOTIFY_URL_MAP event notification can occur multiple times on a single request, occasionally with an empty string for the *pszURL* member. One known cause

of this is a call to *ServerSupportFunction ()* with HSE_REQ_ MAP_URL_TO_PATH.

d. If integrated Windows authentication (also called NTLM authentication) is being negotiated, there are typically three requests and responses between the client and server. The SF_NOTIFY_END_OF_REQUEST notification will not occur on the second request when IIS responds to the client with a challenge. All other notifications occur as expected on this second request and response. For this reason, one should not rely on this notification as a signal to free any memory allocated elsewhere in the filter unless it is known that integrated Windows authentication is not being used.

e. SF_REQ_DISABLE_NOTIFICATION of ServerSupportFunction can be used to disable all further notifications, for the duration of the current request, if the filter has completed all processing for the request.

f. Priority ratings for filters (high, medium, and low) are temporarily reversed for any filters that participate in the SF_NOTIFY_SEND_RAW_DATA event notifications, for the duration of the notification. This reversal allows higher priority filters, such as encryption filters, to process the data after other raw data filters have operated on the unprocessed data, and just before the data is sent to the client browser. The priority ratings are unchanged for SF_NOTIFY_ READ_RAW_DATA event notifications.

3.6 Summary

Currently three alternatives namely CGI, ASP and ISAPI are available for creating custom designed dynamic Web applications on server side. CGI executables run in address space separate from the IIS program, contrarily, ISAPI applications can run in the same address space as IIS or in a separate memory space. Therefore ISAPI applications are faster than CGI executables, but they are difficult to program because these are written to a specialized interface on a compiled programming language such as C++. ASPs are HTML pages that contain scripts written in VBScript or JScripts, as well as regular HTML text. It is even more critical to keep Web site secure while running ASPs.

ISAPI is the selected method for achieving the objectives. Because of its importance in the thesis work a substantial amount of discussion has been made later in this chapter on important issues like ISAPI architecture and ISAPI event processing. Appendix B has also been attached to consult as reference about important data structures of ISAPI filter.

Chapter 4
Design Issues

4.1 Introduction

This project basically maintains a database that keeps all the information about users, groups to which users can be assigned and directories those are registered as protected. All sort of event logging information is also being maintained by this database. Figure 4.1 graphically explains the conceptual working of this project.

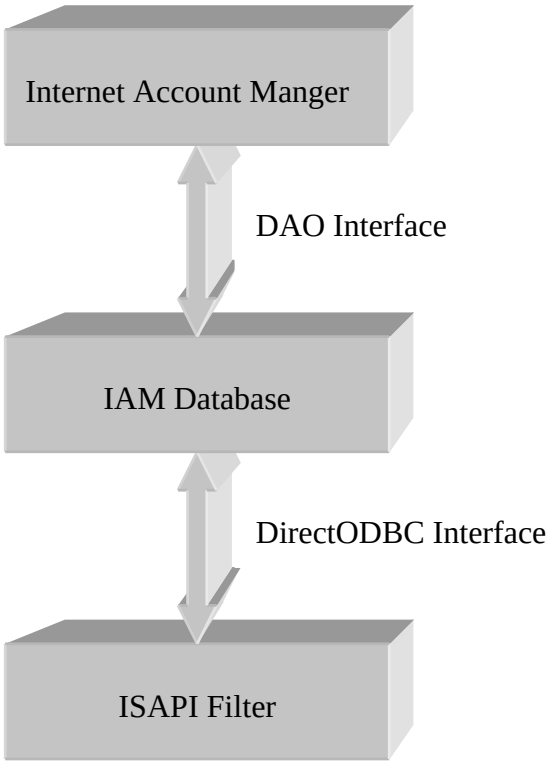

Fig 4.1-Conceptual Model

This chapter explains the database design that has been used as per specific need of the project. Then, some of design parameters of Internet Account Manager are discussed. Finally, some of the important design considerations of ISAPI filter are explained in this chapter.

4.2 Database Design

For designing such a database that can be interfaced with Visual C++, factors like choice of database management system (DBMS) and interface engine between DBMS and Visual C++ are of prime importance.

As far as regarding choice of the database, Microsoft Access 97 has been chosen for the reasons that it a relational database management system (RDBMS) and any RDBMS carries the advantage [8] that basic structure of relational model is simple, making it easy to understand on an intuitive level. Data operations are also easy to express and do not require that users be familiar with the storage structure used. The model uses few very powerful commands to accomplish data manipulations that range from simple to very complex. Hence for large databases RDBMS is the ideal choice.

As far as regarding choice of the interface engine, Microsoft Jet Database Engine (MJDE) is being used. This engine supports two basic techniques Data Access Objects (DAO) and Open Database Connectivity (ODBC). IAM is specifically using DAO for handling the database. MJDE will be discussed in detail in this section.

One of the important factors of a good relational database design is to have such a model that should have minimum redundancy in the database tables. Besides that, it is also ensured that the design is free of certain *update*, *insertion* and *deletion* anomalies [8] and for doing so process of database normalization is performed in systematic way.

4.2.1 Database Normalization

The process of transforming existing data into relational form is called normalization. Normalization of data is based on the assumption that the data has been organized into a tabular structure wherein the tables contain only a single entity class [9]. The objectives of normalization of data include the following:

a. Eliminating duplicated information contained in tables

b. Accommodating future changes to the structure of tables

c. Minimizing the impact of changes to database structure on the front-end applications that process the data

4.2.2 Database Design For Authentication Filter

After carrying out the process of normalization the final database design for authentication filter has emerged as a relation of seven tables connected with each other with *one to many* relations. Database design along with different relations among the tables is graphically shown in Fig 4.2.

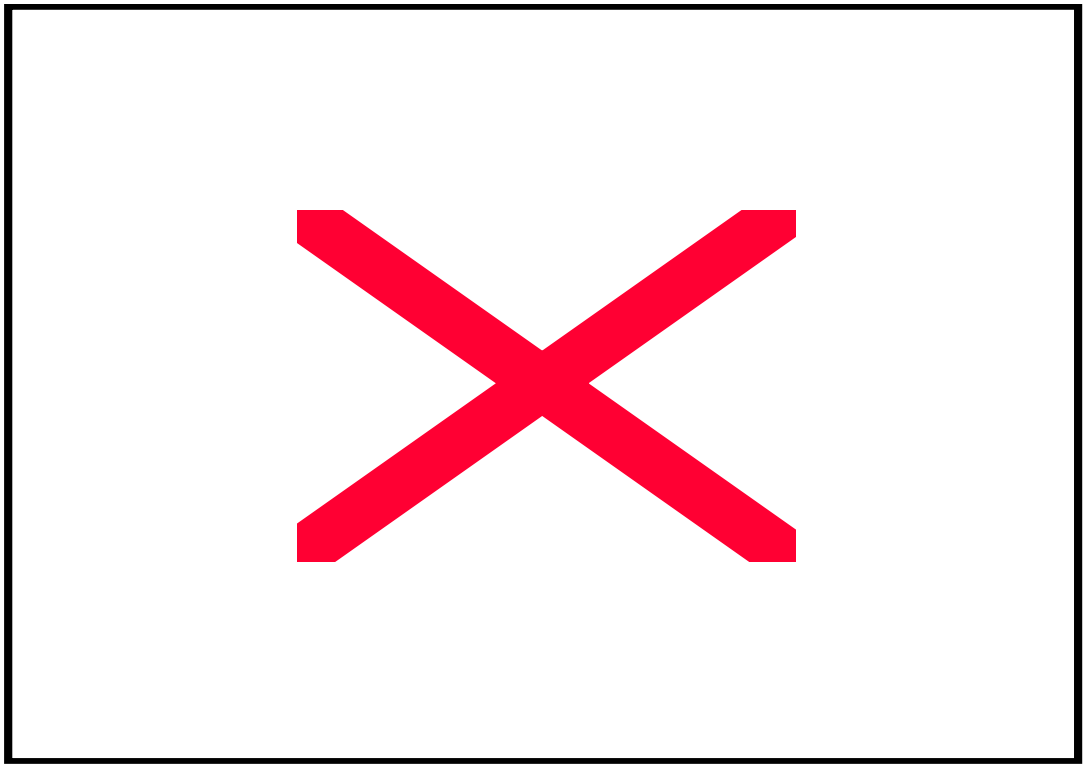

Fig 4.2 - Database Design for authentication filter

Purpose of each of the seven tables is described briefly in table 4.1.

Table 4.1 - Database tables for authentication filter

Table Name	**Purpose**
1. DirAccess	Keeps record of all the directories registered as protected directories for which user must be authenticated in order to access its resources. It also keeps the information about the way, directory will be protected, i.e. by internal database (not ACL), by referrer, by IP, by mapped user of NT access control list (ACL.).
2. User	Keeps all information about the created user with user password, expiry date of the account and remarks about the user account.
3. Group	Keeps all information about the group created by the administrator.
4. InternalDb	This table keeps the information about all such groups or users that have been provided access to any directory.
5. IP	Keeps the information about all such client's IP addresses that have been assigned to any particular directory and also whether the IP address is permitted or not permitted for this directory.
6. Referrer	Keeps the record of all permitted referrers for particular directories
7. MapToNT	Keeps the record of all the internal users those have been creates by *User Manager* of NT administrative tools and NT uses this username and password to allow access to any particular NTFS directory.

4.2.3 Database Design For Logging Filter

For logging filter two relations are incorporated to achieve the objective namely *Logging* table and *Flags* table. Fig 4.2 graphically represents the two tables used for logging filter.

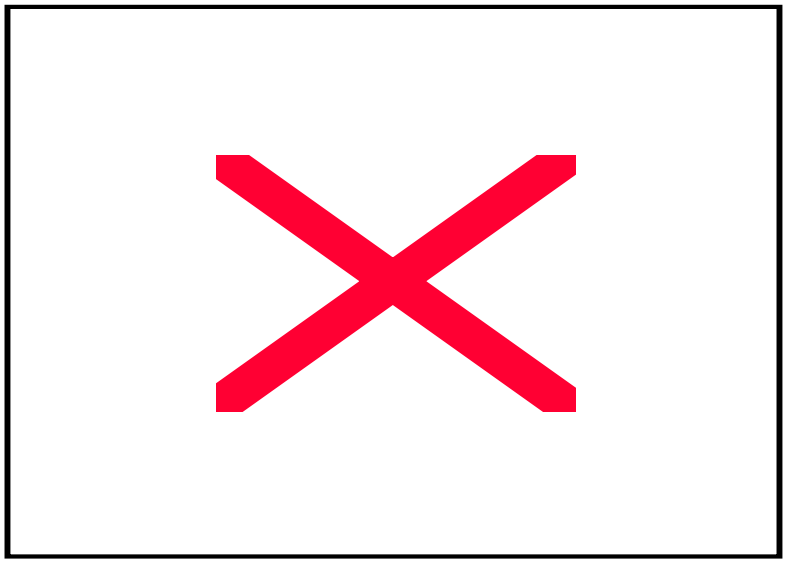

Fig 4.3 - Database design for logging filter

Purpose of each of the two tables is described briefly in table 4.2.

Table 4.2 - Database tables for logging filter

Table Name	Purpose
8. Logging	Keeps record of logging information like date/time, resource requested, remote IP address, remote username, referring Web server, client browser type (user agent) and status of the connection on client request.
9. Flags	Keeps all necessary flags to save the logging view environment for later use. It also keeps the date on which log was cleared. ISAPI filter uses the *enable logging* flag to enable or disable logging from this table.

4.3 Design of Internet Account Manager

Internet Account Manger (IAM) is basically a database front-end developed in Visual C++ 6. The aim is to develop such a front-end that should be user friendly and should provide powerful database handling environment with capabilities to manage the requirements discussed in following sub sections: -

4.3.1 .User Accounts

a. Administrator must be able to add, delete or change any user's credentials.

b. User's credentials should include user name, user password, expiry date and remarks for specific user account.

c. User name must be unique in a database and moreover, blank password must not be allowed.

d. A user friendly calendar must be provided to add or change expiry date, moreover user must not be able to set an expiry date earlier then the current system date.

4.3.2 Group Accounts

a. Administrator must be able to add, delete or change any group accounts.

b. Administrator should be able to add user to specific group at any time, even at the creation of the group.

c. If a protected directory is given access to a specific group then all such users those are member of that group must have access to that directory.

4.3.3 Directory Registration

a. Administrator must be able to add, delete or change any directory that should be kept protected. IIS will authenticate the client computer for such *protection methods* that are set for that particular directory

b. *Protection methods* that can be enabled or disabled for any directory during authentication are *by Internal Database, by IP address, by referrer Web site* and *by Mapping of directory to NT user*.

4.3.4 Protection Methods

Any directory that has been registered by IIS can be protected by various methods. These methods are custom developed and can be further enhanced according to specific needs of the user. Following are the methods those are implemented in this project.

a. **By Internal Database**: -A separate interface provides a method to assign username or groups (from IAM database) authorized to access a particular directory.

b. **BY IP Address**: - A separate interface provides a method to set such client's IP addresses like (1.1.1.127) those are permitted or not permitted for a particular directory.

c. **By Referrer**: - A separate interface provides a method to set such referrer Web sites like (http://cnn.com) as permitted referrers for a particular directory.

d. **By Map to NT**: - The directory being protected is also protected in Windows NT operating system under NTFS file format. Specific users of Windows NT have access to a particular directory. These rights are provided by *User Manger* utility of *Administrative Tools.* An Internet user that has been registered by IAM is not internal user of NT, hence that registered user cannot access any directory under NTFS file format. By mapping the directory to internal user of NT makes it possible for the system to provide access of the directory to Internet user who is not internal user of NT. Therefore, a dedicated interface is provided to map an NTFS directory to internal user of NT.

