Secure/Non-secure page redirector documentation

Budapest, June 16, 1999.

Peter Verhás

Introduction

This documentation describes a tool, which was developed during a consultation project by XXXX for YYYY. Therefore the resulting software is a tool usable as it is and it is not a product of XXXX. This means that the compiled binary together with the source was given to YYYY as knowledge without any product guarantees.

YYYY extranet Web server, which is currently under development has secure and non secure pages[1]. The web server handles all pages that pass secure data securely. The secure and non-secure pages reference each other using relative URLs. This means that a non-secure page referencing in a link presents an URL to the browser, which is non-secure. On the other hand the web server serves the page only through secure channel. This results an error response from the web server.

On the other hand: when a secure page references a non-secure page the non-secure page will be handled via secure channel. This does not cause any error, but results slower service consuming more resource. The web server serves the non-secure pages via secure as well as non-secure web server.

The official Microsoft Knowledge Base article suggests using absolute URLs whenever a secure/non-secure page link is needed. This is inconvenient. This prevents the developers to test the application in a non-secure web server and redeployment of the pages on a different server needs the alteration of all these absolute URLs because they contain the machine name.

The solution is a tool in the form of an ISAPI filter. This automatically detects secure and non-secure page requests and whenever the http request type differs from the type of the page, it responds to the client browser with a redirect command targeting the same page with the proper protocol (http or https).

Kit content

The installation kit contains the binary code of the program, documentation, source of the program and sample files.

The files in the installation ki:

securedi.doc	Program documentation
debug.c	Source C file
debug.h	Source h file
registry.c	Source C file
registry.h	Source h file
securedi.c	Source C file
securedi.def	Source def file
securedi.dll	Compiled binary
nsposta.asp	Test ASP page
nstsget.asp	Test ASP page
nstspost.asp	Test ASP page
sposta.asp	Test ASP page

[1] In this document we call a web page *secure* if the web server processes the page only through secure channel. The web server in response to non-secure requests processes *non-secure* pages. However non-secure pages are processed by the IIS web server in response to secure requests as well.

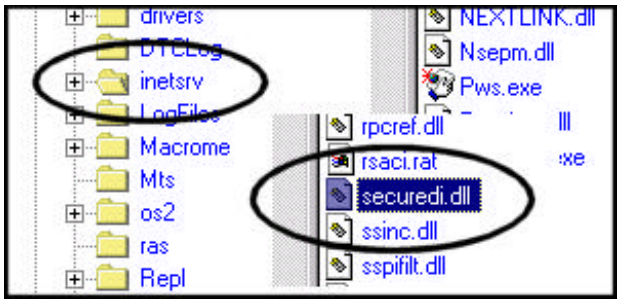
stnsget.asp	Test ASP page
stnspost.asp	Test ASP page
rules.txt	Sample rule file

Installation

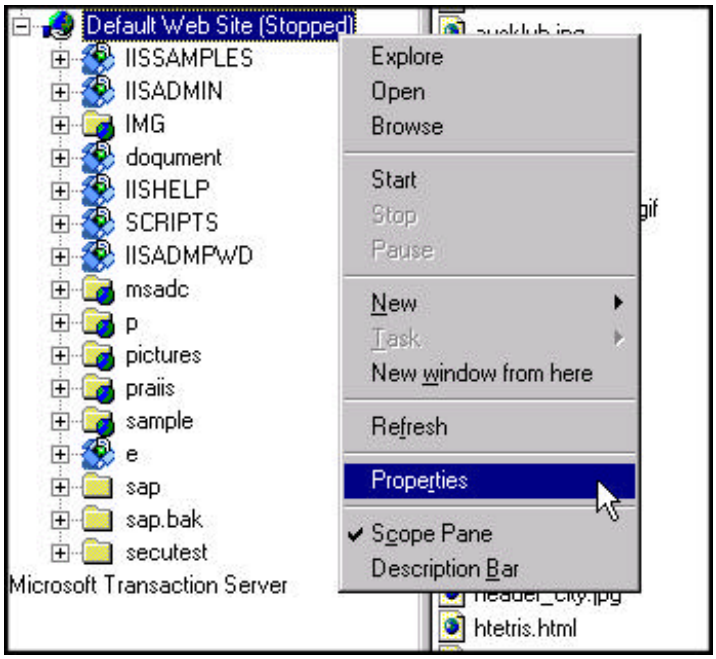
On the web server machine stop the IIS Admin Service using the Service Manager Control Applet or issuing the command

```
net stop "IIS Admin Service" /yes
```

