



IsaScript Documentation

(The following graphics are screen shots and SDK quotations from Microsoft® ISA Server 2006 and TMG which are the property of Microsoft Corp. and are included here for instructive use. Some images illustrate IsaScript, which is the property of Collective Software.)

Table of Contents

IsaScript Documentation.....	1
ISA/TMG Web Filtering API.....	4
Solution.....	4
Features.....	4
Requirements.....	4
Help is Available!.....	5
Installation of IsaScript.....	6
Install Procedure.....	6
Troubleshooting.....	6
Install rolls back (with red error message at the end).....	6
Frozen or hung install.....	6
A simple test to prove IsaScript is working.....	8
Main script setup.....	8
Access rule setup.....	11
Testing the Hello World script.....	12
Web filtering concepts.....	13
Some Good Starting points in the SDK.....	13
Avoid confusion on these pages.....	13
Global, Listener, and Per-rule scripts.....	14
Global script area (in the main filter settings tab).....	14
Listener script areas.....	14
Per-rule script areas.....	15
Filter priority class.....	16
Script Reference.....	17
Lua Language.....	17
Require (include) path for scripts or libraries.....	17
Initialize Function.....	17
Event Functions.....	18
OnAccessDenied.....	18
OnAuthComplete.....	18
OnAuthentication.....	19
OnEndOfNetSession.....	20
OnEndOfRequest.....	21
OnLog.....	21

OnPolicyCheckCompleted.....	22
OnPreprocHeaders.....	23
OnReceiveResponseHeaders.....	23
OnRouting.....	24
OnSendResponse.....	24
Callback Functions.....	25
AddHeader.....	25
GetHeader.....	25
SetADAuthenticatedUser.....	26
SetAuthenticatedUser.....	26
SetHeader.....	27
SetUserCachingKey.....	28
Utility Functions.....	28
ISA.DisableNotifications.....	28
ISA.DisableTextMatch.....	28
ISA.DisableWPXNotifications.....	29
ISA.GetHeader.....	29
ISA.GetServerVariable.....	29
Util.AddResponseBody.....	30
Util.AddResponseHeaders.....	30
Util.Base64Decode.....	31
Util.Base64Encode.....	31
Util.GetFBACookieName.....	31
Util.GetGlobal.....	31
Util.GetLDAPAttributeValues.....	32
Util.LogEvent.....	33
Util.LogInformation.....	33
Util.LogWarning.....	34
Util.PrintClient.....	34
Util.Redirect.....	34
Util.SendResponse.....	35
Util.ServeFile.....	36
Util.SetGlobal.....	36
Util.SetLDAPAttributeValues.....	36
Util.URLDecode.....	37
Util.URLEncode.....	37
Content Matching.....	38
SetupContentMatching.....	38
SetupTextMatch.....	38
Text matching callback function.....	40
DisableTextMatch.....	40
Content matching Example.....	41
Even more features.....	43
Filter licensing.....	44
Demo/Lab mode.....	45
Troubleshooting.....	45
Support for IsaScript.....	45
Appendix A: Other Examples.....	46
Strip domain part out of forwarded Basic Authorization.....	46

<u>Global script.....</u>	<u>46</u>
<u>Per-rule script.....</u>	<u>46</u>
<u>Read data posted to the FBA logon form.....</u>	<u>47</u>
<u>Per-listener script.....</u>	<u>47</u>

ISA/TMG Web Filtering API

Microsoft provides a powerful API to interact with web content passing through the proxy, allowing dramatic expansion and control over ISA/TMG's feature set. But there are many limitations:

- In the ISA/TMG interface, there is no facility to add or modify HTTP headers, just to block them.
- In the ISA/TMG interface, there is no facility to modify request or response bodies, just a limited ability to block by simple signatures.
- Filters must be coded as C/C++ DLLs.
- A high level of expertise is needed to avoid memory leaks or crashing the firewall service!
- It is very work-intensive to build a proof of concept filter.
- Due to the need for expert development, producing even small filters may be prohibitively expensive for many organizations.

Solution

IsaScript from Collective Software is a filter for TMG, ISA 2006 and 2004 that wraps the web filtering API and exposes it via the popular and easy to learn [Lua](#) scripting language. Create filters the fast and easy way.

Features

- Majority of the web filter API is exposed to Lua script.
- Improved [content matching](#) abilities not available in the native ISA/TMG API.
- No knowledge of C/C++ needed.
- Lua errors are handled and reported gracefully; no need to worry about memory leaks or firewall crashing.
- Useful for rapid filter prototyping, or even on production systems. Lua scripting incurs only a minimal performance overhead.
- Veteran ISA/TMG filter developers will find *IsaScript* intuitive to use, because it mirrors the C/C++ API in many ways.
- Build your own filters, or take advantage of our [expert consultants](#) to help create the perfect solution.
- *IsaScript* filter development work is far less expensive than building a native C/C++ filter.

Requirements

- TMG or ISA Server 2004/2006
- Microsoft .NET Framework version 2 should be installed on each ISA/TMG server.

Help is Available!

We are always happy to help you get our software set up and working. If you have questions or need assistance understanding/configuring/testing a Collective product, you can get in touch with our support staff quickly and easily. For the most up-to-date information, please see our Support page at <http://www.collectivesoftware.com/Support/>

Installation of IsaScript

Install Procedure

1. Close the ISA/TMG management console if it's open.
2. Execute the IsaScript msi file. This will stop your firewall service, install the filter and interface software, register the filter, and then re-start the firewall service.
3. If you are installing over a remote desktop session, keep in mind that when the firewall service stops and restarts your RDP connection may be frozen, dropped or timed out. If an error occurs during the installation and the firewall service cannot be restarted, you will need to access the console to troubleshoot further (see below).
4. You must run the installer on each ISA/TMG server in an array separately, so they will all have the filter files installed and registered.
5. If the installation completes with no errors, then you can proceed to the configuration section.

Troubleshooting

The installation normally completes without errors. However there are a few possible failure modes that can occur for this complex install process.

Install rolls back (with red error message at the end)

If you are presented with an error message on the final screen, then check out the application event log, which often will contain details on why the installation failed. The problem may be immediately solvable from this information, or you may need to work with Collective support for additional troubleshooting assistance.