4.3.5 Advance Logging

The administrative front end also requires a method to choose among the custom logging options. These *logging options* are those, which are specifically required with custom authentication scheme and are not covered by the default logging of NT.

4.3.5.1 Logging Options

The administrator must have the control to enable or disable logging. Custom logging slows down the IIS process of supplying the browser with requested documents, so if logging is not required it should be disabled. Custom logging options are namely *unmapped remote user, referrer, and user agent*.

a. **Unmapped Remote User**: - This is to log the user who requested the protected document. User name is the one that was provided in original request of the client before any alteration by the authentication filter.

b. **Referrer**: - This is the URL of the page from which a link refers a page in the protected directory.

c. **User Agent**: - The type and version of the browser used to make the request is stored in this logging. On the information based on user agent it is possible to program the server to launch specific scripting programs for specific browsers.

These logging option s can be enabled or disabled. In addition to these logging options, credentials like *resource requested, date / time* and *remote IP* are also displayed log view.

4.3.5.2 Clearing Log

As log size grows very quickly so a mechanism is provided to clear the log. Two methods namely *automatic log clearing* and *manual log clearing* are designed.

a. **Automatic Log Clearing**: - Administrator has the choice to enable log clearing with intervals namely *daily, after three day* and *weekly*

b. **Manual Log Clearing: -** It is also possible to clear the log manually if the present log is not required.

4.4 Design Considerations for Developing ISAPI Filters

The MFC library provides an object model, as well as a wizard that aids in creating ISAPI filters. Classes such as *CHttpFilter* and *CHttpFilterContext* are provided for ease of programming and many intricacies are hidden by these classes. There are some important choices that should be considered while designing the structure of ISAPI filters.

4.4.1 Choice of Notifications [5]

All notifications must be handled with extreme care. Following consideration must be given thought while making selection:

a. ***Do not register for a notification unless it is needed*:** Filters should be registered for notification of only those events that are needs for processing. Some filter notifications are very expensive in terms of CPU resources and I/O throughput (most notably SEND_RAW_DATA), and can have a significant effect on the speed and scalability of IIS.

b. ***Register for SF_NOTIFY_END_OF_NET_SESSION***: Most of the filters should be notified for SF_NOTIFY_END_OF_NET_SESSION, as it is a good time to perform session-level resource maintenance, such as recycling buffers. For performance reasons, most filters keep a pool of filter buffers and only allocate or free them when the pool becomes empty or too large to save on the overhead of the memory management.

c. ***Do not attempt to change the notification flags in the metabase***: The metabase property *FilterFlags* contain flags that indicate to IIS which notification types a particular ISAPI filter can support. However, this property is populated by IIS from the bit flags passed in the HTTP_FILTER_VERSION structure from GetFilterVersion (). Filter Flags should be considered read-only.

4.4.2 Flow of the Filter [6]

This section explains what happens when a filter is added to a Web server. A server processes information from a client in seven steps as graphically represented in Fig 2.1. The IIS works fine as long as one does not change the default processing of HTTP requests through IIS. An ISAPI filter, however, allows us to change the default processing of an HTTP request. Fig-4.4 is a graphical representation of the IIS flow with filters added.

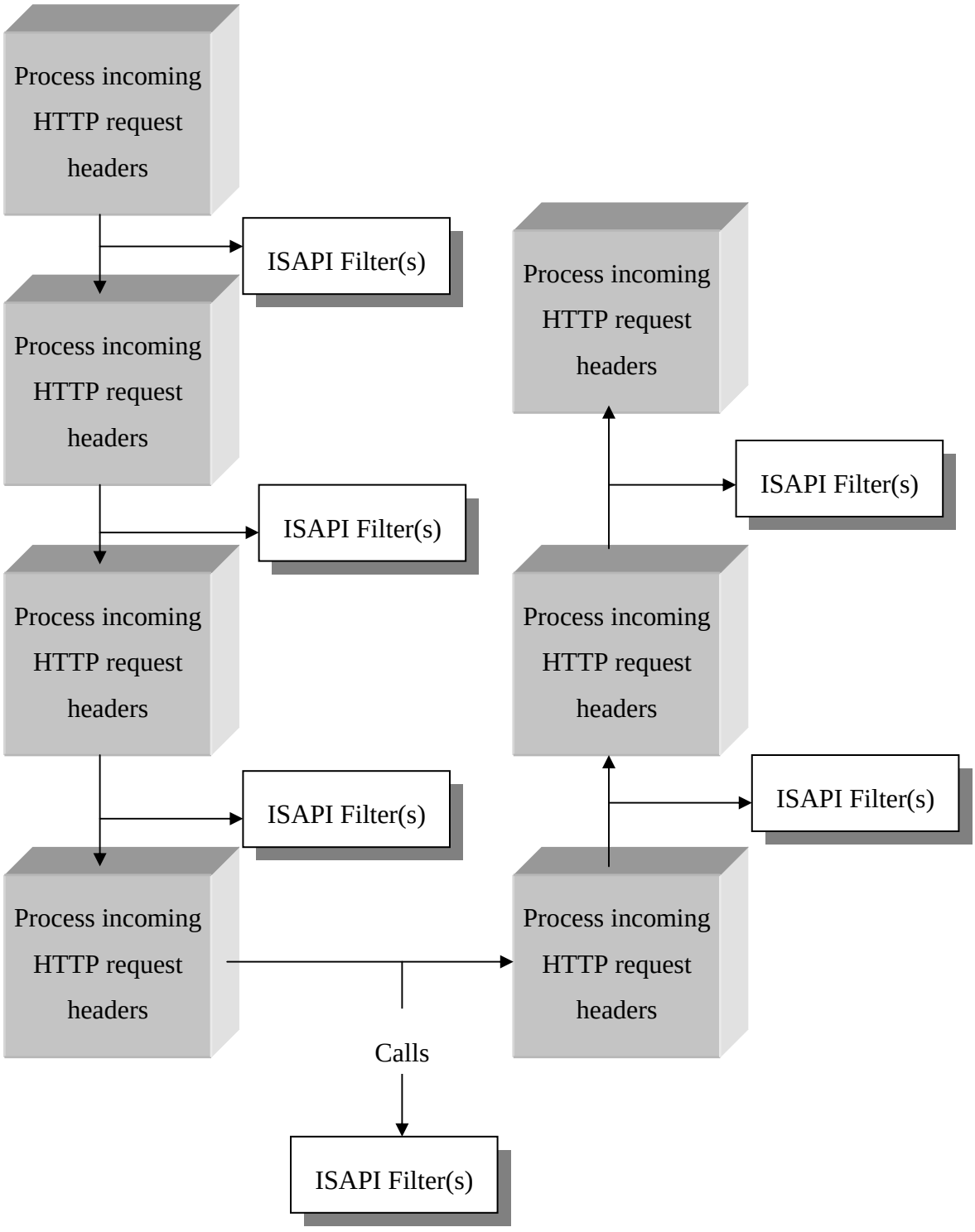

Fig 4.4-Flow of a request in IIS with filters

Fig 4.4 demonstrates that IIS makes calls to the registered filters as it is processing each HTTP request. This allows any ISAPI filter to change the default processing. Each call is different in that it gives the filter a chance to process a different event during the processing of the request. Each filter can choose to process the event or to ignore it. If the filter ignores it, IIS processes the event.

The function HttpFilterProc () processes the request in most of the filters in five steps as shown in the Table 4.3.

Table 4.3 - Process in HttpFilterProc () when request comes in

Step Number	**What Happens**
1.	Determine the notification type (refer to Table 3.3)
2.	Decide whether to process this occurrence.
3.	If yes, process it.
4,	Decide whether IIS should pass it on or not.
5.	Return the proper status to IIS

4.4.3 ISAPI Filter Design Rules

While designing ISAPI filter, following design rules have been considered:

a. ***Must Be Registered before IIS Starts.:*** - Because IIS loads all ISAPI filters at startup, it must be clearly defined that which DLL to load. In IIS 3.0 it is done by adding an entry to the registry by using command REGEDT32.EXE. However in IIS 4.0 filter is loaded through Microsoft Management Console (MMC). After the filter is loaded WWW service must be stopped and then

start it again. Since for this project IIS 4.0 has been used filter has been registered through MMC in default Web site.

b. ***Must Be A 32-Bit DLL: -***. Each ISAPI filter must be a 32-bit DLL. This is because IIS itself is 32-bit program and it must call entry points in the filter DLL. For this purposed *Application Wizard* of Visual C++ is used to create the raw skeleton of ISAPI DLL. *Application Wizard* automatically creates skeleton of 32-bit IAPI DLL.

c. ***Must Expose Defined Entry Points****:* - Each filter DLL must have the *GetFilterVersion ()* and *HttpFilterProc ()* entry points defined. This is the only way IIS can communicate with the DLL. If these entry points are not defined, IIS won't be able to load and use the filter DLL. For defining the entry points a definition file has been created by the *Application Wizard* with *.def* extension. This file defines the two functions as exported and enable IIS to communicate with itself.

d. ***Must Be Thread-Safe:*** - IIS makes extensive use of Windows NT's multithreading capabilities. It can respond to many requests simultaneously. This means that all ISAPI filters must be thread-safe. For making the filter thread safe, firstly DAO classes for accessing the database have been avoided in the filter and instead Direct ODBC has been used that is thread safe as well as carries fast response to access the database. Secondly, for such part of the filter where multithreading may cause problems, use of *critical sections* is made to ensure one thread at a time.

4.5 Summary

This chapter has discussed design issues of the project. Design issues have been divided into three basic groups namely *Internet Account Manager* (IAM), *IAM database* and *Authentication and Logging Filter*. Firstly, database design is explained in two parts, for authentication filter and then for logging filters. Secondly, IAM is discussed in a scenario of database front-end and prime design parameters are given user point of view. Thirdly, ISAPI filter design requirements and strategies employed for that matter are elaborated.

Chapter 5
Implementation Issues

5.1 Introduction

This project has two basic components, which interact with the *IAM database*. These components are namely *IAM* and *ISAPI filter* Database keeps all the information about users, groups (to which users can be assigned) and directories those are registered as protected. All sort of event logging information is also being maintained by this database. Figure 4.1 graphically explains the conceptual working of this project in the form of block diagram.

Separate workspaces are developed for both of the components because filters do not allow any mean of communication other than the browser i.e. the filters do not support view windows and dialog boxes. View of each workspace is shown in fig 5.1

IAM communicates with the *IAM database* through Data Access Object (DAO) classes of MFC. DAO classes provide more power and flexibility [5] to handle databases through MJDE than ODBC but DAO classes are not thread-safe in a multi-user environment when multiple users are accessing the database simultaneously. However, since *IAM* is designed for the utilization of Account Manager (AM) alone there is no chance of simultaneous access to the database through *IAM*.

ISAPI filter interacts with the database through *DirectODBC* API tool. *DirectODBC* is employed firstly due to its fast response time as compare to DAO classes in accessing the database secondly *DirectODBC* is thread-safe API. ISAPI filter works in multi-user environment where a Web server may be accessed by hundreds of browsers on Internet simultaneously.

This chapter discusses the implementation of *IAM* in detail and then makes extensive discussion on the implementation of ISAPI filter. Diagrammatically, the implementation layout of the whole system is shown in fig 5.2.