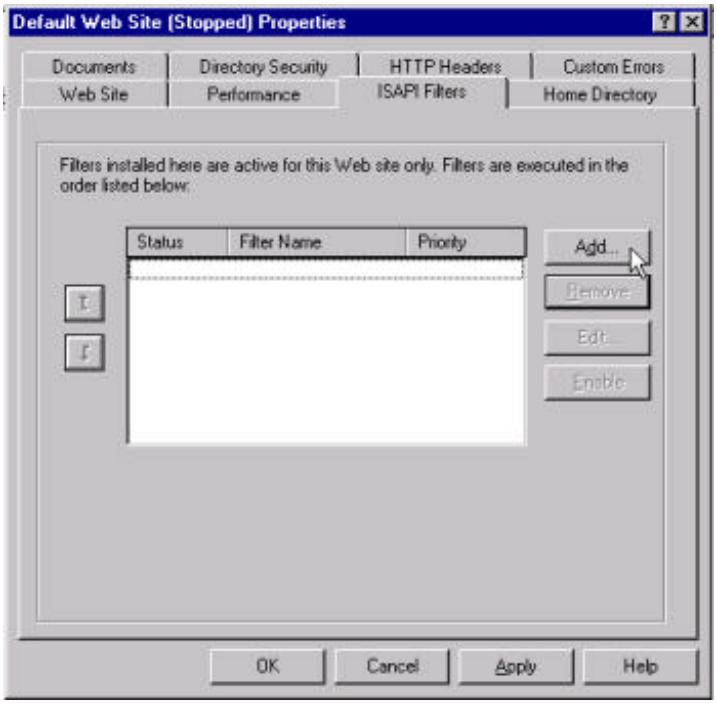
Copy the dll to the web server into a directory. Usually `WINNT\System32\Inetsrv` is a good place.



Configure IIS to use this filter. You can do this using the Microsoft Management Console. Select the Web server you want to use the filter and using the right mouse button select the Web server properties.

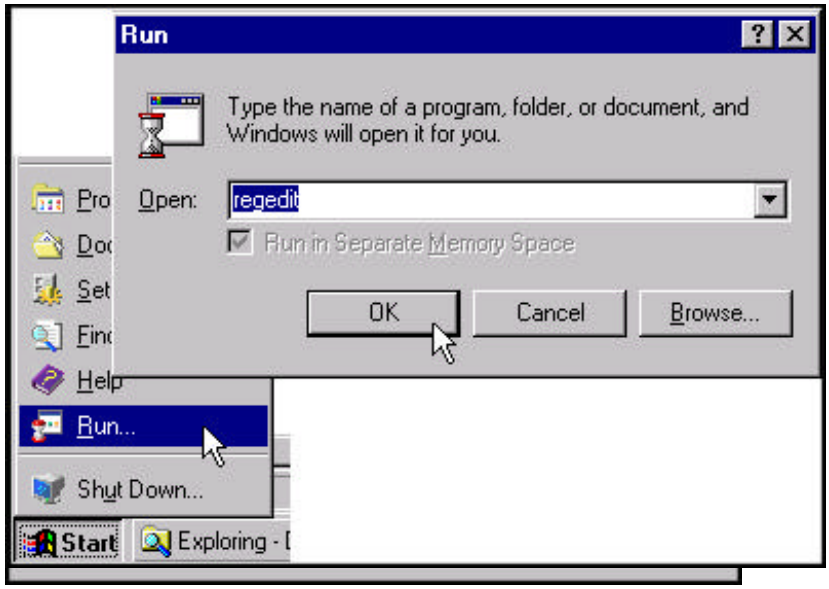


On the property sheet select the ISAPI Filters tab and press the button add.