Frozen or hung install

The installer tries to start the firewall service after it is done registering the filter components. In rare cases, everything may register properly but there could still be a problem preventing the firewall service from starting. In this situation, the installation may appear to hang on the "Starting services..." item. This is because it is trying repeatedly to start the service, and failing. In fact if you look at the application event log, you will see several errors from the firewall service as it tries to start. These messages may help identify the cause of the problem.

The install should eventually give up on starting the service, but it may take a long time. If necessary, you can expedite the rollback by going into the services control panel and setting the Microsoft Firewall service to Disabled temporarily (and applying that change). This will cause the installer to quickly give up, and it should then correctly roll back the installation while leaving the firewall service down. After this happens you can then re-enable and restart the firewall service.

This kind of problem should not normally occur, and will probably require additional troubleshooting by Collective support. However if you are able to fix the problem you

can re-run the install safely after completing this procedure.

A simple test to prove IsaScript is working

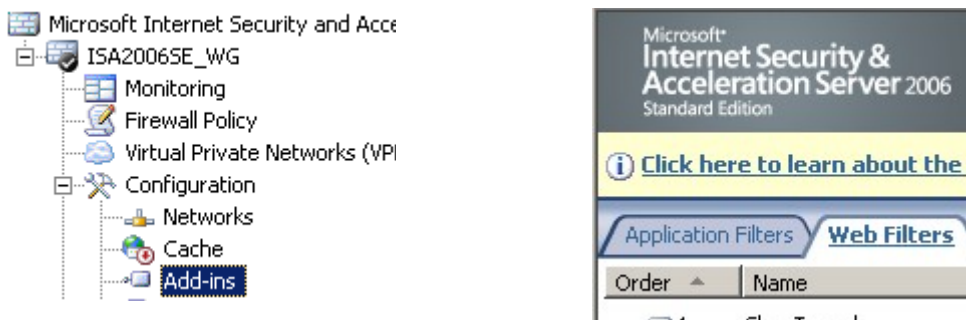
Before learning about everything IsaScript can do, it is useful (and instructive) to start with a simple “Hello world” example filter.

We will make a script that tells ISA/TMG to return the message “Hello World!” when a request is made to a specific URL.

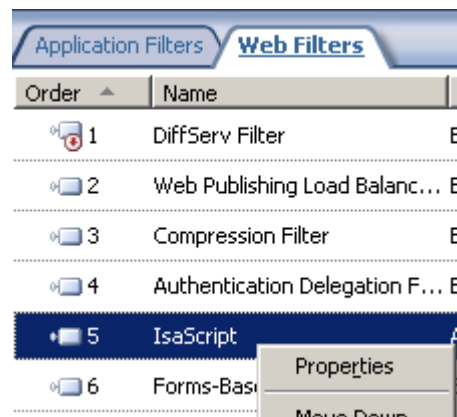
It's OK if you don't understand the details of how the script works yet.

Main script setup

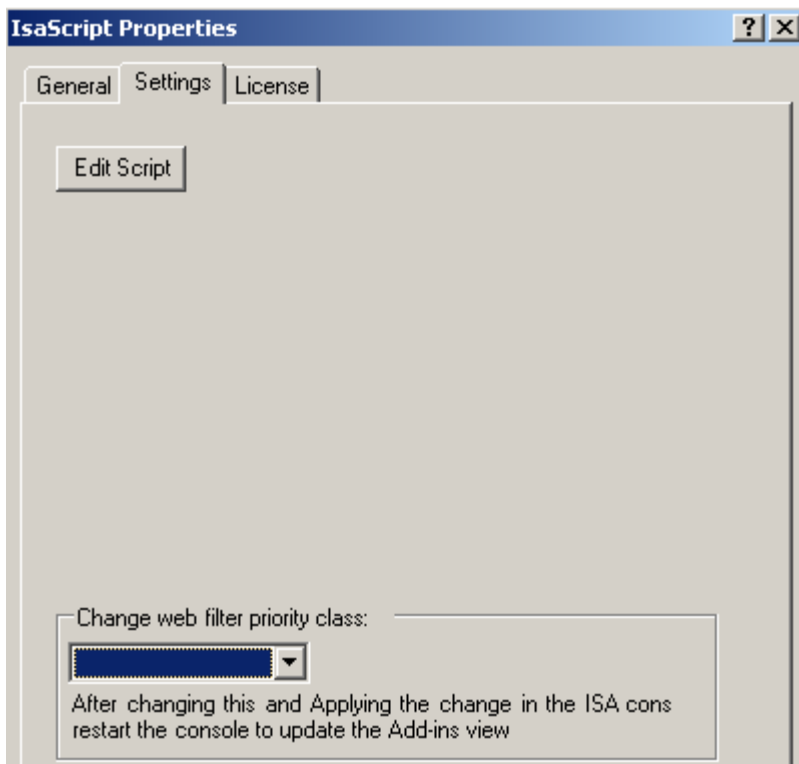
The main settings for IsaScript can be found in the Web Filter properties. In the Add-ins section, choose the Web Filters tab. Note that in TMG this tab is under the System item.