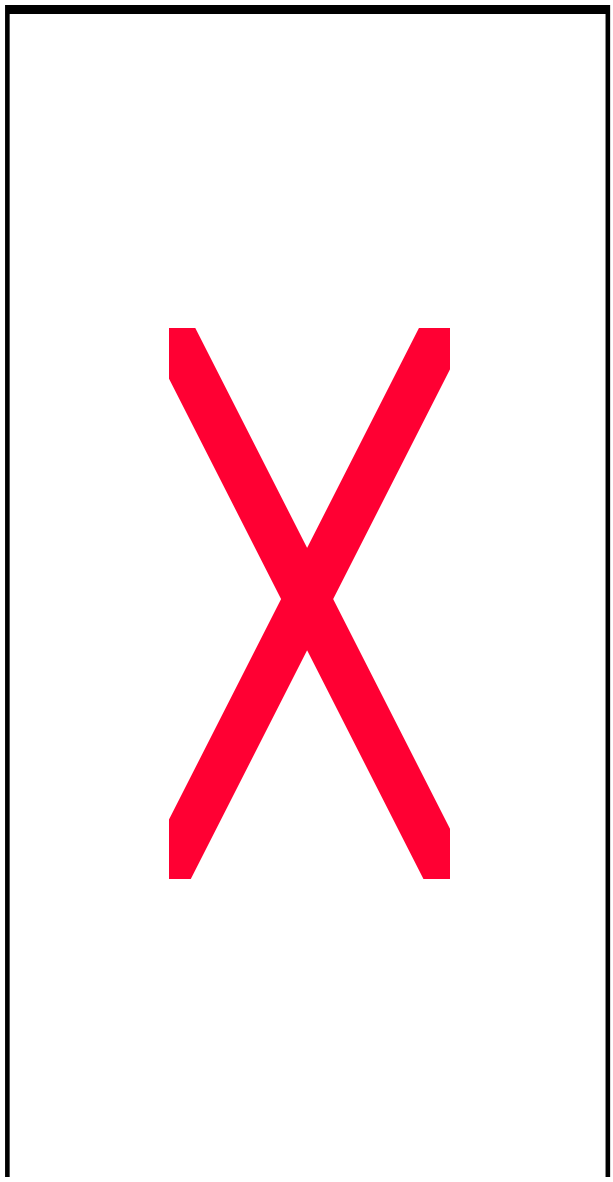

Fig 5.1 (a). *IAM Workspace.*

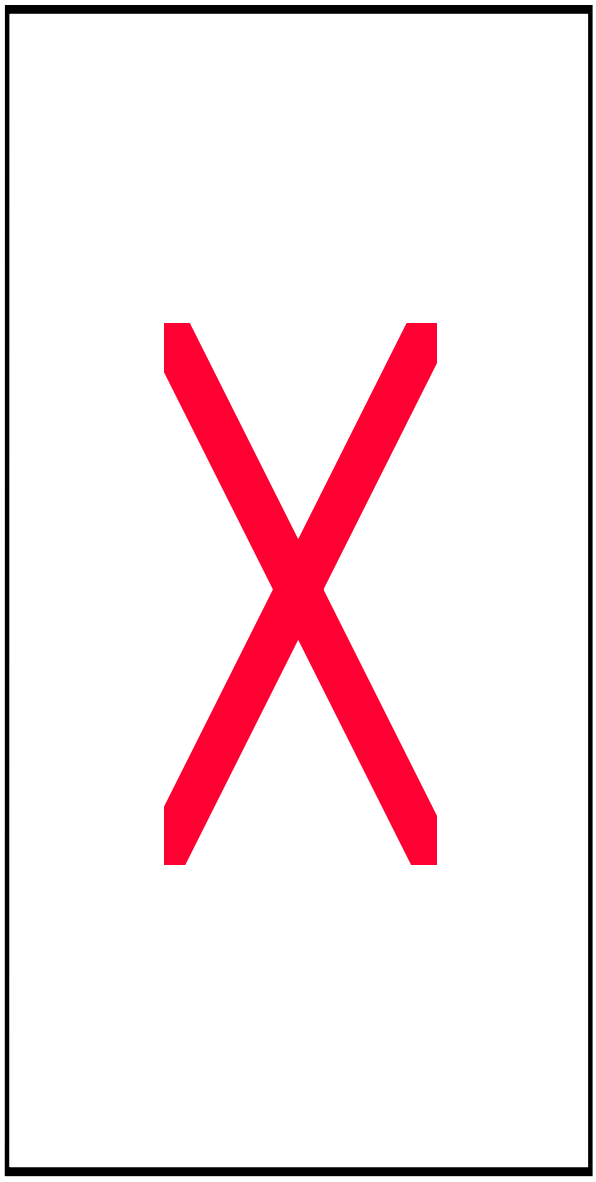

Fig 5.1 (b). *Filter Workspace.*

Fig 5.1- Workspace view of the software

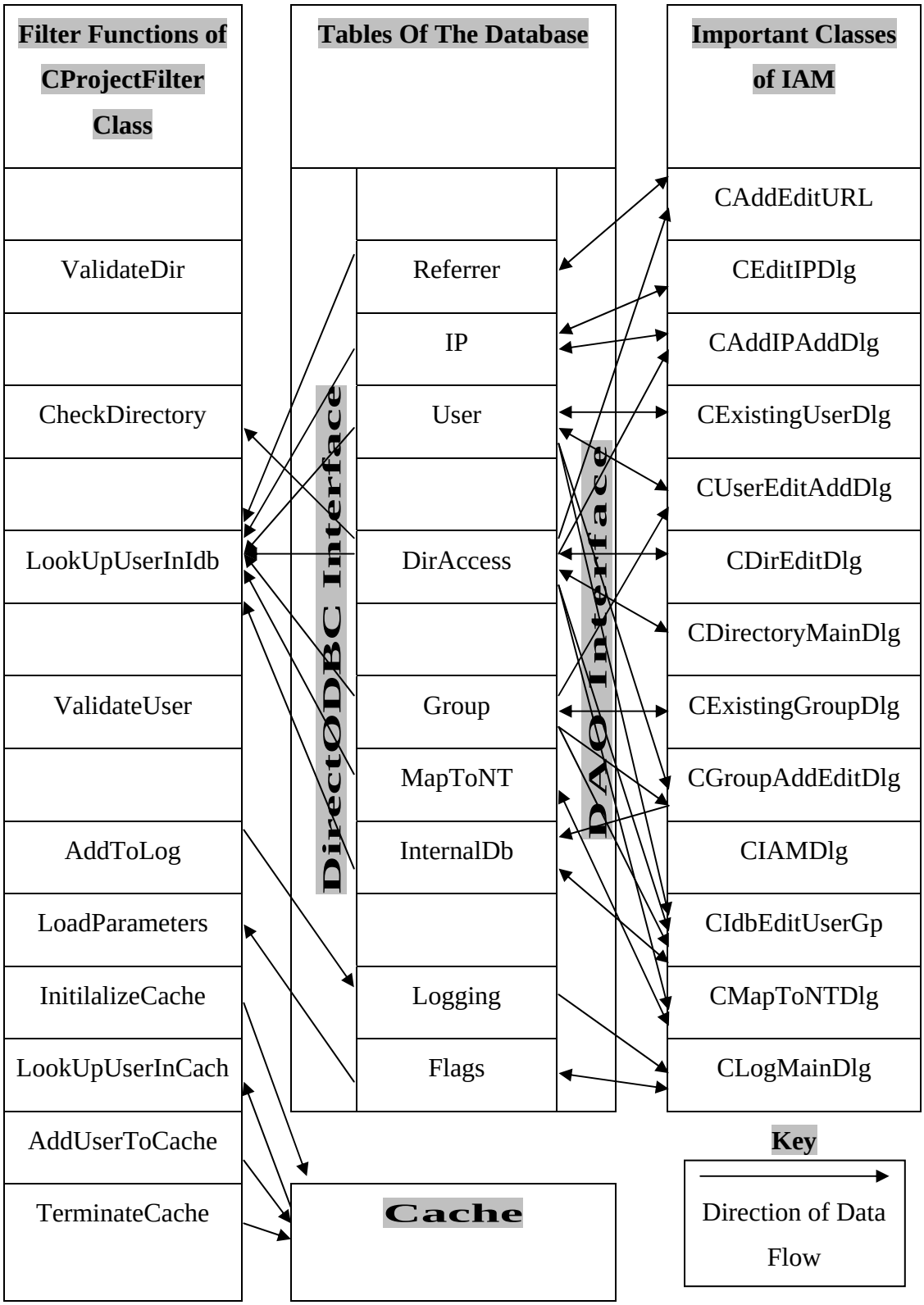

Fig 5.2 - Implementation model of the software

5.2 Implementation Of IAM

IAM is a dialog-based applications developed through *Application Wizard* of Visual C++ 6. An extensive use of Microsoft Foundation Classes ((MFC) has been made because of its user friendly interface. MFC programming is done through object oriented approach in which, different modules are encapsulated in different classes performing a specific task. Normally each dialog box and its visible objects are encapsulated in a separate class. Each *class* may have a number of *member functions* and *member variables* to execute the logic.

IAM is developed with the combination of 23 classes and 201 functions total. This section will discuss the classes and functions that form core of the design. Each *class* will be discussed in a separate table along with its important *member functions* and *member variables*.

Table 5.1 - Implementation details of CIAMDlg class

Class Name /Base Class:		CIAMDlg / CDialog
Class Purpose / Description	The class is used to handle the main 'Internet Account Manager' dialog prompted to the user at the start of the program.	
Member Functions		
Visibility	**Name**	**Description**
Protected	OnBtnGpAdd ()	This function is used to call the dialog box that adds a new group to the group table in IAM database.
Protected	OnBtnDirectory ()	This function is used to call the dialog box that displays the protected directories.
Protected	OnBtnLogging ()	This function is used to call the dialog box that displays the log view.

Protected	OnGpExisting ()	This function is used to call the dialog box that displays already registered groups.
Protected	OnUserExisting ()	This function is used to call the dialog box that displays already registered users

Data Members

Visibility	Type	Name	Description
Protected	CExistingUserSet	m_UserSet	This is an object of CExistingUser Set class. This class is derived from CDaoRecordset class of MFC. This object is used to access the '*User*' table of the database.
Protected	CExistingGroupSet	m_GroupSet	Similar object of derived class from CDaoRecordset. It is used to access the '*group*' table.
Protected	CButton	pButton	Pointer object of CButton class, used to disable the '*Existing user'* and *'Existing group'* buttons if no user or group is registered in the database yet.

Table 5.2 - Implementation details of CAddEditURL class

Class Name /Base Class:	CAddEditURL / CDialog
Class Purpose / Description	The class is used to add and delete the permitted referrer for current directory.

Member Functions

Visibility	**Name**	**Description**
Protected	FillCombo ()	Fills the combo box with permitted referrer URLs from '*Referrer*' table.
Protected	OnButtonAdd ()	Adds referrer URL to the combo box.
Protected	OnButtonDel ()	Delete referrer from the combo box.
Protected	OnOk ()	Update the Referrer table from the combo box.

Data Members

Visibility	**Type**	**Name**	**Description**
Public	CDaoDatabase	db	Used to open the database through properties of CDaoDatabase class.
Public	CString	m_EditDir	Keeps the value of current directory
Public	CStrings	m_EditURL	Value type variable of edit box. Used to receive referrer URL manually added by the user.
Public	CDaoRecrdset	rs	Object-used to open Recorset of the table Referrer.

Table 5.3 - Implementation details of CAddIPAddDlg class

Class Name /Base Class:	CAddIPAddDlg / CDialog
Class Purpose / Description	The class is used to add IP address to the currently selected directory.

Member Functions

Visibility	**Name**	**Description**
Protected	OnRadioNot Permit ()	Resets the flag if a radio button '*Not Permitted*' is checked.
Protected	OnRadioPermit ()	Sets the flag if a radio button Permitted is checked.
Protected	OnInitialDialog ()	Opens the database table '*IP*' for addition.

Data Members

Visibility	**Type**	**Name**	**Description**
Public	CDaoDatabase	dbIP	Used to open the database through properties of CDaoDatabase class.
Public	CIPAddressCtrl	m_CtlIPAdd	Control variable of *'IP control box'*.
Public	CStrings	m_CurDir	Keeps the value of currently selected directory..
Public	CDaoRecrdset	rsIP	Object-used to open Recorset of the table *IP*.

Table 5.4 - Implementation details of CDirectoryMainDlg class

Class Name /Base Class:	CDirectoryMainDlg / CDialog
Class Purpose / Description	The class is used to display all the currently registered directories. It also adds new and deletes the displayed directories.

Member Functions		
Visibility	**Name**	**Description**
Protected	FillCombo ()	Fills the registered directories in the combo box from *DirAcces* table..
Protected	OnAdd ()	It runs ShBrowseForFolder () function that opens a 'Directory Tree' for folder selection.
Protected	OnEdit ()	Opens an other dialog box for editing the protection methods for the selected directory

Data Members			
Visibility	**Type**	**Name**	**Description**
Public	CComboBox	m_ctlDirCombo	Control variable to manipulate the directory combo box.
Public	CDaoDatabase	db	Used to open the database through properties of CDaoDatabase class.
Public	CStrings	strQuery	Keeps the value of database query.
Public	CDaoRecrdset	rs	Object-used to open Recorset of the table *DirAccess*.

Table 5.5 - Implementation details of CDirEditDlg class

Class Name /Base Class:	CDirEditDlg / CDialog	
Class Purpose / Description	The class is used to further open different dialog boxes in order to set corresponding protection methods for the currently selected directory.	
Member Functions		
Visibility	**Name**	**Description**
Protected	OnBtnIdbEditGroup () OnBtnIdbEditUser () OnButtonIpAdd () ObButtonIpEdit () OnButtonMapEdit () OnButtonURLEdit ()	All these functions are activated by their corresponding buttons and opens new dialog boxes for setting up the protection method
Protected	OnCheckDb () OnCheckIp () OnCheckMap () OnCheckReferer ()	All these functions are activated when corresponding checkboxes are clicked by mouse. These functions hide /show the corresponding buttons if that checkbox is unchecked / checked respectively.

Table 5.6 - Implementation details of CEditIPDlg class

Class Name /Base Class:	CEditIPDlg / CDialog
Class Purpose / Description	The class is used to display all the currently registered IP addresses in *permitted* or *not permitted* category list boxes. Registered IP addresses can be deleted or their category can be changed among *permitted* or *not permitted* ones.

Member Functions		
Visibility	**Name**	**Description**
Protected	FillBothLists ()	Fills both the *permitted* or *not permitted* category list boxes from the *IP* table of the database.
Protected	UpdateIps ()	Updates the *IP* table from both the lists at closing of the dialog box.
Protected	OnBtnDelete ()	Deletes the selected IP address from any of the list.

Data Members			
Visibility	**Type**	**Name**	**Description**
Public	CIPAddressCtrl	m_ctlIPAdd	Control variable to manipulate the IP addresses.
Public	CListBox	m_ctlPermitList	Control variable for filling permitted IP addresses in the list.
Public	CStrings	m_EditDir	Keeps the value of currently selected directory.

Table 5.7 - Implementation details of CExistingGroupDlg class

Class Name /Base Class:	CExistingGroupDlg / CDialog
Class Purpose / Description	The class is used to display all the currently registered groups in a combo box and *deletes, changes* its member users or *registers* new group.

Member Functions		
Visibility	**Name**	**Description**
Protected	FillCombo ()	Fills the group combo box from the group table.
Protected	OnAddGroup ()	Activates by *'Add'* button to open a dialog for adding a group.
Protected	OnBtnEdit ()	Opens a new dialog box to change the member users of the selected group when *'Edit'* button is pressed.