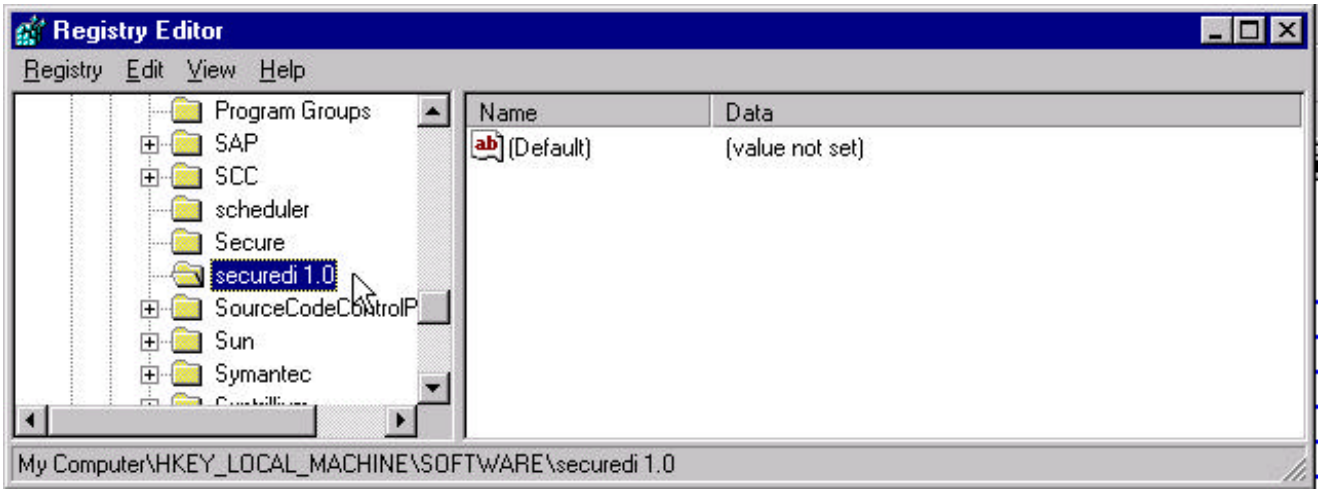


Specify a name for the filter that helps you later remember the function of the filter and specify the full path to the DLL file. Press OK. The filter name appears in the filter list with priority *Unknown*. Press OK again on the property sheet.

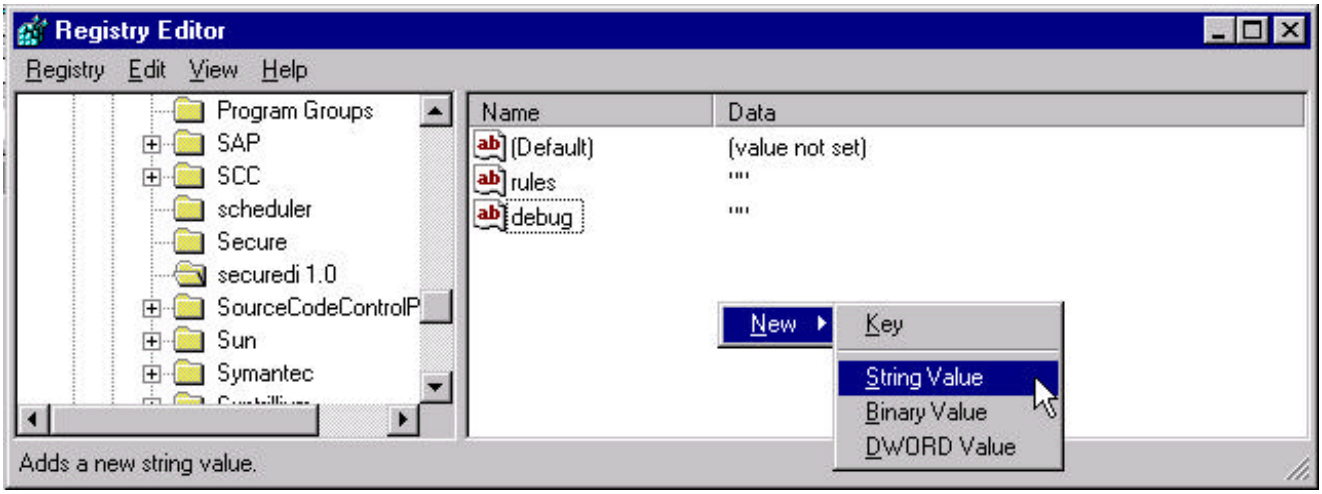
Edit the registry to define the configuration file and the debug information file. To this start the registry editor from the start menu start the program `regedit`.