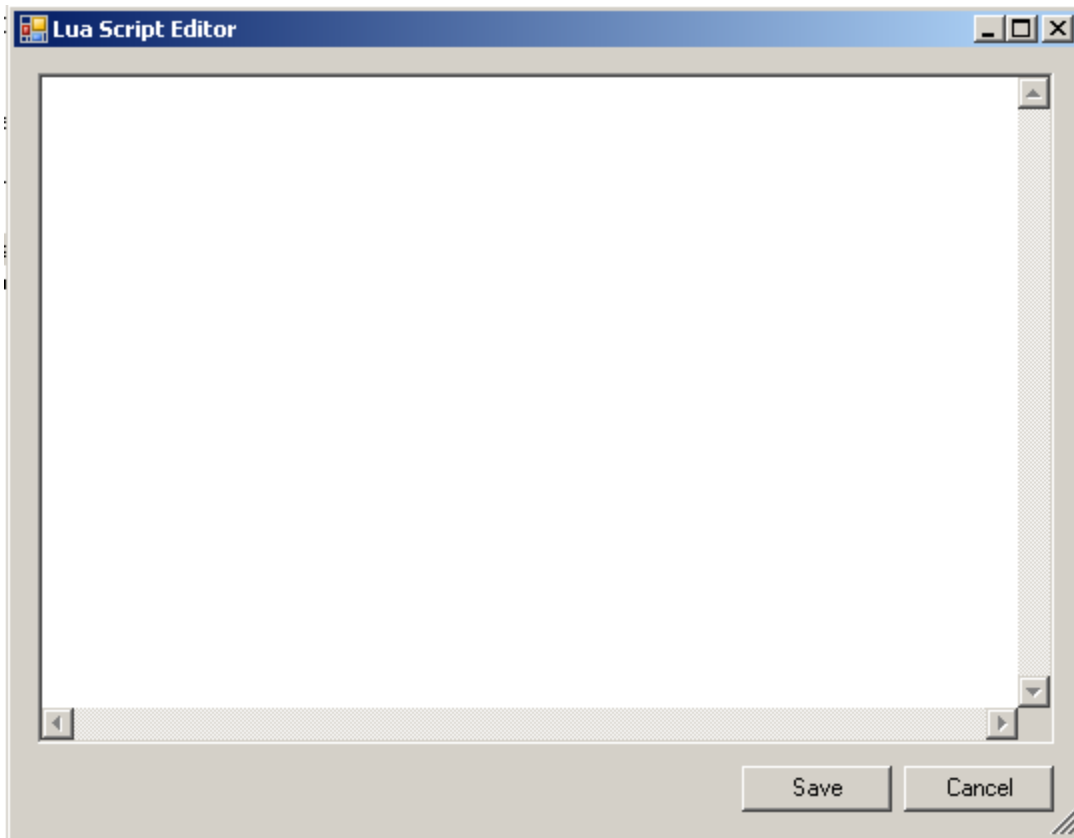
right-click IsaScript and go to properties:



Choose the settings tab:



Select the "Edit Script" button to open the editor window:



Add the following lines:

```
function OnPolicyCheckCompleted()  
end
```

Note that the body of the function is empty.. We don't want to perform any action globally for *all* requests. But this declaration is necessary to tell IsaScript that we intend to use the “[OnPolicyCheckCompleted](#)” hook later on. (More about the need for [global declarations](#).)

Save and apply these changes.

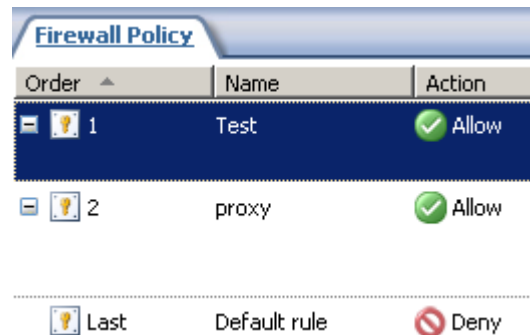
Access rule setup

Now let's create a policy rule that will use our custom filter. Create an access rule that matches a URL set, containing "www.collectivesoftware.com/Test".

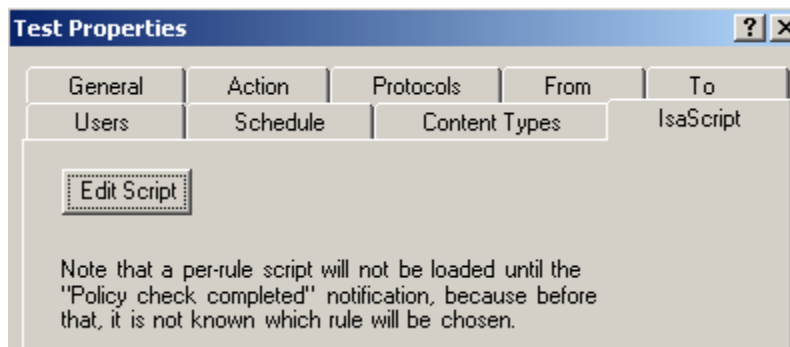
The image is a collage of screenshots from the Microsoft Internet Security & Acceleration Server 2006 "New Access Rule Wizard".

- Top Left:** The main wizard window titled "New Access Rule Wizard" with the Microsoft Internet Security & Acceleration Server 2006 logo. It says "Welcome to the New Access Rule Wizard" and "This wizard helps you create a new access rule. Access rules define the action that is taken, and the protocols that may be used, when specified clients from one network attempt to access a specific destination on another network." The "Access rule name" field contains "Test".
- Top Right:** The "Rule Action" section, titled "Rule Action", with the instruction "Select how clients should be treated if the condition is met." The "Action to take when condition is met" section has "Allow" selected with a radio button and "Deny" unselected.
- Middle Right:** The "Protocols" section, titled "Protocols", with the instruction "Select the protocols that this rule applies to." The "This rule applies to:" section has "Selected protocols" selected. The "Protocols:" list includes "HTTP" and "HTTPS".
- Bottom Left:** The "Access Rule Sources" section, titled "Access Rule Sources", with the instruction "This rule will apply to traffic originating from the". Below it, a box says "This rule applies to traffic from these sources:" and lists "Internal".
- Bottom Middle:** The "New URL Set Rule Element" dialog box. It has a "Name:" field with "collective/Test". A note says "If the DNS is not configured correctly, rules using this set will not be applied as expected." Below, it says "URLs included in this set (applicable for HTTP traffic only). Example: http://microsoft.com/somepath/*". The "URLs" list contains "http://www.collectivesoftware.com/Test".
- Bottom Right:** The "Access Rule Destinations" section, titled "Access Rule Destinations", with the instruction "This rule will apply to traffic sent from the rule source in this page." Below it, a box says "This rule applies to traffic sent to these destinations:" and lists "collective/Test".

Save the new rule. Make sure the new rule is *higher* than your existing proxy rules, if any. (Otherwise the more general proxy rule will run and our new one will *never* be matched.)