Data Members			
Visibility	**Type**	**Name**	**Description**
Public	CComboBox	m_ctlGroup Combo	Control variable to manipulate the group combo box.
Public	CEistingGroupSet	m_Existing GroupSet	Object of a derived class from CDaoRecordset class for accessing the *group* table of the database.
Public	CButton	pButton	Object of CButton class for enabling / disabling the buttons according to specific requirements.

Table 5.8 - Implementation details of CExistingUserDlg class

Class Name /Base Class:	CExistingUserDlg / CDialog
Class Purpose / Description	The class is used to *display* all the currently registered users in a combo box and *deletes*, or *registers* new users.

Member Functions		
Visibility	**Name**	**Description**
Protected	FillCombo ()	Fills the group combo box from the *user* table.
Protected	OnAdd ()	Activates by *'Add'* button to open a dialog for adding a new user to the database.
Protected	OnEdit ()	Opens a dialog box to change user credentials.
Protected	OnDelete ()	Deletes the currently selected user from the database.

Data Members			
Visibility	**Type**	**Name**	**Description**
Public	CComboBox	m_ctlGroup Combo	Control variable to manipulate the group combo box.
Public	CEistingUserSet	m_Existing UserSet	Object for accessing the *user* table of the database.
Public	CEistingGroupSet	m_Existing GroupSet	Object for accessing the group table of the database.

Table 5.9 - Implementation details of CGroupAddEditDlg class

Class Name /Base Class:	CGroupAddEditDlg / CDialog
Class Purpose / Description	The class is used to register a new group to the database and also changes the user members of the selected group by selection.

Member Functions

Visibility	**Name**	**Description**
Protected	FillMemberCombo () FillNonMemCombo ()	Fills the member user and non-member user combo boxes for further selection by the operator.
Protected	OnButtonAddMem ()	Adds a user from non-member list to member list.
Protected	OnButtonRemMem ()	Adds a user from member list to non-member list.
Protected	OnButtonAdd ()	Adds a new group to the group table.
Protected	OnOK ()	Updates the *InternalDb* table.

Data Members

Visibility	**Type**	**Name**	**Description**
Public	BOOL	m_bAddMode ()	Flag is set to 1 when a new group is to be added, else remains 0.
Public	CQryGpMember	m_Member UserSet	Object for accessing the *group* table of the database.
Public	CString	strGroupName	Keeps the value of selected group.

Table 5.10 - Implementation details of CIdbEditUserGp class

Class Name /Base Class:	CIdbEditUserGp / CDialog
Class Purpose / Description	The class is used to allocate or de-allocate users and groups to the selected directory.

Member Functions		
Visibility	**Name**	**Description**
Protected	FillPermitList () FillNotPerList ()	Fills the groups or users in the *'Member'* list and *'non-Member'* list from *InternalDb* table for the selected directory.
Protected	OnButtonAddMem ()	Moves selected users or groups from *'non-Member'* list to *'Member'* list.
Protected	OnButtonRemMem ()	Moves selected users or groups from *'Member'* list to *'non-Member'* list.
Protected	UpdateDbFromList ()	Updates the *InternalDb* table according to final layout of the two lists.

Data Members			
Visibility	**Type**	**Name**	**Description**
Public	CString	m_EditDir	Keeps the value of currently selected directory.
Public	CComboBox	m_ctlPermitList m_ctlNotPerlist	Handles the *permitted* & *not-permitted* user or group list boxes.
Public	CEistingGroupSet	m_Existing GroupSet	Object for accessing the group table of the database.

Table 5.11 - Implementation details of CLogMainDlg class

Class Name /Base Class:		CLogMainDlg / CDialog	
Class Purpose / Description		The class is used to display the '**Log View**' along with various switches to set different '**Logging Options**'.	
Member Functions			
Visibility	**Name**	**Description**	
Protected	AutoClearLog ()	Clears the log automatically if *'Auto Clear'* is enabled.	
Public	LoadFlags ()	Loads already set flags from *'Flags'* table.	
Protected	OnBtnClearLog()	Manually clears the log. Activated by a button.	
Public	SetFlags ()	Sets the current environment in the *'Flags'* table.	
Protected	ShowHeadings ()	Displays the headings of *'List Control Box'* for log.	
Protected	ShowData ()	Displays the logging data, picked from *logging* table.	
Data Members			
Visibility	**Type**	**Name**	**Description**
Public	Integer	RadioOption	Keeps the value of currently selected option for auto-log clear.
Protected	CListCtrl	m_ctlLogList	Object to handle all functions of the *'List Control'* that displays log.
Public	CButton	m_ctlShowUser	Control variable of *'Unmapped Remote User*' **check box**. Enables or disables this view in the log.

Table 5.12 - Implementation details of CMapToNTDlg class

Class Name /Base Class:	CMapToNTDlg / CDialog
Class Purpose / Description	The class is used to map internal users of NT ACL to the selected directory so that when Internet user that does not have access to the directory in NT, the mapped user will be provided to IIS at the time of accessing the directory.

Member Functions		
Visibility	**Name**	**Description**
Protected	FillCombo ()	Fills the combo box with mapped users.
Protected	OnButtonAdd ()	Adds an NT user name and user password to the combo box.
Protected	OnBtnDelete ()	Deletes the selected user name from the combo box.

Data Members			
Visibility	**Type**	**Name**	**Description**
Public	CString	m_NTUser	Keeps the value of manually entered username..
Public	CListBox	m_ctlNTUser	Control variable for filling the user names in the list.
Public	CStrings	m_NTPassword	Keeps the value of manually entered password

Table 5.13 - Implementation details of CUserEditAddDlg class

Class Name /Base Class:	CUserEditAddDlg / CDialog
Class Purpose / Description	The class is used to register a new user to the database and also changes the credentials for registered user.

Member Functions		
Visibility	**Name**	**Description**
Public	FillGroupCombo ()	Fills the list with all registered groups.
Protected	CountDaysLeft ()	Calculates the no of days to expire the account.
Protected	ShowControls ()	Adds a user from member list to non-member list.
Protected	OnDatetimechange PickDate (*pNMHDR, *pRESULT)	Function is invoked whenever a date is selected from the Calendar control.

Data Members			
Visibility	**Type**	**Name**	**Description**
Public	BOOL	m_bAddMode ()	Flag is set to 1 when a new user is to be added, else remains 0.
Public	CString	m_Confirm Pasword	Keeps the value of the confirmed password.
Public	CDateTimeCtrl	m_ctlPickDate	Object to get date from the Calendar
Public	COleDateTimeSpan	m_DaysLeft	Object-finds the difference of dates
Public	CEdit	m_ctlTlimit	Keeps date in short format.
Public	CInternalDbSett	m_IntDB	Object-accesses the *InternalDb* table.

5.3 Implementation Of ISAPI Filter

The ISAPI filter DLL is loaded on IIS before it starts functioning. For developing the ISAPI filter IDE of Visual C++ 6.0 provides *Application Wizard* to provide basic functionality of the filter. The filter is designed for two basic purposes. Firstly, for *authentication* of the user and secondly, for *logging* the events.

In addition to basic purposes, the filter has been design to achieve optimum efficiency by reducing the authentication time for the client requesting any resource through the browser. Authentication time has been reduced by using a '*cache*' in the RAM so that whenever a user has been provided access to specific protected directory after a resource extensive authentication from the database, all the successful credentials are saved in the *cache* for later use by IIS. The *cache* works extremely fast because it is created in the RAM. The *cache* has been implemented as '*circular double link list'* of a structure (*USER_INFO*).

Since ISAPI filter works in multi-user environment it is a thread-safe design. It has been designed to work perfectly when simultaneous clients are accessing the Web server. In all such parts of the filter those are not thread-safe, *critical sections* are made to avoid parallel execution of that part.

5.3.1 Filter DLL Functions

This DLL has been created with *MFC Application Wizard.* A user-defined class **CProjectFilter** creates the DLL**.** This class is derived from **CHttpFilter** class of MFC. All the functions are member of **CProjectFilter** class. There are 18 functions in total. 8 out of these are system-defined and rests of all are user-defined function. In this section the system-defined functions are explained briefly in table 5.14. The code of these functions has been modified as per the system requirements.

Table 5.14 - Implementation details of system-defined functions of the filter

Class Name /Base Class:		CProjectFilter / CHttpFilter
Class Purpose / Description		The class is used to create an ISAPI filter DLL.
System-Defined Functions		
Visibility	**Name**	**Description**
Public	CProjectFilter ()	Constructor. Initializes cache by calling InitilizeCache () function.
Public	~CProjectFilter ()	Destructor. Releases cache memory by calling TerminateCache () function when Web service stops.
Public	GetFilterVersion	Called by IIS when filter is initially loaded. Explained in section 3.5.2.2.
Public	HttpFilterProc	An entry-point function. Explained in section 3.5.2.2.
Public	OnUrlMap	Called by SF_NOTIFY_URL_MAP notification explained in section 3.5.2.3.
Public	OnAuthentication	Called by SF_NOTIFY_AUTHENTICATION notification explained in section 3.5.2.3.
Protected	OnLog	Called by SF_NOTIFY_LOG notification explained in section 3.5.2.3.
Protected	OnEndOfNetSession	Called by SF_NOTIFY_END_OF_NET_SESSION notification explained in section 3.5.2.3..

5.3.2 Database Handling Functions

Database handling functions use *DirectODBC* interface to access the database for two reasons. Firstly, nothing matches the speed of *DirectODBC* in access time of the database. Secondly, it is thread-safe interface. There are 4 different functions used to handle the operation of the database. These functions are briefly explained in tables 5.15 to 5.19.

Table 5.15 - Implementation details of ValidateDir function

Name of the Function	ValidateDir
Visibility	Public
Return Value	BOOLIAN
Arguments	CHAR * pszUserName CHAR * pszPassword
Description	
This function is called by OnAuthentication () function of the filter. This function is ignored if the remote user is an anonymous one. This function further calls under mentioned two functions 1. *CheckDirectory*: - Checks whether the requested directory is registered for protection or otherwise. This function is explained in table 5.16 2. *ValidateUser:* -Called only if the requested directory is found to be registered one. This function is explained in table 5.17	

Table 5.16 - Implementation details of CheckDirectory function

Name of the Function	CheckDirectory
Visibility	Public
Return Value	BOOLIAN
Arguments	BOOL *mb_IP, BOOL *mb_URL, BOOL *mb_MapToNT, BOOL *mb_InternalDb
Description	
This function is called by *ValidateDir* function of the filter. The function checks the database table *'DirAccess'* to search for the requested directory. If directory is found, it sets TRUE in its argument values for those protection methods that are enabled. The function returns TRUE if the requested directory is found otherwise returns FALSE.	

Table 5.17 - Implementation details of ValidateUser function

Name of the Function	ValidateUser
Visibility	Public
Return Value	BOOLIAN
Arguments	CHAR * pszUserName, CHAR * pszPassword, BOOL * pfValid
Description	

This function is called by *ValidateDir* function of the filter. It finds out if the user is allowed to access the directory and if allowed, will transform the first two argument values to NT username and password. If all the enable protection tests are passed, it will set argument value *pfValid* to TRUE. For achieving these objectives the function calls under mentioned three functions.

1. *LookupUserInCache:* - Firstly, the user credentials are found in the cache by this function. This function is explained in table 5.21

2. *LookupUserInIDb:* - Secondly, if credentials are not found in the cache, this function finds them in the database. This function is explained in table 5.18.

3. *AddUserToCache:* - Thirdly, this function is called to add such credentials in the cache that are successfully passed by the database. This function is only called if credentials are not found in the cache and they are successfully found in the database. This function is explained in table 5.22.

.

Table 5.18 - Implementation details of LookupUserInIDb function

Name of the Function	LookupUserInIDb
Visibility	Public
Return Value	BOOLIAN
Arguments	CHAR * pszUserName, BOOL * pfFound, CHAR * pszPassword, CHAR * pszNTUser, CHAR * pszNTUserPassword
Description	

This function is called by *ValidateUser* function (table 5.17). It finds out all the enabled protection methods from the database. All the enabled protection methods are checked sequentially and if any of the tests fail, argument value for *pfFound* is set to FALSE otherwise remains TRUE. Following sequence is used:

1. **Internal Database Test**: -The test is performed if protection method '*by Internal database'* is enabled. It checks group of the user in *InternalDb* table that whether this group is provided access of the directory or otherwise. If group is not provided access then user is checked for direct access of the directory.

2. **Referrer Test: -**This test is performed, if protection method '*by referrer'* is enabled.

3. **IP Address Test: -** This test is performed, if protection method '*by IP address'* is enabled.

4. **MapToNT Test: -** This test is performed, if protection method *by 'MapToNT'* is enabled. This test provides the NT user name and password to IIS for authentication for those remote users that do not have local account on the server. This is only required for NTFS directory.

Table 5.19 - Implementation details of AddToLog function

Name of the Function	AddToLog
Visibility	Public
Return Value	VOID
Arguments	VOID
Description	

This function is called by *OnLog* function (table 5.14). The function is called, only if logging has been enabled in IAM. *LoadParameter* () function checks enabling of logging. *AddToLog ()* function provides the log to the following events.

1. Requested resource on the server e.g. *home.htm* document.
2. Remote username that has requested a resource through browser e.g. *farooq*.
3. Remote IP address of the client browser e.g. *(1.1.1.127)*
4. URL of a Web server that has referred a user to this IIS Web server. E.g. *http://micrrosoft.com* .
5. The properties of the client browser (User Agent) that has accessed the IIS server. E.g *I.E. 5.0*
6. Status of the connection e.g. *Access Denied, OK, Access Granted* etc.