Create the key [HKEY_LOCAL_MACHINE\SOFTWARE\securedi 1.0] and create two string values under this key pressing the right mouse button and selecting new string.



The names of the string values are "rules" and "debug". The string value "rules" should contain the file name of the file containing the secure pages. This file name should specify



the full path name including the drive letter and the directory. The file should be placed on a local disk and the built-in user SYSTEM should be able to read it.

The string value "debug" should contain the name of the file that the filter will generate listing its activities. This file name should contain the full path including the drive letter and the directory.

Create the rules file using notepad. You can name it as you like, but since it contains text data the obvious extension you may choose is `txt`. This file should contain all the URLs, which are to be served secure. The URLs in the file should NOT contain the protocol and the machine name, only the local virtual path and file name. The following URLs are valid examples:

```
/myvirtual/directory/index.html
/index.html
/default.asp
/scripts/submit.pl
```

The following examples contain invalid lines:

```
http://www.mycom.com
myvirtual/directory/index.html
/myvirtual/*
```

Contains the protocol and the machine name
It does not contain the leading /
Wild characters are NOT allowed!

Start the web server using the Service Manager Control Applet starting the service World Wide Web Publishing Service (usually this is the last service listed) or
Edit the registry and edit the configuration file. Start the web server using

the Service Manager Control Applet and starting the World Wide Web Publishing Service (usually the last service listed), or issuing the command

```
net start "World Wide Web Publishing Service" /yes
```

Use your browser to request some pages and have a look at the generated debug file. Check that the messages are correct.

Stop the web server again. Edit the registry specifying an empty debug file name. Start the web server.

Whenever you change the rules file or change any of the two registry values listed above you have to start and stop the web server using the Service Manager Control Applet or issuing the commands:

```
net stop "IIS Admin Service" /yes
net start "World Wide Web Publishing Service" /yes
```

Configuration

The configuration of the program is done using the registry editor and notepad. The registry entries the program uses are

[HKEY_LOCAL_MACHINE\SOFTWARE\securedi 1.0\rules]

and

[HKEY_LOCAL_MACHINE\SOFTWARE\securedi 1.0\debug]

The string value "rules" should specify the full path of the rule file. The string value "debug" should specify the file name where the filter should print its debug information. You can use the debug file to check the filter activities whenever the system behaves erroneous. The filter writes its entire activity log into this file.

Note however that this feature is for trouble shooting purposes only. The debug file generation is resource consuming and may severely impact system performance. Different http serving threads under heavy load may result in garbled debug file. Use debugging only in test environment testing the server with single client.

To switch off the debug file-creation delete the string value "debug" or specify an empty name. After changing the registry value restart the web server.

Create the rules file using notepad. You can name it as you like, but since it contains text data the obvious extension you may choose is `txt`. This file should contain all the URLs, which are to be served secure. The URLs in the file should NOT contain the protocol and the machine name, only the local virtual path and file name.

Whenever you change the rules file or change any of the two registry values listed above you have to start and stop the web server using the Service Manager Control Applet or issuing the commands:

```
net stop "IIS Admin Service" /yes
net start "World Wide Web Publishing Service" /yes
```

Note that the rule file may contain invalid lines without generating error. Any line which is invalid consumes resource but does not generate any other malfunctions.