Go into the Test rule's properties, selecting the IsaScript tab:



Click Edit Script, and enter the following lines:

```
function OnPolicyCheckCompleted()  
    Util.PrintClient("Hello World!")  
    return SF_STATUS_REQ_NEXT_NOTIFICATION  
end
```

This means “as soon as ISA/TMG decides that this rule has matched, print this string to the browser”. After printing a string to the client, the request ends automatically, because this output replaces whatever content would normally be returned by the web server.

Save and apply these changes.

Testing the Hello World script

From the Internal network, open a browser and enter the URL “<http://www.collectivesoftware.com/Test>”. Instead of showing the contents of that page, you should see a single printed line “Hello World!”.

If this is not what you see, check the Monitoring Alerts tab to see if any script errors were recorded. If you get stuck, [help is available](#).

Web filtering concepts

This document assumes some level of familiarity with the [Microsoft ISA 2004/2006 SDK](#). If you are not comfortable with development concepts, you can [request](#) one of our consultants to help you write your script.

Some Good Starting points in the SDK

- [Web proxy documentation](#): Starting page for web filter API.
- [Introduction to Web Filters](#): Contains a good flowchart showing the organization of notifications in the proxy. When looking through events, it helps to understand what happens when.
- [Event Notifications](#): Information about what each notification does. Note that the "...RAW_DATA" notifications are not exposed to the Lua script directly; their functionality is wrapped by IsaScript's text matching features instead (which are a lot more useful)

Avoid confusion on these pages

There are some concepts and pages in the SDK that don't apply to IsaScript, and can be confusing for IsaScript users.

- [Web filter entry point functions](#), and [Entry point functions](#) pages: This functionality is all done automatically by IsaScript, and so you don't need to deal with or understand it.
- [Web filter basics](#): This page talks about installing the filter, which you don't need to worry about in IsaScript.
- [Signaling Events and Alerts](#): IsaScript provides a much simpler interface to [log events/alerts](#), so you can ignore this page.
- [Best practices for web filters](#): Most of this stuff is handled internally by IsaScript.
- [Callback functions](#): Some of these are useful in IsaScript, others handled automatically or not needed. The ones you can use yourself are discussed in the [Script reference](#) below.

Global, Listener, and Per-rule scripts

If you look back at the “Hello World” example, you can see that there were two areas where we entered script:

- In the filter's main settings tab
- In the “IsaScript” tab of the access rule

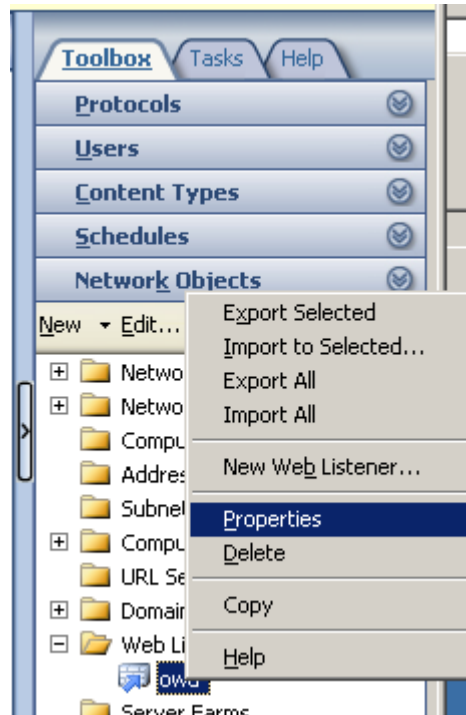
It is important to understand what the differences and limitations are for different script sections.

Global script area (in the main filter settings tab)

- [Event functions](#) defined here will run for every web request that goes through the proxy, regardless of which rule matches. This may or may not be what you want, depending on the filter you are making.
- If you want to set up an [event function](#) to run in a per-rule script, you must define an *empty* version of it in the Global area, the same way as we did for [OnPolicyCheckCompleted](#) in the “Hello World” example. The reason for this is that IsaScript needs to know right away which events may get used in your filter. At the beginning of a request it's not known yet what rule will be matched, so you have to give it a hint about the events by doing empty defines for all the ones you plan to use.
- If you wish to do any [content matching](#), you must define the [SetupContentMatching](#) function in the global area.
- You can set variables, make functions, and use the “[require](#)” statement in the global area. Everything you set up here will also be visible to the per-rule scripts you write, if any.
- The global script gets pre-compiled for speed and efficiency. The more of your functionality you can define in this area, the faster your filter will run.

Listener script areas

- In each Web Listener properties page, there is an IsaScript tab. To see this tab, make sure to open the properties from the "Toolbox" view:



Otherwise any third-party software tabs will appear missing (this is a design flaw in the ISA/MTMG console).

- You can use these per-listener areas instead of, or to override contents from the global script area. Because ISA/MTMG knows immediately which listener is used for a request, none of the limitations of per-rule script areas (below) apply here.
- Listener scripts are only run on sessions that flow through that listener.

Per-rule script areas

- [Event functions](#) defined here will only work if you have already defined an *empty* version of them in the Global area or Listener area, as noted above.
- Some web proxy [event functions](#) and [content matching](#) processing occur *before* the firewall policy rule is chosen, so you **cannot** use them in the per-rule area. These items are:
 - **SIDE_CLIENT / MODE_REQUEST / PART_HEADER**: Request headers from the browser are processed. If you need to do a per-rule match on headers, you can use the SIDE_SERVER value instead. That way you'll invoke the matching "after" ISA/MTMG has completed its rule processing.
 - **SIDE_CLIENT / MODE_REQUEST / PART_BODY**: Some (or all) of the request body may be processed, depending on how the packets are received and processed by ISA/MTMG. Again, you can define your matcher to use the SIDE_SERVER value, and it will only run after ISA/MTMG has finished rule processing.
 - **Initialize Function**: If you need to reset variables for each request, do this in global or listener scope.

- [OnPreprocHeaders](#): If you need to change headers before ISA/TMG does its rule processing, then you have to do it in the global area. To read the value of a header you can use the [GetHeader](#) callback function.
- [OnAuthentication](#)
- [OnAuthComplete](#)

You cannot use these events/matchers in a per-rule script, because until the request headers and authentication are finished, ISA/TMG has not yet decided on which rule to use!