5.3.3 Cache Structure

The *cache* has been implemented as '*circular double link list*' of a structure (*USER_INFO*). This structure is given in fig 5.3:

```
typedef struct _USER_INFO
{
        LIST_ENTRY  ListEntry;  // Double linked list entry
        CHAR  achUserName[SF_MAX_USERNAME];
        CHAR  achPassword[SF_MAX_PASSWORD];
        CHAR  achDirectory[255];
        CHAR  achRemoteIP[255];
        CHAR  achReferer[255];
        CHAR  achNTUserName[SF_MAX_USERNAME];
        CHAR  achNTUserPassword[SF_MAX_PASSWORD];
} USER_INFO, *PUSER_INFO;
```

Fig 5.3-Cache structure

The structure consists of eight *member* variables. One of the members *ListEntry* is used to establish the link list and rests of the seven are used to store the credentials usable during authentication.:

5.3.4 Cache Handling Functions

There are 4 different functions used to handle the operation of the *cache*. These functions are briefly explained in tables 5.20 to 5.23.

Table 5.20 - Implementation details of InitializeCache function

Name of the Function	InitializeCache
Visibility	Public
Return Value	BOOLIAN
Arguments	VOID
Description	
This function is use to initialize the cache. It returns TRUE if initialized successfully and returns FALSE on failure. Initialization includes the creation of a blank 'circular double link list'.	

Table 5.21 - Implementation details of LookupUserInCache function

Name of the Function	LookupUserInCache
Visibility	Public
Return Value	BOOLIAN
Arguments	CHAR * pszUserName, BOOL * pfFound, CHAR * pszPassword, CHAR * pszNTUser, CHAR * pszNTUserPassword
Description	
This function is used to find the received credentials in the cache. It finds each entry one by one and if an entry is found, then variable *pfFound* .is set to TRUE otherwise remains FALSE. In case, if credentials are found and mapping of directory to the NT user is enabled, then strings *pszNTUse*r and *pszNTUserPassword* are filled up with the mapped username and password. When a credential is found, the credentials entry is brought in front in the list, if it's not already near the front. Purpose of moving credentials in front of the list is to further reduce authentication time in the cache.	

Table 5.22 - Implementation details of AddUserToCache function

Name of the Function	AddUserToCache
Visibility	Public
Return Value	BOOLIAN
Arguments	CHAR * pszUserName, CHAR * pszPassword, CHAR * pszNTUser, CHAR * pszNTUserPassword
Description	
This function is use to add the specified user to the cache. It returns TRUE if no error occurs in the process. Firstly, it Searches the cache for the specified user to make sure there are no duplicates and if user is found it overwrites the existing entry. If duplicate doesn't exist then new entry is made at the front of the list.	

Table 5.23 - Implementation details of TerminateCache function

Name of the Function	TerminateCache
Visibility	Public
Return Value	VOID
Arguments	VOID
Description	
This function is called when the Web service (loaded with the filter) is stopped. It releases the entire allocated memory of the cache.	

5.4 Summary

This project has two basic components, which interact with the *IAM database*. These components are namely *IAM* and *ISAPI filter*.

A separate workspace has been developed for both of these because filters do not allow any means of communication other than the browser i.e. the filters do not support view windows and dialog boxes.

This chapter discussed the implementation of *IAM* with the help of important user-defined classes, which form heart of the program.

Later, elaborate details on the implementation of ISAPI filter have been given with the help of user-defined functions. Development of cache is also an important feature of ISAPI filter that has been explained exclusively.

Chapter 6
User Guide

6.1 Introduction

This system has been developed as two separate projects namely *IAM* and *ISAPI Filter*. IAM is a user interface for managing accounts for Internet users and ISAPI filter is the interface of IIS for client browsers. This chapter will firstly explain the installation procedure of ISAPI filter on IIS 4.0 then installation procedure of IAM will be discussed. Secondly, working of each dialog of IAM will be elaborated.

6.2 Installation Procedures

The software uses external data source (IAM database), for managing all the credentials. External data source is a Microsoft Access 97-database *xproject.* Since ISAPI filter is using *DirectODBC* interface, the database is registered *as 'System DSN'* in '*ODBC Data Source Administrator'*. This database is located in the default directory of IAM after it is installed. The data source name is entered as '*project*' during installation.

6.2.1 ISAPI Filter

After registration of data source, the filter has to be loaded in IIS. Loading procedure in IIS 3.0 is somewhat complex because filter entry is made in the 'Windows Registry'. However, since IIS 4.0 is used for development of the software, loading is simple. Here loading of the filter in IIS 4.0 is elaborated in table 6.1

Table 6.1-Steps for adding a filter to a Web server or Web site

1. In '*Internet Service Manager'*, select the *Web server* or *Web site* and open its property sheets.
2. Click the **ISAPI Filters** tab.
3. Click the **Add** button.

4. Type the name of the filter in the **Filter Name** box and either type or browse for the DLL file in the **Executable** box.
5. Click **OK**.
6. To change the load order of a filter, use the arrows.
7. If a global filter has been added or changed, the Web server must be stopped and restarted to load the new filters into memory. A filter that is added at the Web site level is automatically loaded when added.

6.2.2 IAM

A user-friendly installation procedure has been developed by IAM. Installation procedure has been developed in such a way that it prompts the user about the task for each installation step. The program can also be executed without installation by running *IAM.exe* file.

6.3 User Interface-IAM

The interface has been kept as simple as possible. This interface has been created with Visual C++ dialog based application wizard. The entire user interaction is visual. It has been tried that user should manually enter bare minimum values and most of the choices are selected with the mouse click. When the program is executed the first dialog (Internet Account Manger) that appears, is shown in fig 7.1.

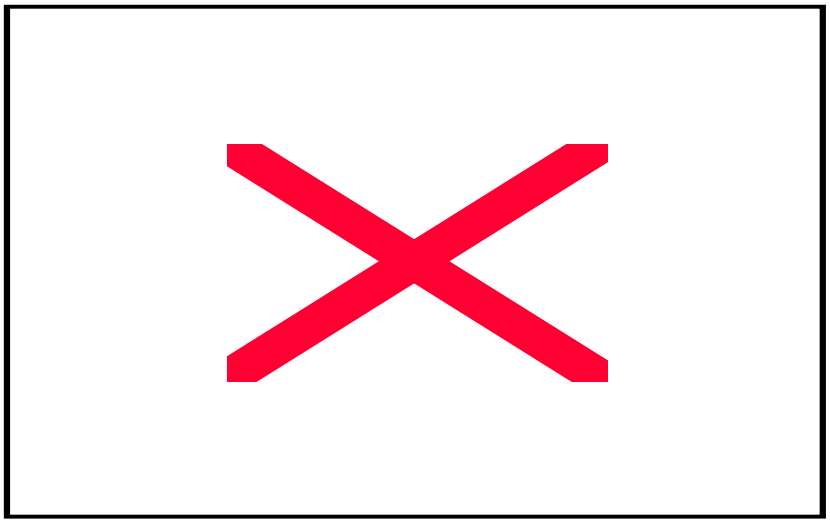

Fig 6.1- Internet Account Manger-Main Dialog

The Internet Account Manger of fig 7.1 is the main dialog. It displays a number of options selectable by different buttons. Each button does to following:

a. **User Existing**: -Opens a dialog to display already registered users.(Fig 6.3)

b. **User Add**: - Opens a dialog to register a new user to the database.(Fig 6.4)

c. **Group Existing**: - Opens a dialog to display already registered groups.(Fig 6.6)

d. **Group Add**: - Opens a dialog to register a new group to the database.(Fig 6.7)

e. **Customize Directory: -** Opens a dialog to display already registered directories. These may NTFS or FAT directories. (Fig 7.7)

f. **Customize Logging: -** Opens a dialog to display log viewing dialog (Fig 7.8)

g. **OK**: - Closes the dialog box.

h. **Help**: - Used to invoke the online help about IAM.

i. **About**: - Displays the dialog that shows the version of the application and the developer's name as shown in Fig 6.2

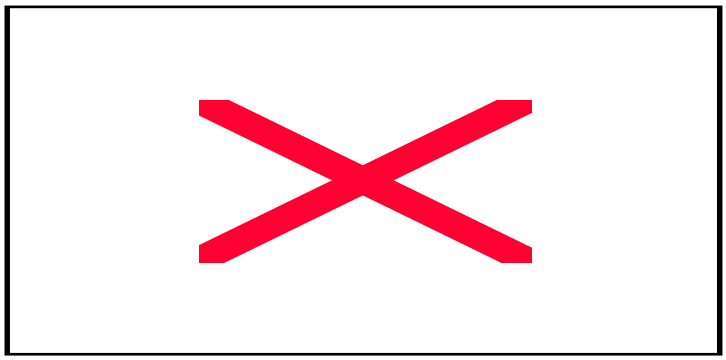

Fig 6.2-About IAM Dialog

A number of dialogs are opened for setting up different credentials. There are total of 17 dialogs developed for the entire interface. These dialogs can be distributed under following main subjects.

a. User manipulation Dialogs

b. Group manipulation Dialogs

c. Directory manipulation Dialogs

d. Log manipulation Dialog

6.3.1 User Manipulation dialogs

Under this subject three dialogs are explained namely *Existing Users*, *Add User* and *Edit User*. The operation of these dialogs will be explained in this section.

6.3.1.1 Existing Users

This dialog displays already registered users in a list. It can select and delete a user with **Delete** button. It invokes an *Add User* dialog with **Add** button. It also invokes *Edit User* dialog to change user credentials with **Edit** button. This dialog is shown in fig 6.3.

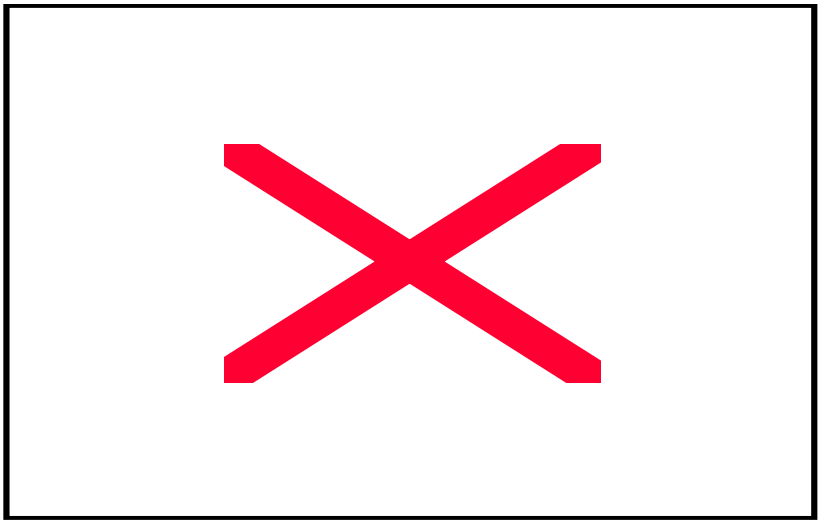

Fig 6.3-Existing Users Dialog

A user account that has been expired is automatically displayed as (Expired). An expired account cannot be used by IIS. The manger has the option to either delete the expired account or update its expiry date.

6.3.1.2 Add User

In this dialog user name, user password and remarks about the account are added for creating new account. Duplicate username or blank password can not be added for any account. Fig 6.4 shows this dialog:

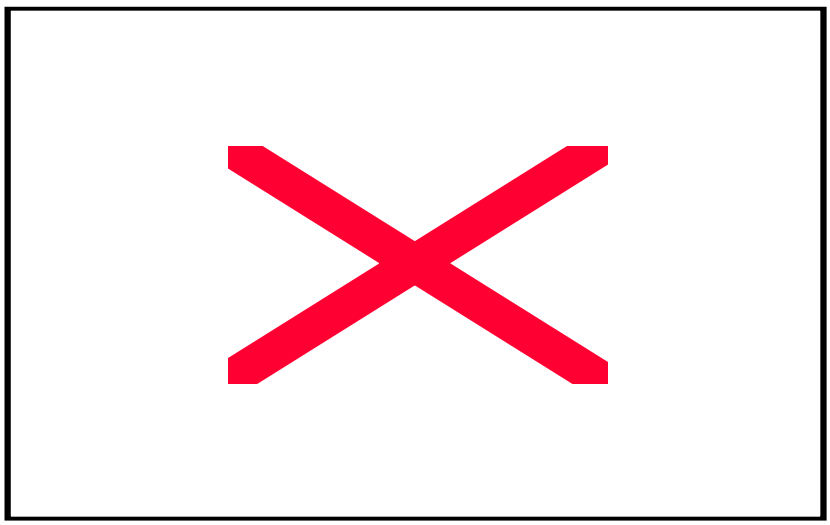

Fig 6.4-Add User Dialog

Once an account is successful created, the expiry date is automatically set after one month the account manager (AM) can change this expiry date from *Edit User* dialog.

6.3.1.3 Edit User

This dialog can change any of the added user credentials in Add User dialog (Fig 6.4). It can further change the expiry date of the account with the help of an elegant calendar control. The expiry date is shown in two (long/short) date formats. Fig 6.5 shows the dialog.