Checking and testing the configuration

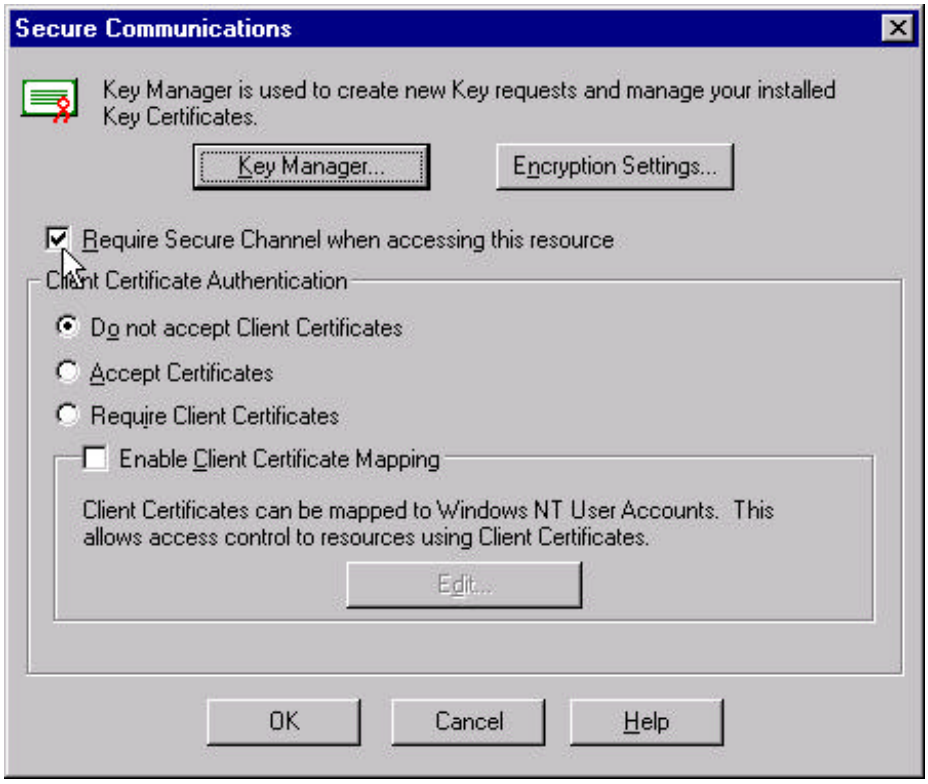
Copy the six ASP files supplied in the kit to the virtual root directory of the web server. Assume that the server name is `www.mycompany.com`. In this case the six test web pages will have to URL:

```
http://www.mycompany.com/stnsget.asp
http://www. mycompany.com/nstsget.asp
http://www. mycompany.com/stnspost.asp
http://www. mycompany.com/nstspost.asp
http://www. mycompany.com/sposta.asp
http://www. mycompany.com/nsposta.asp
```

The characters of the names mean:

`stns`	Secure to non-secure. The page is secure and the form target is non-secure.
`get`	The page contains a form and the method of the form is GET.
`post`	The page contains a form and the method of the form is post.
`sposta`	Secure page, post (and get) accept.
`nsposta`	Non-secure page, post (and get) accept.

Start the Microsoft Management Console and change the security setting of the files `stnsget.asp`, `stnspost.asp` and `sposta.asp`.



You can do this individually selecting each of the files, clicking the right mouse button, selecting properties and pressing the "Edit" button in the middle frame titled "Secure Communications". After pressing this button you get the dialog above, where you have to check the checkbox "Require Secure Channel when accessing this resource" as pointed by the mouse in the picture above. You have to do this for all the three files. Be sure that this check box is NOT checked for the files `nstsget.asp`, `nstspost.asp` and `nsposta.asp`.

You can do it only if you have installed some certificate on your server before. On the topic how to install a certificate on the server see the documentation of the product Microsoft Internet Information Server 4.0

After this setting the URLs of the pages become:

```
https://www. mycompany.com/stnsget.asp
```

```
http://www. mycompany.com/nstsget.asp
https://www. mycompany.com/stnspost.asp
http://www. mycompany.com/nstspost.asp
https://www. mycompany.com/sposta.asp
http://www. mycompany.com/nsposta.asp
```