- Apart from the above, you can use other events or matchers in the per-rule script area.
- Another approach is to set the value of a variable here, and have all the filter events and matchers defined in the global script. The global functions can change their behavior based on the value of this variable, thus yielding a “per-rule” effect while keeping the code base in one location. This has some advantages, because:
- Script in per-rule areas gets re-compiled during each request, in contrast to the fast pre-compiled global script. Thus as you add more code lines into per-rule scripts, it will add some CPU overhead to your proxy.

Filter priority class

The order in which web filters are called is important. To attain the behavior you want, sometimes it is necessary to position your filter higher or lower than other filters. For example, if your filter is trying to match text but getting only compressed content, move it below the Compression filter. If you wish to process all requests but find you are missing certain ones, you may need to move the filter to a higher priority than some other one that is blocking requests (such as a URL filter or anti-virus).

All web filters are divided into three priority classes: High, Medium, and Low. The class assigned to each filter is normally determined at design time by the developer, by analyzing where the filter should fit broadly into the list. Since IsaScript's behavior is not pre-defined, we can't know ahead of time what priority class you may need to use. Therefore, the main settings tab has a drop-down selection that allows you to set the class to any of the three values.

After setting the priority class, you can then use the up/down arrows in the web filters pane to position IsaScript within the other filters of its new class.

Script Reference

Lua Language

IsaScript uses [Lua](#) as its scripting language, because it is free, powerful, efficient, and easy to bind with C/C++ (which is the language the IsaScript filter uses internally). This document won't provide instruction on Lua, but several [tutorials](#), [references](#), and [books](#) are available online for free. The appendices of this document contain some examples that may also be instructive.

Require (include) path for scripts or libraries

The lua [require](#) statement is used to include compiled lua DLL libraries and other lua files. Any files you wish to include should be placed within the folder:

```
[ISA/TMG path]\Collective Software\IsaScript\lua
```

Some libraries are invoked with a dot-notation such as:

```
require 'luasql.odbc'
```

and in this case the file odbc.dll should be placed in the sub-folder:

```
IsaScript\lua\luasql
```

In basic operation, no additional libraries are required for IsaScript. However, lua library DLLs may be added to provide additional features such as database connectivity. Keep in mind that web filter operations should be short and synchronous, so as not to cause a slow-down in the proxy.

“require” statements should always be located in the global script.

Initialize Function

If you have a Lua function called Initialize() in your global or listener scope, it will get called (at least) once before each HTTP request. This is useful because the scripts will normally remember global variables for the duration of the web browser's connection to ISA.

If you want to reset global variables to a known value freshly for each request, use this function. It is not guaranteed that there will be exactly one request per call to this function; but it will always be called at least once before a fresh request cycle begins in the session.

Example

```
-- a string in global scope, available to all functions
-- for the duration of this "net session":
city = ""

function Initialize()
    -- reset this to blank each time we get a new request
    -- so stale values don't confuse us!
    city = ""
end
```

Event Functions

In general, when you implement an event function it should return SF_STATUS_REQ_NEXT_NOTIFICATION unless you have a compelling reason to return a different value.

SF_STATUS_REQ_ERROR can be used to signal an error condition and stop the request processing. In general it is more elegant to write a response to the client instead using the WriteClient function, and then return SF_STATUS_REQ_FINISHED

OnAccessDenied

SDK Summary

“Sent just after ISA Server determines that access is denied for the requested resource, but before ISA Server sends a response to the client.”

Arguments

- pszURL: A string that specifies the URL that requested access to the resource.
- pszPhysicalPath: A string that specifies the physical path of the resource that was requested.
- dwReason: A numeric value containing flags that indicate the reasons for the denial. [More information](#).

Notes

This event is *not* called by ISA/TMG for 502 responses, such as the common “The ISA Server denied the specified Uniform Resource Locator”. To catch these and other filter responses, use the [OnSendResponse](#) event instead.

Example

```
function OnAccessDenied(args)
    Util.LogInformation("Denied access to URL: " .. args.pszURL)
    return SF_STATUS_REQ_NEXT_NOTIFICATION
end
```

OnAuthComplete

SDK Summary

"Sent just after ISA Server determines that access is allowed to the requested resource in an HTTP authentication scenario."

Arguments

- GetHeader: the [GetHeader](#) callback function.

- SetHeader: the [SetHeader](#) callback function.
- AddHeader: the [AddHeader](#) callback function
- fResetAuth: If set to true, the authentication process will be reset, and no impersonation will be done.
- SetUserCachingKey: the [SetUserCachingKey](#) callback function.

Notes

The real [auth complete event](#) contains a callback to get a Win32 impersonation token, useful in cases where the web filter needs to perform actions on behalf of the user. This version of IsaScript does not support impersonation.

Important SDK note: "When the request is authenticated, ISA Server removes every Authorization header before sending the request on to the upstream proxy server or Web server. (In forward-proxy scenarios, it removes Proxy-Authorization headers. In reverse-proxy scenarios, it removes Authorization headers.) Therefore, even if the filter adds an Authorization header after receiving the SF_NOTIFY_AUTH_COMPLETE notification, it will be removed."

If you want to add a custom Authorization header to the forwarded request, you can use the [Content Matching](#) system (set to SIDE_SERVER, MODE_REQUEST, PART_HEADER).

Example

```
function OnAuthComplete(args)
    -- cache this content keyed to the user name
    -- in addition to the URL
    user = ISA.GetServerVariable("AUTH_USER")
    url = ISA.GetServerVariable("URL")
    args.SetUserCachingKey(url .. "://" .. user)
    return SF_STATUS_REQ_NEXT_NOTIFICATION
end
```

OnAuthentication

SDK Summary

"Sent just before ISA Server attempts to authenticate the client. Can be used to provide a custom authentication scheme. If authentication is not required by any rule, the notification is sent with an anonymous user."

Arguments

- pszUser: string that specifies the user name for this request. An empty string indicates an anonymous user.
- pszPassword: string that specifies the password for this request.
- fAuthIsRequired: true/false, Indicates whether authentication is required for this request.