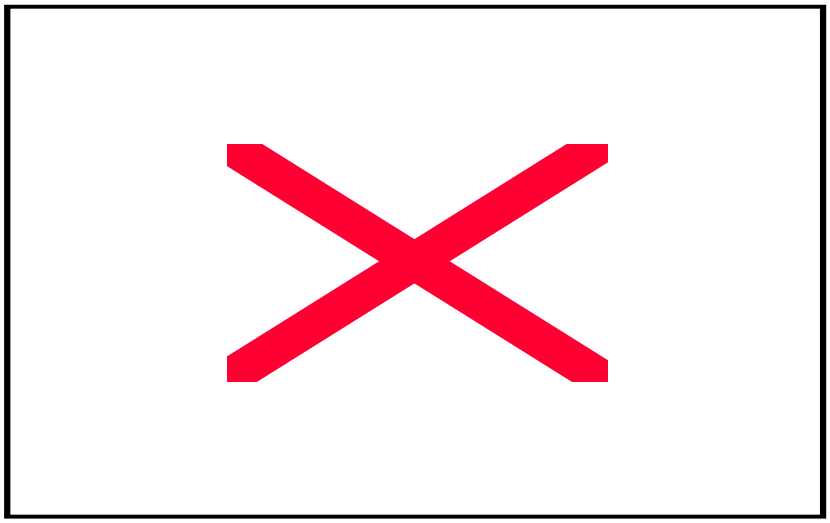

Fig 6.5-Edit User Dialog

6.3.2 Group Manipulation Dialogs

Two dialogs fig 6.6 to fig 6.8 are discussed in this section. When **Group Existing** button of main dialog (fig 6.1) is pressed a dialog for displaying the registered groups is invoked. This dialog can delete the selected group by pressing **Delete** button. After pressing the '**Delete**' button, the system reconfirms the deletion process. This dialog has two more buttons namely '**Add**' and '**Edit**', which invoke another dialog either to add a new group or to change the members of selected group by selection from the lists. Dialog is shown in fig 6.6

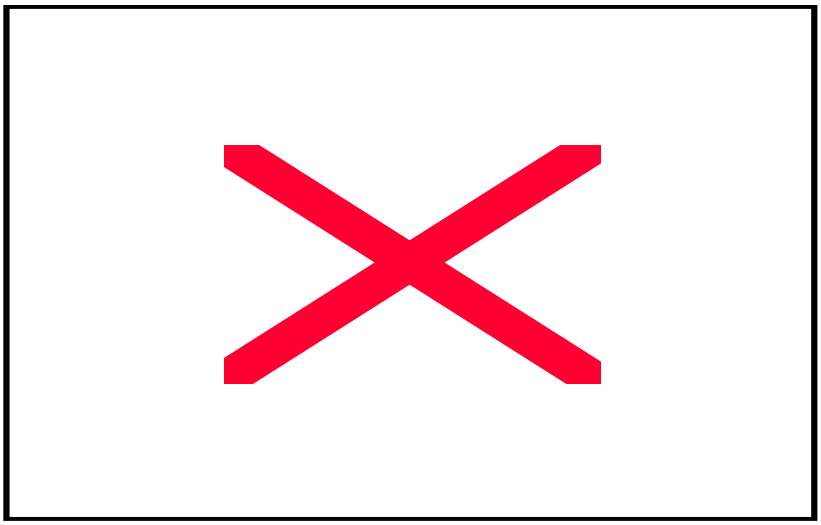

Fig 6.6-Existing Group Dialog

Dialog invoked by either Add or Edit button is the same. However, it displays either of the two modes namely *'Add Mode'* or *'Edit Mode'*. In both the modes, behavior of the dialog is entirely different. Fig 6.7 and fig 6.8 show the dialog in '*Edit'* and *'Add'* modes respectively.

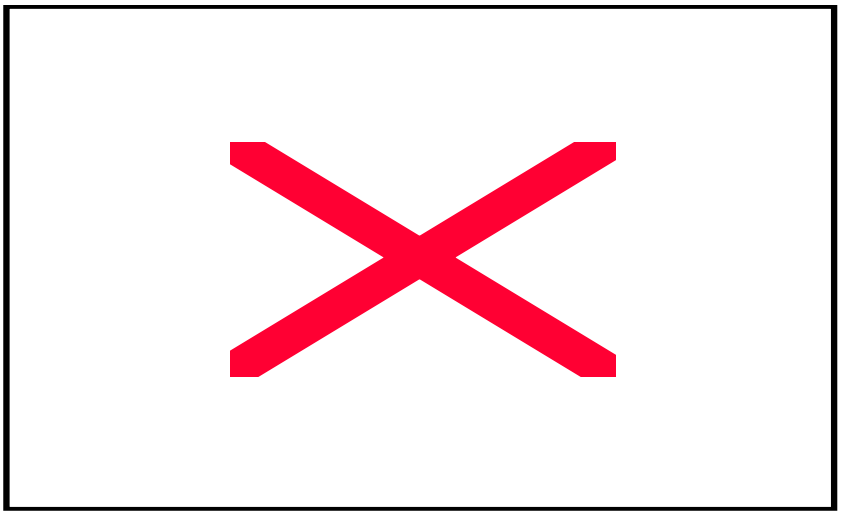

Fig 6.7-(Edit mode) Group Dialog

The dialog in 'Add' mode (fig 6.8) can be invoked from two buttons. Firstly from **Group Add** button of main dialog of fig 6.6 and secondly from the **Add** button in the Edit mode in fig 6.7

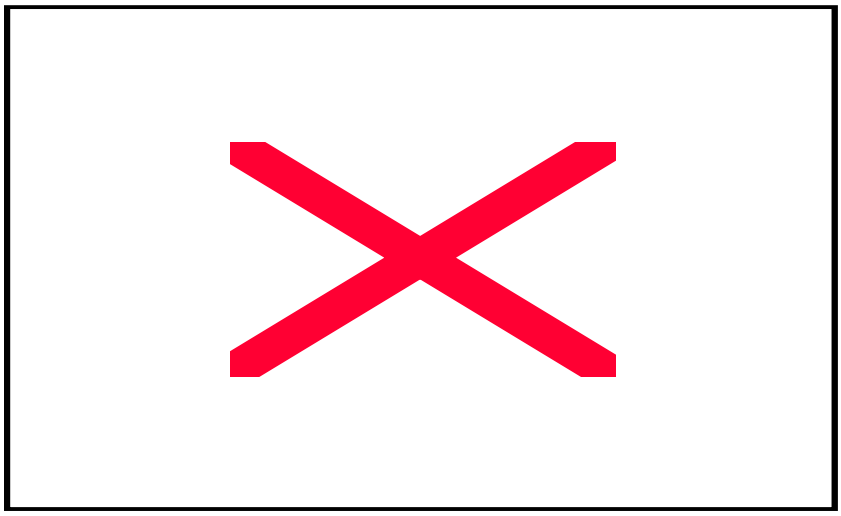

Fig 6.8-(Add mode) Group Dialog

In 'Add' mode, all the buttons and list boxes are disabled and only addition of a new group name in 'group-box' is possible. Protection has been made to avoid duplicate and blank entries for a new group.

6.3.3 Directory Manipulation Dialogs

In IAM, any of the NTFS or FAT directories can be registered and different protection methods can be applied on such registered directories. All such dialogs that are related to directories and their protection methods are grouped n this section.

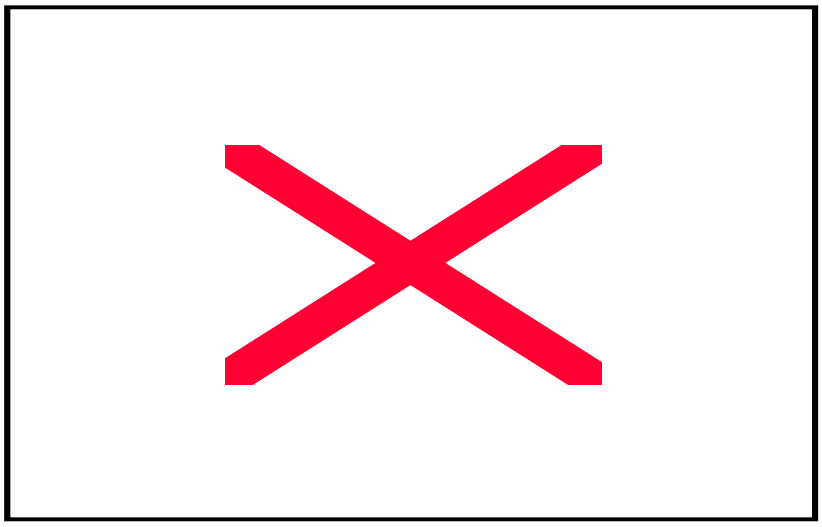

Fig 6.9-Protected Directories Dialog

When **'Customize Directory'** button on main dialog (fig 6.1) is pressed a dialog as shown in fig 6.9 is invoked that displays all the registered directories. On this dialog, **Delete** button can be used to delete any selected directory in the list. **Add** button invokes another dialog as shown in fig 6.10 to add a directory (folder) in the list of protected directories.

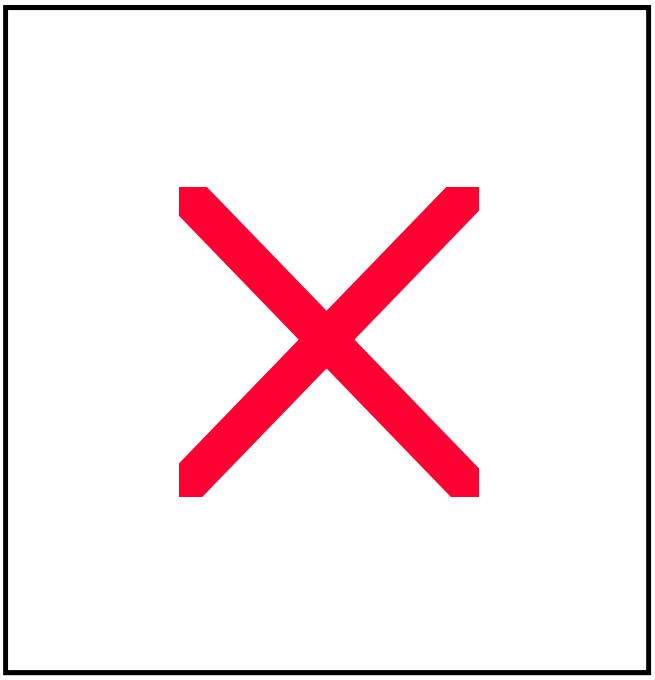

Fig 6.10-Browse For Folder Dialog

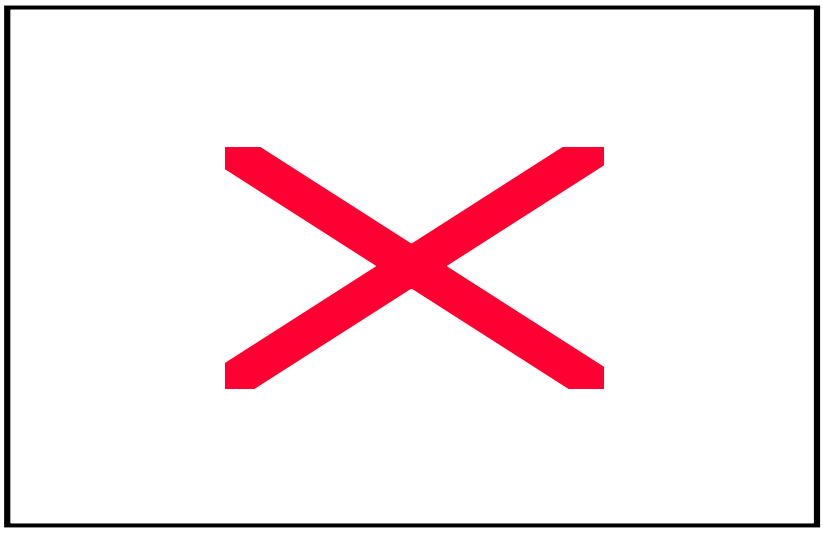

Fig 6.11-Edit Directory Dialog

The **Edit** button on protected directories dialog of fig 6.9 invokes another dialog to display the active protection methods for the selected directory. This dialog is shown in fig 6.11: Possible protection methods by which, a directory can be authenticated are four as enumerated below

a. By IP address

b. By referrer URL

c. By internal database

d. By mapping an NT user to the directory

A protection method is not active unless it is checked. The buttons for changing the corresponding method are displayed only if a method is active. Each button of the corresponding protection method invokes another dialog for handling the required job.

6.3.3.1 Setting Up IP Addresses

An IP address can be registered for a particular directory as permitted or not permitted one. Fig 6.12 shows the dialog for adding a new IP address.

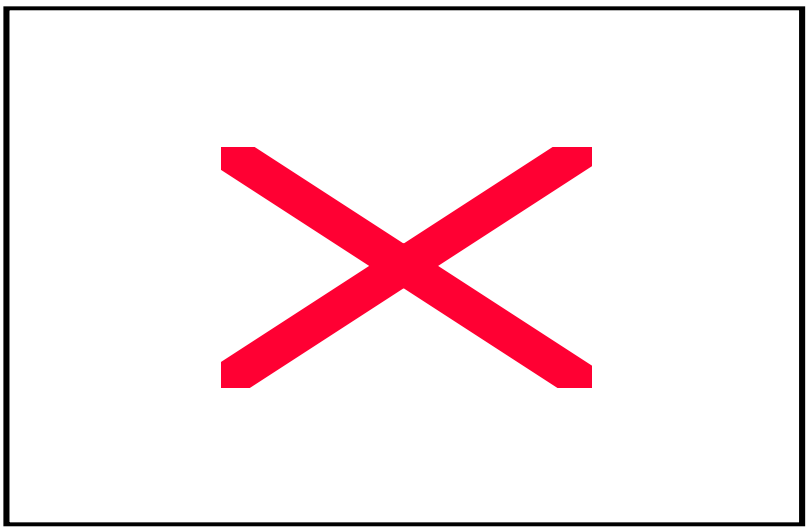

Fig 6.12-Add IP Address Dialog

The permission for an already entered IP address can be changed by pressing **Edit IP** button of fig 6.11. For changing IP permission dialog of fig 6.13 is opened.