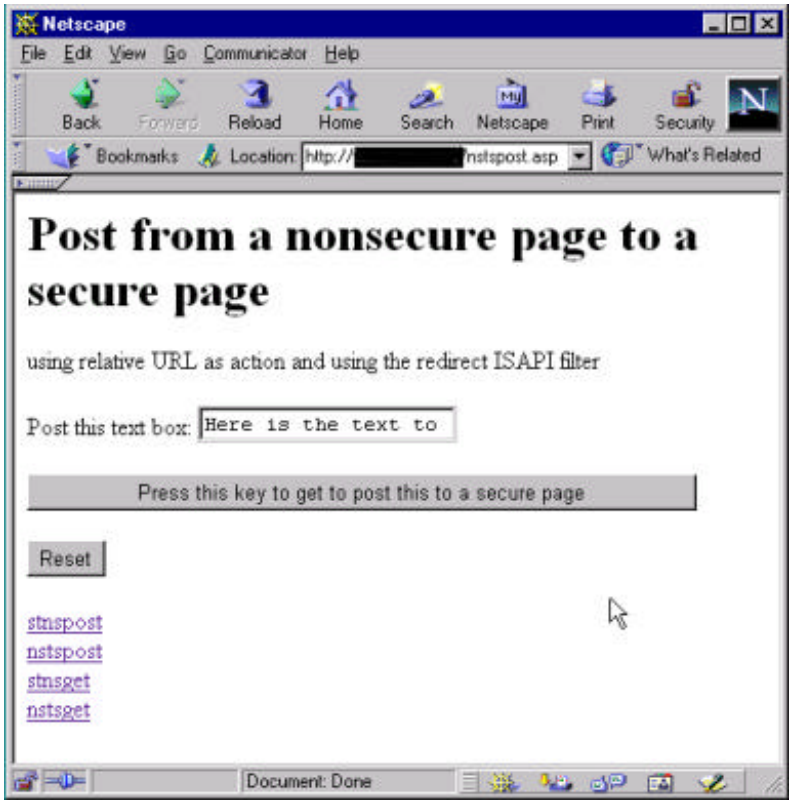
The pages with name starting with the letter ‘s’ are secure pages and can be accessed via secure channel. The other pages reference these pages using relative link in the href link and in the form action fields. The same is true to other way around. The pages starting with the letter ‘hs’ are non-secure pages and the secure pages reference them using relative reference, and therefore not changing the channel from secure to non-secure r the other way around.

Copy the content of the file `rules.txt` supplied in the installation kit into the configured rules file. The rules file may contain other lines as well but be sure that the following lines do **NOT** present in the rules file:

```
/nstsget.asp
/nstspost.asp
/nsposta.asp
```

as these file should be accessed via non-secure channel. This way the rules file tell the filter that the files `nstsget.asp`, `nstspost.asp` and `nsposta.asp` are accessed via a secure channel and the files `nstsget.asp`, `nstspost.asp` and `nsposta.asp` are accessed via normal, non-secure channel.

Open the URL `http://www.mycompany.com/nstspost.asp`



Press the button “Press this key to get to post this to a secure page”. This will create a POST http request to the URL `http://www.mycompany.com/sposta.asp`. The filter will see that the result is to be secure and send back a redirect and thus finally the browser opens the page `http://www.mycompany.com/sposta.asp`

Navigate through the pages and test the automated switching from secure to non-secure pages and the other way around.

Note that POST operations do NOT work with some browsers when redirected from one page to another. This is a functional limitation of the browsers. On the other hand this limitation will prevent the web architect to generate insecure page links. See details in the frame below.

> *When a form is included in a page, which is downloaded via normal http the result of the form is sent to the web server via http without encryption in case the form action is specified as a relative URL. Redirecting this POST http request to a secure page does not increase security, because when the redirection of the request is performed the data is already sent through the network in a non-secure form.*
>
> *Therefore such a redirection even decreases the security because it makes the user feel secure while they are not secure.*
>
> *When a form is was downloaded securely and the resulting POST http request is redirected to a non-secure page the browser asks the user whether to perform the operation. This is a mandatory feature of most browsers, because the form was posted by the user to be transmitted via secure channel, and posting via non-secure channel was requested after the user already have pressed the submit button.*
>
> ***When designing secure and non-secure page transitions it should only be done using href links and never following form actions.***
>
> *The page showing the form of secure data should be transmitted securely using this filter.*

How the program works

This filter checks all incoming connections and compares the URL against a table listing all secure URLs. If the URL needs authentication and the connection is not secure it sends a redirect to the secure page. If this is a secure channel, but it does not need a secure connection it sends back a redirect to a non-secure page.