- SetAuthenticatedUser: the [SetAuthenticatedUser](#) callback function.
- SetADAuthenticatedUser: the [SetADAuthenticatedUser](#) callback function.

Notes

If Basic credentials are passed in, the pszUser and pszPassword arguments will be filled. You can use one of the callback functions to set the user that ISA/TMG will use for this request.

The ISA/TMG form authentication filter *does not use this event* at all. To extract credentials from form authenticating clients you'd have to parse the body of the POST request before the form authentication filter handles it, using text matching. Collective's FlexForm filter uses this approach.

For windows integrated authentication, the password is never sent, so you cannot read it out of the pszPassword value.

If you just want to read information about what user has authenticated, it is most reliable to wait until the [OnPolicyCheckCompleted](#) event and use [GetServerVariable](#) to poll the AUTH_USER and other values. [OnAuthComplete](#) is also an OK choice, but be aware that for NTLM authentication, it is only called for the *first request* in a session. So if your filter needs to read out the user during each request, keep that in mind.

You cannot alter the user by changing pszUser and pszPassword directly. If you want to set these values, use the [SetADAuthenticatedUser](#) callback, or change the Proxy-Authorization header before this event fires.

OnEndOfNetSession

SDK Summary

"Sent when a network session with a client is ending."

Arguments

(none)

Notes

An ISA/TMG "session" normally refers to a single connection from the browser. This session can span several request/response cycles, including requests to different HTTP servers. Don't assume that all requests within a session will be to the same server, or that a session will last for all connections from a browser to a server.

OnEndOfRequest

SDK Summary

"Sent when the end of a request is detected."

Arguments

(none)

Notes

The term "request" here can be confusing. This notification is sent at the end of every request/response cycle, after all data has been received from the server. Note that [OnLog](#) is called *after* this event.

OnLog

SDK Summary

"Sent after the end of a request is detected, just before ISA Server writes the session to the ISA Server log."

Arguments

- pszClientHostName: The client's host name or IP address.
- pszClientUserName: the client's user name if authenticated, or "anonymous" otherwise.
- pszServerName: the name of the server to which the client is connected.
- pszOperation: the HTTP method.
- pszTarget: the target of the HTTP command.
- pszParameters: the information that is to be written in the Filter Information field of a Web proxy log entry.
- dwHttpStatus: The HTTP return status.
- dwWin32Status: The Windows error code.
- dwBytesSent: The number of bytes sent from the server to the client.
- dwBytesRecvd: The number of bytes received by the server from the client.
- msTimeForProcessing: The time, in milliseconds, required to process the client request.

Notes

This event is used to read (and rewrite) data that will go into the proxy log. Not

all columns in the log can be changed. This does not affect the *firewall log*; there is currently no way to programmatically change the firewall log entries.

If you set the pszParameters entry, you should append the old value to the end, so that other filters' values can be preserved too.

In our testing, the dwBytesSent value does not seem to update when re-set; the log window always reflects the value originally passed in.

Example

```
function OnLog(args)
    -- prepend something to the "Filter Information" column
    args.pszParameters = "Filtered! ;" .. args.pszParameters
    return SF_STATUS_REQ_NEXT_NOTIFICATION
end
```

OnPolicyCheckCompleted

SDK Summary

"Sent after the ISA Server policy check has been completed, and the request has either been allowed or denied. After this notification has been received, the Web filter can request the GUID of the policy rule that either allowed or denied the request."

Arguments

(none)

Notes

This event is the first one (on the flow chart) that can be used in a per-rule script. Also, many of the [GetServerVariable](#) values aren't populated prior to this point. When this event is reached, your per-rule script (if any) has been loaded. If you have events or variables defined there, they will be available.

Example

```
-- This example would be placed in a per-rule script
function OnPolicyCheckCompleted()

    -- assume this variable was set to a default
    -- value of false in a previous event,
    -- such as OnStartOfRequest

    processThisRequest = true

    -- now on some future event callback in this request,
    -- we can read that and act accordingly

    return SF_STATUS_REQ_NEXT_NOTIFICATION
end
```

OnPreprocHeaders

SDK Summary

"Sent when the ISA Server Web proxy finishes preprocessing the headers in a request. May be used by filters to modify the headers before the Web proxy begins to process the information in them."

Arguments

- GetHeader: the [GetHeader](#) callback function.
- SetHeader: the [SetHeader](#) callback function
- AddHeader: the [AddHeader](#) callback function
- SetUserCachingKey: the [SetUserCachingKey](#) callback function

Notes

This is a popular event because it can be used to read and change request headers on their way in to ISA. But keep in mind that this occurs in the flow chart before authentication and before rule processing, so you must put your code in the global section.

Example

```
myAuthorization = ""
function OnPreprocHeaders(args)
    -- save the auth header, since ISA/TMG deletes it later
    -- some future event in the request could read the var
    myAuthorization = args.GetHeader("Proxy-Authorization:")
    return SF_STATUS_REQ_NEXT_NOTIFICATION
end
```

OnReceiveResponseHeaders

SDK Summary

"Sent after the response headers have been received. May be used by Web filters to modify the headers received from the Web server before the response is processed by ISA Server."

Arguments

- GetHeader: the [GetHeader](#) callback function.
- SetHeader: the [SetHeader](#) callback function
- AddHeader: the [AddHeader](#) callback function
- SetUserCachingKey: the [SetUserCachingKey](#) callback function

Notes

Similar to [OnPreprocHeaders](#) but on the response side. Headers dealt with here are from the HTTP response.

OnRouting

SDK Summary

"Sent before the Web filter forwards a request to the destination server. Can be used by a filter to override the host name, IP address, or port for the connection."

Arguments

- pszHost: string that specifies the host name or IP address.
- wPort: The port number for the host. (A numeric value, not a string)

Notes

If you want to make the request go to a different server than it was destined for, you can change the host and port here. This is also the first event where you can learn where ISA/TMG plans to send the request.

You can use the [GetServerVariable](#) function to read the ROUTING variable, and detect whether the request is being web-chained upstream or sent directly out. However (annoyingly) there's no way to set which behavior is being used. In other words, if the request is being chained there's no way to force it to be direct, and vice versa. This is an important distinction because for chained requests the protocol and security used are different than to a direct web server.