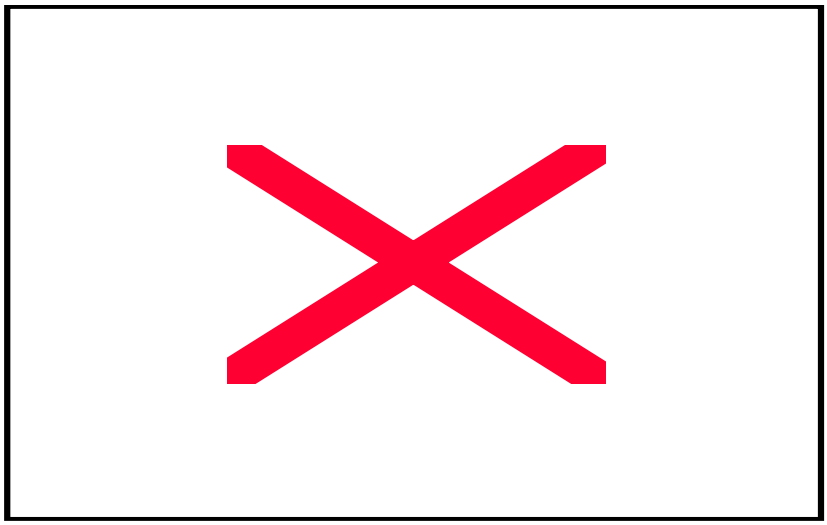

Fig 6.13-Edit IP Address Dialog

In the dialog of fig 6.13, the '**Add**' button moves an IP address from *'Not Permitted'* list to *'Permitted'* list whereas the '**Remove**' button does vice versa. Delete button removes the selected IP address from either of the list.

6.3.3.2 Setting Up Referrer URLs

A URL can be added for a particular directory that is permitted to refer a client browser from itself. A user that has been referred from registered URL is considered valid and not authenticated by the IIS for any other protection method.

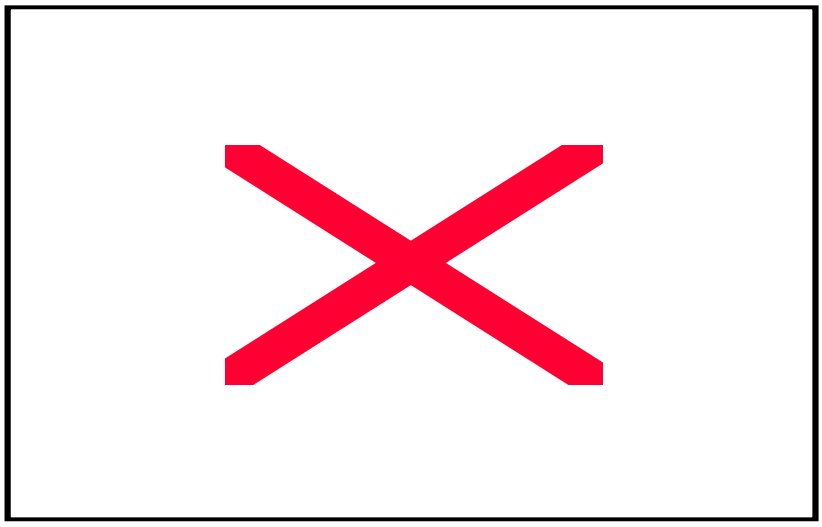

Fig 6.14-Add Delete permitted URL Dialog

Add button adds the entered URL in the list of permitted URLs. **Delete** button deletes the selected URL in the list. Dialog for performing the task shown in fig 6.14.

6.3.3.3 Assigning Registered Groups & Users To The Selected Directory.

Another protection method is to assign already registered groups or users to the selected directory. Fig 6.15 shows the dialog that assigns the groups to the selected directory. If a group is assigned to a directory then all users that are member of that group are automatically assigned to the directory.

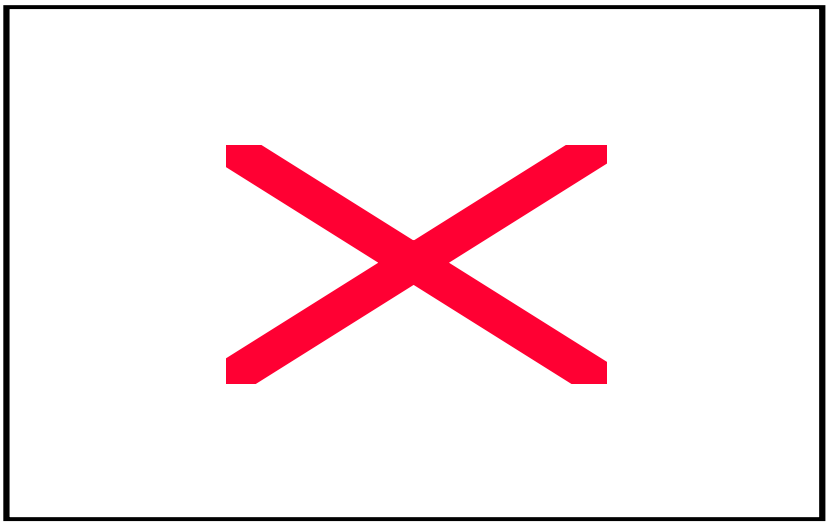

Fig 6.15-Edit Group Directory Dialog

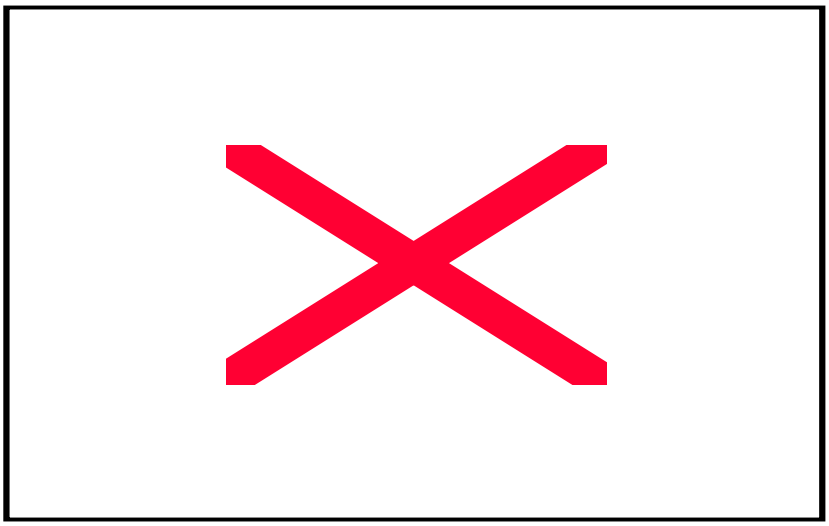

Fig 6.16-Edit User Directory Dialog

Similarly, users can also be assigned to any directory as permitted users. Fig 6.16 shows the dialog for changing the Internet user permissions.

6.3.3.4 Mapping A Directory To NT User

Any directory can be assigned a number of NT users created in ACL of the NT Web server. The advantage of mapped internal user to the directory is to provide access of the NT server to Internet user during authentication process with mapped user, when Internet user is not registered in the internal ACL of the server. Fig 6.17 shows the dialog that maps the directory to NT user.

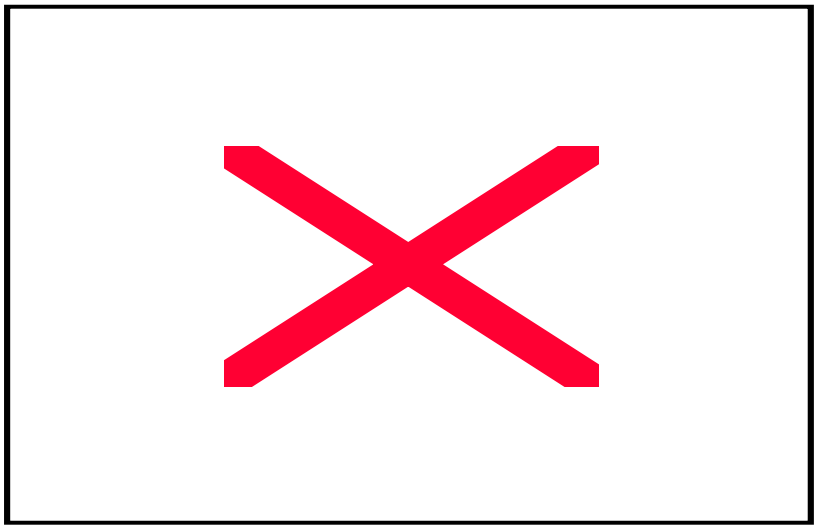

Fig 6.17-MapTo NT User Dialog

6.3.4 Log Manipulation Dialog

Logging of events provides very valuable information to the account manager for better management of the Web site. The system provided in the software gives log of such events those have not been provided by IIS by default. *Seven* events can be logged. There are *four* permanent events namely *'Date / time', 'Resourse', 'RemoteIP'* and *Status.* However, *three* events are selectable namely *'Unmapped remote users', 'Referrer URLs'* and *'User agent'*. Fig 6.18 shows the dialog for logging events. There are different selectable options in the dialog e.g. ' Logging Options' and ' Automatic Clear Log Options'.

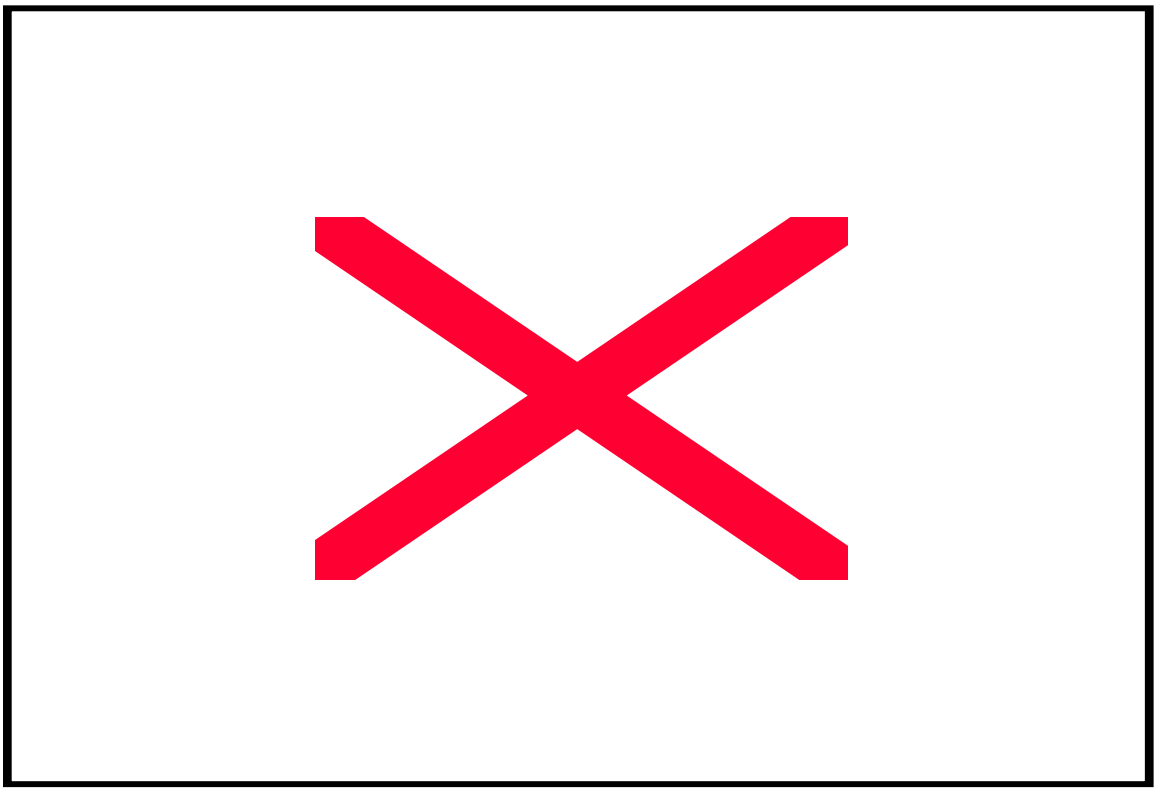

Fig 6.18 -Logging Record Dialog

a. **Logging Options: -** If **Enable** is checked only then the filter will log the events. Moreover, out of three check boxes, only those events will be logged that are checked.

b. **Automatic Clear Log Options: -** If **Enable** is checked only then log is cleared automatically with further three options to clear daily, after three days or after a week

Log can be cleared manually any time by pressing the '**Clear Log**' button. The dialog can be closed by pressing the '**Done**' button. As the **Done** button is pressed a message box appears asking whether to save the settings for the options for later us or otherwise. If '**Yes**' is pressed then all the settings for the options will be saved in the database and when the dialog will be opened next time, the same environment will be available to the user.

6.4 Summary

This chapter has provided an elaborate user guide. It provided the installation procedure of the filter and IAM and later explained the working of each dialog of the IAM. Discussion of IAM has been divided into four main sections according to the model of the IAM interface. The four subsections are regarding manipulations of users, groups, directories and logging events.

Chapter 7
Conclusion

7.1 Introduction

This chapter firstly evaluates the software by discussing its capabilities. Secondly, methodology adopted for software debugging and testing is explained. Thirdly, possible future enhancements are enumerated.