This ISAPI filter hooks itself into the PREPROC_HEADER point. This is the very first point where a http request can be accessed by a filter. This filter hooks itself both for secure and non-secure connections. This is done in the `GetFilterVersion` function.

```
BOOL WINAPI GetFilterVersion(HTTP_FILTER_VERSION * pVer){
  FILE *fp;
  char *s;
  int ch;
  int iCount;

  pVer->dwFilterVersion = HTTP_FILTER_REVISION;

  pVer->dwFlags = SF_NOTIFY_ORDER_LOW       |
                  SF_NOTIFY_SECURE_PORT     |
                  SF_NOTIFY_NONSECURE_PORT  |
                  SF_NOTIFY_PREPROC_HEADERS |
                  0
                  ;

  strcpy( pVer->lpszFilterDesc, "Securedi URL Filter 1.0" );

.. .. .. //read config and return TRUE ...
```

The priority of the filter is low. The IIS Web server calls this function after starting itself after the DLL is loaded. This function sets the appropriate parameters informing the web server when to call the filter function and reads the configuration parameters from the registry and from a text file.

The web server calls the function `HttpFilterProc` for each request it serves. The functions retrieves the URL of the request using the call-back functions the web server supports and decides if the page the URL is mapped to is secure or is non-secure. If the request type does not match the page "secureness" then it sends a redirect to the client. The actual code fragment that does this:

```
if( pfc->fIsSecurePort && ! fSecureUrl ){
      // here we have to redirec to the same URL with non-secure port
      sprintf(szBuffer,"HTTP/1.0 301 Permanently moved\nLocation: http://%s%s\n\n",szServerName,szOriginalUrl);
      DebugMessage("Redirecting %s to %s",szUrl,szBuffer);
      cbBuffer = strlen(szBuffer);
```

```
    pfc->WriteClient(pfc,szBuffer,&cbBuffer,HSE_IO_SYNC);
    return SF_STATUS_REQ_FINISHED;
    }
  if( ! pfc->fIsSecurePort && fSecureUrl ){
    // here we have to redirect to the same URL with secure port
    sprintf(szBuffer,"HTTP/1.0 301 Permanently moved\nLocation: https://%s%s\n\n",szServerName,szOriginalUrl);
    DebugMessage("Redirecting %s to %s",szUrl,szBuffer);
    cbBuffer = strlen(szBuffer);
    pfc->WriteClient(pfc,szBuffer,&cbBuffer,HSE_IO_SYNC);
    return SF_STATUS_REQ_FINISHED;
    }
```

When the page is redirected the return value of the function is `SF_STATUS_REQ_FINISHED` telling the web server that the request was fully served by the filter and the connection to the client can be closed. If the page needs no redirection the filter is transparent and returns `SF_STATUS_REQ_NEXT_NOTIFICATION` telling the server to process the request further.

The redirection location is automatically created to include the protocol (http or https) and to include the server name. The server name is created from the result of the function call to `GetServerVariable`.

Compilation

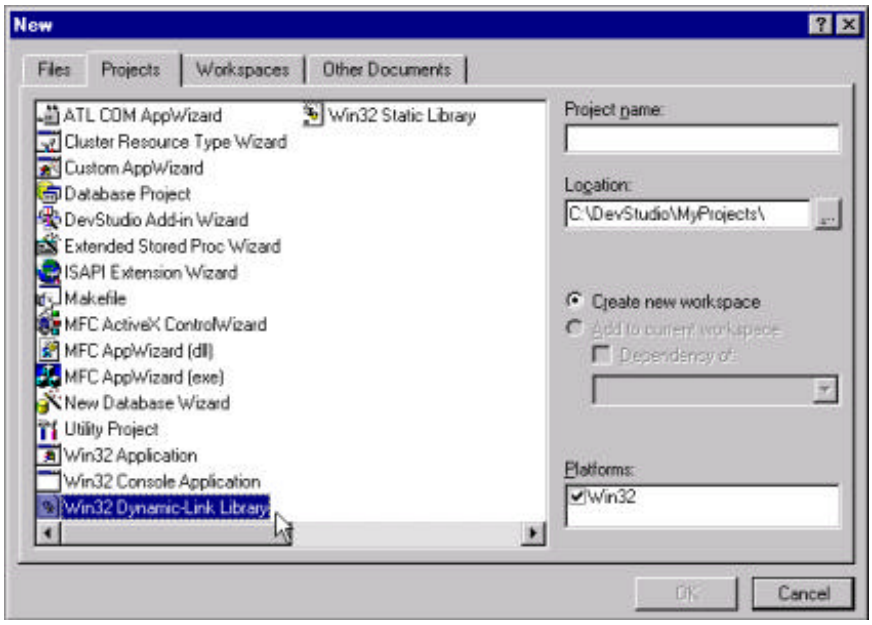
To compile the program and to generate the binary executable you need a C language compiler. During the consultation project we used Microsoft Visual C++ V6.0. This chapter describes, how to compile the source code using this tool. The compilation process using a different compiler may be different, but should be similar.