OnSendResponse

SDK Summary

"Sent after the request is processed, but before any headers are sent back to the client. This notification is not used when ISA Server returns a response from the Web server or from the cache. However, it is called when a filter generates a response by using SF_REQ_SEND_RESPONSE_HEADER (), or when the Web proxy generates a response, such as an error page."

Arguments

- GetHeader: the [GetHeader](#) callback function
- SetHeader: the [SetHeader](#) callback function
- AddHeader: the [AddHeader](#) callback function
- HttpStatus: The current HTTP status code.

Notes

Defining this event is the easiest way to detect 502 responses generated by the proxy. They do not trigger [OnAccessDenied](#) as one might expect.

Callback Functions

These functions can only be used when they are passed to your event function as an argument. For example:

```
function OnAuthComplete(args)
    args.SetAuthenticatedUser("someuser", "somespace", "somepassword")
    return SF_STATUS_REQ_NEXT_NOTIFICATION
end
```

Notice how the SetAuthenticatedUser function is a member of the "args" variable that was passed in. Callback functions are only usable in the specific events for which they are passed in.

AddHeader

SDK Summary

"Adds an HTTP header to an incoming request after an SF_NOTIFY_PREPROC_HEADERS or SF_NOTIFY_AUTH_COMPLETE event notification, to an incoming response after an SF_NOTIFY_RECEIVE_RESPONSE_HEADERS notification, or to an outgoing response after an SF_NOTIFY_SEND_RESPONSE notification."

Arguments

- **IpszName:** a string containing the name of the header to add. Header names should include the trailing colon (:). This parameter is not case-sensitive.
- **IpszValue:** a string containing the new value to add to the header.

GetHeader

SDK Summary

"Retrieves an HTTP header. The function retrieves a header for an incoming request after an SF_NOTIFY_PREPROC_HEADERS or SF_NOTIFY_AUTH_COMPLETE notification, for an incoming response after an SF_NOTIFY_RECEIVE_RESPONSE_HEADERS notification, and for an outgoing response after an SF_NOTIFY_SEND_RESPONSE notification. "

Arguments

- **IpszName:** a string containing the name of the header to retrieve. Header

names should include the trailing colon (:). This parameter is not case-sensitive.

The special values "method", "url", "version", and "body" (for requests only) can be used to retrieve individual portions of the request line. When the value "body" is specified, the function retrieves the part of the body that has already been received.

The special value "status" can be used to retrieve the status from the response line.

Note that the special values do not include a trailing colon.

Notes

Retrieving the request body via this function is unreliable. It's generally easier to use a text matching callback.

SetADAuthenticatedUser

SDK Summary

"Sets the authenticated Windows user. This enables associating information about an authenticated user with the current request."

Arguments

- lpszUserName: a string containing the user name.
- lpszPassword: a string containing the user password.

Notes

At this time token management is not supported, so the hToken argument is absent. Use this function when you want to set the user for this request to an integrated windows account that is known to the firewall. Don't use this if you're authenticating with RADIUS or LDAP, only for true Windows integrated logins where the firewall is a domain member.

SetAuthenticatedUser

SDK Summary

"Sets an authenticated non-Windows user. This allows associating information about an authenticated user with the current request."

Arguments

- `lpszUserName`: a string containing the user name.
- `lpszNameSpace`: a string containing the namespace of the authentication scheme for the user.
- `lpszUserGroups`: a string containing the user groups. The string supplied in `lpszUserGroups` can be a comma-separated list of user groups.

Notes

At this time `IsaScript` does not provide a custom authentication scheme of its own, but it is possible to use existing schemes and set the user information with this function. To get the appropriate namespace for this function, you can use the [GetServerVariable](#) utility function to read the `AUTH_TYPE` variable. This assumes that authentication has already completed, however. The values it may have, according to the SDK, are "Basic, Digest, Kerberos, NTLM, and RADIUS".

The groups argument is useful for defining access rules more broadly. If the name of one of these "groups" is in the user set for the rule then access will be allowed. These group names are not mapped to Windows groups; (windows users are not set using this function).

SetHeader

SDK Summary

"Modifies or deletes the value of a header or to add a new header. The function can also be used to modify the special values included in the request or status line."

Arguments

- `lpszName`: a string containing the name of the header to modify or delete. Header names should include the trailing colon (:). This parameter is not case-sensitive.

The special values "method", "url", "version", and "body" (for requests only) can be used to set individual portions of the request line. When the value "body" is specified, the function sets the part of the body that has already been received.

The special value "status" can be used to set the status in the response line.

Note that the special values do not include a trailing colon.

- `lpszValue`: a string containing the new string for the header, or to "\\0" (if the header is to be deleted).

Notes

The "body" value is unreliable. In general if you want to modify the body you should use text matching callbacks instead.

SetUserCachingKey

SDK Summary

"Set the user caching key-- the key that the request will be cached with. This key is used to check if the given request is in the cache, so there is no need to pass the request to the server. The key is also used to cache the server response after it arrives. The URL of the request is used as the default cache key."

Arguments

- `lpszCacheKey`: a string containing the new caching key.

Utility Functions

These functions are provided to all scripts by ISA/TMG itself or IsaScript. They can be called regardless of which event you are in, but some may not make sense or work properly in some cases. For example once response data has been sent to the browser, calling [Util.Redirect](#) won't really redirect the user, because it's too late to send a 302 response.

ISA.DisableNotifications

Summary

For advanced use. See [SF_REQ_DISABLE_NOTIFICATIONS](#) documentation.

Notes

Some flags, such as [OnEndOfRequest](#), cannot be disabled because IsaScript uses them internally to do important clean-up and resetting procedures.

ISA.DisableTextMatch

Summary

This is an advanced function that tells IsaScript to disable content matching that was previously set up using [SetupTextMatch](#). It is primarily useful to increase efficiency. See the Content Matching section [DisableTextMatch](#).

ISA.DisableWPXNotifications

Summary

For advanced use. See [SF_REQ_DISABLE_WPX_NOTIFICATIONS](#) documentation.