7.2 System Evaluation

As the custom authentication and logging system has been developed to improve on default working of IIS, so it must provide extra facilities to the administrator and allow him to perform tasks that he cannot do in the IIS. The developed system provides the following capabilities not handled by the default IIS system.

a. The custom software built does not depend on the NTFS based security and ACL of NT but can work and protect directories both on NTFS and FAT file formats.

b. New system does not use NTFS ACL's but instead it keeps its own independent database that reduces the burden on the NT database. Since the database is custom built, it can be altered according to the specific requirements to accommodate more information when required.

c. Special emphasis has been given to make interface easy and familiar to the administrators who have prior knowledge of the NT operating system.

d. Filters generally reduce the speed of request processing of the IIS as they perform database hits. A *caching* mechanism in the form *of circular double link-list* has been developed to store the 100 most recently authenticated credentials of directories. If credentials are more than 100 in memory then least recently used credentials will be removed from the memory, so there is

no over burden on the memory. This method speeds up the verification of request and reduces database hits.

e. In case, there is requirement to allow access of some NTFS directory to remote users on Internet, that directory can be accessed by mapping remote username and password to the NT Web server's ACL username and password.

f. Software is capable of blocking any IP address on the Internet, moreover, a URL of permitted referrer Web site can be added for any directory. User being referred from permitted Web site will be considered valid and shall be permitted to access the protected directory without prompting the user for his credentials.

g. Since the software is providing custom authentication for a Web site, so, to make the process more useful it is also providing such custom logging that is not performed by NT. This enables the administrator to manage the Web site and improve it further as per the user's requirements.

7.3 Software Debugging

Debugging of ISAPI filter is a tricky and time-consuming job. Therefore, developers must understand the mechanism of debugging DLLs before stepping into this area of software development. Since in this project IIS 4.0 is used, debugging applicable to IIS 4.0 is explained next.

There are several ways to establish an environment for debugging server components and Internet Server extension DLLs when using IIS 4.0 or later [5]. If a debugger capable of attaching to a Windows NT process is used, this functionality can be used to debug component or extension. For example, if Visual C++ is used, following steps are adopted:

1. Start the *iisadmin* process. This can be done from the command line with the command *net start iisadmin*. Services dialog box from the Control Panel can also be used to start the *IIS Admin Service*, which will start *iisadmin.*

2. Start *Visual C++*. Point to *Start Debug* on the *Build* menu and click *Attach to Process*.

3. Select the *Show System Process* check box

4. Select the *inetinfo* process from the list and click *OK*.

5. Start the *w3svc* service. This can be done from the command line with the command *net start w3svc*. Services dialog box in the Control Panel can also be used to start the *World Wide Web Publishing Service*.

7.4 Software Testing

Since software is related to network communication, therefore its testing over a network environment is very essential to ensure smooth communication. Moreover filter if does not function properly that can crash the whole NT operating system (OS). Hence, extreme care is required to handle the filters.

For development of the filter, a PC duly loaded with NT Server OS with 128 MB RAM was used over Intranet environment. During development phase, Internet Explorer 4.0 was used on local host (stand alone PC) environment. This environment successfully provided the development of around 80% logic for the *ISAPI filter* whereas 100% development of *IAM* was possible on stand alone PC.

Final testing in debug mode of the filter was done over Intranet environment of computer lab of computer department of MCS. Strategy included the activation of three additional Web sites and then communication of these Web sites was tested over TCP / IP protocols. Debugging with the help of additional Web servers made it possible to fine tune the functioning of ISAPI filter software.

The release version of the software was tested over Intranet environment during normal working of IIS. The OS used were NT Server, NT workstation and Windows 98. The browsers tested were Internet Explorer 3.0, 4.0 and 5.0.

7.5 Future Enhancements

Although the software has tried to cover as many aspects of custom authentication but still advancements and improvements are possible. Possible enhancements in the software are highlighted in the

a. Since this authentication filter uses basic authentication of NT, by default it does not use the secure socket list (SSL) to encrypt the data. This make the data sent venerable to be snatched at any place on the network. In order to use SSL for outgoing data, SSL can be enabled form *the key manger* tool of NT, which can be executed from the Windows NT start button or from MMC. Presently, software does not provide a method to enable the SSL form with in it.

b. Most of the Web sites require that the no of times users refer the Web site that should be counted. This is done to determine the popularity of a particular resource on the site. Although a general hit log is visible in log view, but a hit count corresponding to each protected directory can be maintained and displayed separately for the administrator who can determine popularity of a particular resource.

c. The IAM provides administration of custom authentication from only the server terminal hosting the protected directories and it is not possible to exercise such administration remotely. ASPs can be used to provide remote administration of the protected directories if required and can be included in the software.

Presently the software is carrying out authentication in four different ways namely *by IP address, by IAM database, by referrer* and *mapping Internet user to internal NT user of ACL*. However, these methods can be further enhanced. Some of the suggested enhancements could be:

a. *Time Blocking*: - Users can be given access to specific directory with in certain time frame. This may be required on sites where user load is quite

heavy, so, to prevent the degradation in IIS performance, priorities can be assigned to users by allocating specific time to each. The user would then be able to access the resource only during that allowed period.

b. *Resource Size (Content Length)*: - Users can be assigned specific size limit that is allowed to be accessed by the user. A size greater than they're allowed can be denied.

c. *Server Port*: - The method can be inducted to only accept those requests that arrive from a secure port and reject all the rest.

Similarly, the software logs six (6) types of credentials that can be further increased according to the specific needs. Following additional information can be also be logged.

a. *Content Length*: - That is the number of bytes the script can except to receive from the client.

b. *Content Type*: - That is the type of information supplied in the body of the client.

c. *Server Port*: - that is the TCP / IP port on which, the information was received.

7.6 Summary

This small chapter has been included for the interest of those whom to excel in the field of ISAPI filters. This chapter firstly evaluates the software by discussing its capabilities. Secondly, methodology adopted for software testing is explained. Thirdly, possible future enhancements are enumerated.

APPENDIX A

Table A.1 - List of abbreviations

1.	HTTP	Hyper Text Transfer Protocol
2.	ISAPI	Internet Server Application Programming Interface
3.	CGI	Common Gateway Interface
4.	ASP	Active Server Pages
5.	ECB	EXTENSION_CONTROL_BLOCK
6.	HTML	Hyper Text Mark UP Language
7.	DLL	Dynamic Link Library
8.	ODBC	Open Database Connectivity
9.	URL	Uniform Resource Locator
10.	NTLM	NT challenge / Response
11.	FAT	File Allocation Table
12.	NTFS	New Technology File System
13.	IAM	Internet Account Manager
14.	AM	Account Manager
15.	WWW	World Wide Web
16.	DAO	Data Access Objects
17.	IDE	Integrated Development Environment

Table A.2 - The HTTP status codes

Status Code	**Response Phrase**	**Definition**
200	OK	The request has succeeded.
401	Unauthorized	The request needs user authentication.
403	Forbidden	The server understood the request but refusing to fulfill it.
404	Not Found	The server has not found anything matching the request.
500	Internal Server Error	The server encountered an unexpected condition that prevented it from fulfilling the request.
501	Not Implemented	The server can't fulfill the request. This is the appropriate response when the server does not recognize the request method and can't use it for any resource.

APPENDIX B [7]

1. **HTTP_FILTER_CONTEXT Structure**

```
typedef struct _HTTP_FILTER_CONTEXT
{
        DWORD        cbSize;
//  This is the structure revision level.
        DWORD        Revision;
//  Private context information for the server.
        PVOID        ServerContext;
        DWORD        ulReserved;
//  TRUE if this request is coming over a secure port
        BOOL         fIsSecurePort;
//  A context that can be used by the filter
        PVOID        pFilterContext;
//  Server callbacks
        BOOL (WINAPI * GetServerVariable) (
        struct _HTTP_FILTER_CONTEXT * pfc,
        LPSTR                lpszVariableName,
        LPVOID                lpvBuffer,
        LPDWORD                lpdwSize
        );

        BOOL (WINAPI * AddResponseHeaders) (
         struct _HTTP_FILTER_CONTEXT * pfc,
        LPSTR                lpszHeaders,
        DWORD                dwReserved
        );

         BOOL (WINAPI * WriteClient)  (
```

```
        struct _HTTP_FILTER_CONTEXT * pfc,
        LPVOID                  Buffer,
        LPDWORD                 lpdwBytes,
        DWORD                   dwReserved
        );

        VOID * (WINAPI * AllocMem) (
        struct _HTTP_FILTER_CONTEXT * pfc,
        DWORD                   cbSize,
        DWORD                   dwReserved
        );

        BOOL (WINAPI * ServerSupportFunction) (
        struct _HTTP_FILTER_CONTEXT * pfc,
        enum SF_REQ_TYPE            sfReq,
        PVOID                   pData,
        DWORD                   ul1,
        DWORD                   ul2
        );

} HTTP_FILTER_CONTEXT, *PHTTP_FILTER_CONTEXT;
```

2. **HTTP_FILTER_RAW_DATA Structure**

```
typedef struct _HTTP_FILTER_RAW_DATA
{
   //  This is a pointer to the data for the filter to process.
        PVOID       pvInData;
        DWORD       cbInData;       // Number of valid data bytes
        DWORD       cbInBuffer;     // Total size of buffer
        DWORD       dwReserved;
```

```
} HTTP_FILTER_RAW_DATA, *PHTTP_FILTER_RAW_DATA;
```

3. **HTTP_FILTER_PREPROC_HEADERS Structure**

```
//  This structure is the notification info for when the server is about to
//  process the client headers
typedef struct _HTTP_FILTER_PREPROC_HEADERS
{
        BOOL (WINAPI * GetHeader) (
         struct _HTTP_FILTER_CONTEXT * pfc,
        LPSTR                   lpszName,
        LPVOID                   lpvBuffer,
         LPDWORD                    lpdwSize
         );
//  Replaces this header value to the specified value.  To delete a header,
   //  specified a value of '\0'.
        BOOL (WINAPI * SetHeader) (
         struct _HTTP_FILTER_CONTEXT * pfc,
        LPSTR                   lpszName,
        LPSTR                   lpszValue
         );
//  Adds the specified header and value
        BOOL (WINAPI * AddHeader) (
         struct _HTTP_FILTER_CONTEXT * pfc,
         LPSTR                   lpszName,
         LPSTR                   lpszValue
         );
DWORD HttpStatus;            // New in 4.0, status for SEND_RESPONSE
   DWORD dwReserved;             // New in 4.0

} HTTP_FILTER_PREPROC_HEADERS,
*PHTTP_FILTER_PREPROC_HEADERS;
```

4. **HTTP_FILTER_AUTHENT Structure**

```
typedef struct _HTTP_FILTER_AUTHENT
{
//  Pointer to username and password, empty strings for the anonymous user
//  Client's can overwrite these buffers which are guaranteed to be at
   //  least SF_MAX_USERNAME and SF_MAX_PASSWORD bytes large.
        CHAR * pszUser;
         DWORD  cbUserBuff;
        CHAR * pszPassword;
        DWORD  cbPasswordBuff;
} HTTP_FILTER_AUTHENT, *PHTTP_FILTER_AUTHENT;
```

5. **HTTP_FILTER_URL_MAP Structure**

```
typedef struct _HTTP_FILTER_URL_MAP
{
        const CHAR * pszURL;
        CHAR *       pszPhysicalPath;
        DWORD        cbPathBuff;
} HTTP_FILTER_URL_MAP, *PHTTP_FILTER_URL_MAP;
```

6. **HTTP_FILTER_ACCESS_DENIED Structure**

```
typedef struct _HTTP_FILTER_ACCESS_DENIED
{
        const CHAR * pszURL;           // Requesting URL
         const CHAR * pszPhysicalPath;   // Physical path of resource
        DWORD        dwReason;         // Bitfield of SF_DENIED flags
```

```
} HTTP_FILTER_ACCESS_DENIED,
*PHTTP_FILTER_ACCESS_DENIED;
```

7. **HTTP_FILTER_LOG Structure**

```
typedef struct _HTTP_FILTER_LOG
{
        const CHAR * pszClientHostName;
        const CHAR * pszClientUserName;
        const CHAR * pszServerName;
        const CHAR * pszOperation;
        const CHAR * pszTarget;
        const CHAR * pszParameters;

        DWORD  dwHttpStatus;
        DWORD  dwWin32Status;

        DWORD  dwBytesSent;           // IIS 4.0 and later
        DWORD  dwBytesRecvd;          // IIS 4.0 and later
        DWORD  msTimeForProcessing;   // IIS 4.0 and later

} HTTP_FILTER_LOG, *PHTTP_FILTER_LOG;
```

Bibliography

[1] Alan R. Carter, “MCSE Study Guide on Windows NT Server 4.0” , IDG Books, 1998.

[2] Kevin Clements, Chris Wuestefeld and Jeffrey Trent, “Inside ISAPI” ,New Riders Publications, 1997.

[3] Matthew Strebe and Charles Perkins, “ MCSE study Guide on Internet Information Server 4", BPB publications, 1999.

[4] On line help, Microsoft NT Option Pack 4.

[5] Online help, Microsoft Developers Network(MSDN) Library, October 99

[6] Stephen Genusa, Bobby Addison Jr and Allen Clerk, “Special Edition Using ISAPI”, QUE series, 1997.

[7] Httpfilt.h, Microsoft header file provided in Visual C++ version 6.0

[8] Catherine Ricardo, "Database Systems Principles, Design and Implementation", Maxwell Macmillan, 1990.

[9] Roger Jennings, "Database Developer's Guide with Visual C++ 4, Second Edition", SAMS, April 1996.