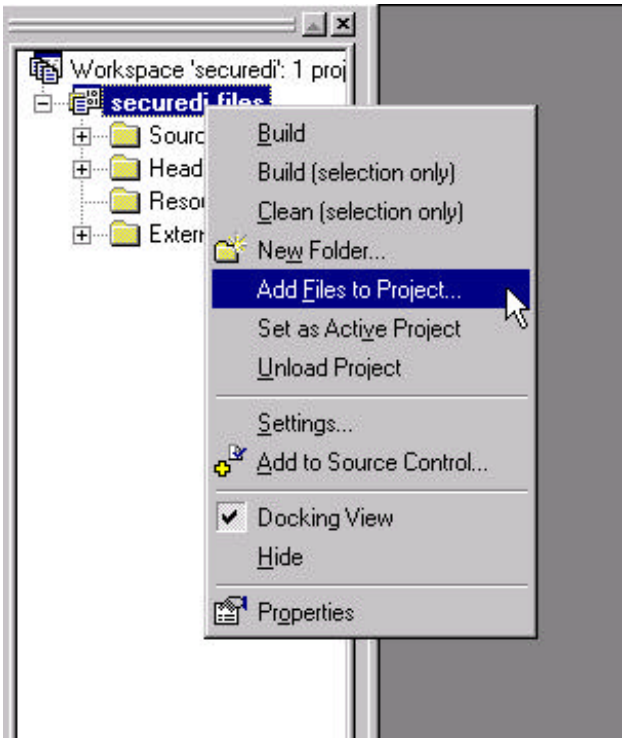
The source files are

- debug.c
- debug.h
- registry.c
- registry.h
- securedi.c
- securedi.def

The *.c file contain the source code, the *.h files contain definitions for the functions implemented in the *.c files. The *.def file define the exported functions that the compiler should put into the exported function table of the resulting DLL file.

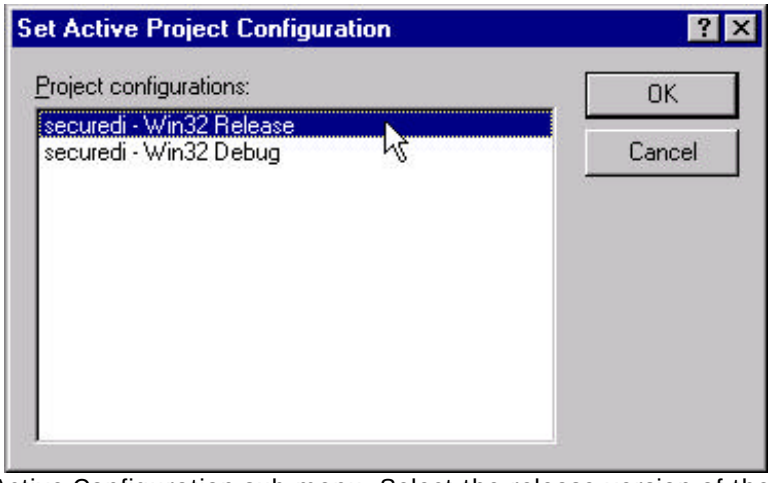
Create a project using the Visual C++ V6.0 integrated development environment, telling the compiler that you are generating an empty DLL project.





Using the right mouse button click on the project and select "Add Files to Project".

Select the *.c, *h and the securedi.def files to add to the project. Select the Build menu,



and the Set Active Configuration sub menu. Select the release version of the project.

Compile the project pressing the key F7. The compiler generates the securedi.dll dynamic load library.

Be sure that the securedi.def file **was** added to the project.