ISA.GetHeader

Summary

This is very similar to the [GetHeader](#) callback function, although it can only be used to read headers, not the special values.

Arguments

- A header name to retrieve, trailing a colon, as in:

```
Authorization:
```

Notes

This is a convenience wrapper of [GetServerVariable](#) that reads the ALL_RAW value and parses out the header you want. It is included because in many cases one wishes to read a header during some call where the normal [GetHeader](#) callback is not available.

Example

```
function OnRouting(args)
    -- can't use args.GetHeader because GetHeader is not given
    -- for this event!
    cookies = ISA.GetHeader("Cookie:")
    -- now have fun decoding the cookies :/
    -- ...
```

ISA.GetServerVariable

Summary

Wraps the SDK function of the same name, [documentation here](#).

Arguments

- Name of the variable to read

Example

```
username = GetServerVariable("AUTH_USER")
```

Util.AddResponseBody

Summary

Use with [Util.SendResponse](#) to add body bytes to the response that will be generated.

Arguments

- A string of data that will be added to the body

Notes

You can call this multiple times or from different locations, but once the [Util.SendResponse](#) function has been called, the response will be considered finished as soon as the currently executing script returns.

Example

```
-- set a cookie in the response
Util.AddResponseHeaders("Set-Cookie: MyCookie=1;path=/\r\n")
Util.Redirect("/some/page.html")
```

Util.AddResponseHeaders

Summary

Use with [Util.Redirect](#) , [Util.SendResponse](#) , or [Util.ServeFile](#) to add one or more headers to the response that will be generated.

Arguments

- A string of values in the form:

```
HeaderName: HeaderValue\r\nHeader2Name: Header2Value\r\n
```

Notes

You can call this multiple times or from different locations, but the argument in each call must be one or more header values, each terminated by the control characters `\r\n`. Once the [Util.Redirect](#) , [Util.SendResponse](#) , or [Util.ServeFile](#) function is called, the response will be considered complete as soon as the currently executing script returns.

Example

```
-- set a cookie in the response
Util.AddResponseHeaders("Set-Cookie: MyCookie=1;path=/\r\n")
Util.Redirect("/some/page.html")
```

Util.Base64Decode

Summary

Decodes a base-64 encoded value such as an Authorization header.

Arguments

- Encoded string

Example

```
-- authorization header looks like name:password when decoded
auth = Util.Base64Decode(ISA.GetHeader("Proxy-Authorization:"))
```

Util.Base64Encode

Summary

Encodes a string into a base-64 encoded representation.

Arguments

- Value to encode

Example

```
encoded = Util.Base64Encode("myuser:mypass")
```

Util.GetFBACookieName

Summary

Returns the name of the cookie that is used in FBA logons for this web listener.

Notes

If the cookie name has not been set by the user, this returns the value "cadata %w+" which is a Lua pattern that can be used to match the unique cookie name randomly chosen by ISA. These default cookies always begin "cadata" and then have alphanumeric characters appended after.

Util.GetGlobal

Summary

Gets a value from persistent memory.

Arguments

- Variable name to retrieve

Notes

Since the Lua script state is loaded fresh for each proxy session, it is not possible to store data during one event function, and have it reliably available for future requests. You can use [GetGlobal](#) and [SetGlobal](#) to store values persistently in ISA's memory.

Values are encrypted in memory, although since the encryption key is also stored in memory it should not be considered strong protection. In a broader sense, if someone is snooping the memory on your firewall, you likely already have bigger problems than this.

This store is lost whenever the firewall service stops. To store values permanently or share values between ISA/TMG servers, you could use *luasql* to interface to a database.

Util.GetLDAPAttributeValues

Summary

If ISA's LDAP settings are configured, you can use this function to get values for one or more attributes on a user object.

Arguments

- A string in the form "domain\username", which specifies the user of interest.
- A string specifying the user's password. This is used to authenticate to the LDAP connection. Ideally leaving this blank would tell it to use default credentials; alas this is not the case currently. You have to specify creds here.
- A list (table) of strings, each string specifying the name of an LDAP attribute on the user object that you are interested in retrieving. Attributes containing non-string types have not been tested at this time.

Notes

The function returns a table whose indices are the attribute names. Each value is really another table, containing one or more strings. This is because many LDAP properties can have a "value" that consists of a list of strings.

On failure, the second return value is an LDAP error code, and the third (if present) is a string containing additional information about where the error occurred.

Currently the function assumes the user you want to query is the same one

whose credentials you are passing in to the function. This is only useful if you have the user's username and password, i.e. if you record them from the Basic auth or FBA. In the future it may be generalized to be more useful in other cases.

Example

```
attribs = {"physicalDeliveryOfficeName", "distinguishedName"}
values, errN, errS =
    Util.GetLDAPAttributeValues(username, pass, attribs)
-- did we find the record?
if(values) then
    if(not values["physicalDeliveryOfficeName"] or not
        values["physicalDeliveryOfficeName"][1]) then
        -- empty gets nil, we'd rather have ""
        values["physicalDeliveryOfficeName"] = {""}
    end
    if(not values["distinguishedName"]) then
        Util.LogWarning("I could not find an LDAP record for user
" .. username .. " (DN check)")
    end
    realCity = values["physicalDeliveryOfficeName"][1]
    dn = values["distinguishedName"][1]
end
```

Util.LogEvent

Summary

Logs an informational event to the special event log "IsaScript", which is visible through the Windows Event Viewer.

Arguments

- A string containing the text you want to record in the event.

Util.LogInformation

Summary

Triggers an ISA/TMG "informational" alert, visible in the alerts tab and (by default) the Application event log.

Arguments

- A string containing the alert message you want to record

Notes

This triggers the IsaScript-Information alert. By default, this alert suppresses multiple triggers within the same one-minute interval. If you are using this function to record critical information, you may wish to adjust this behavior. Handling of alerts can be configured in the Alerts tab.

Util.LogWarning

Summary

Triggers an ISA/TMG "warning" alert, visible in the alerts tab and (by default) the Application event log.

Arguments

- A string containing the alert message you want to record

Notes

This triggers the IsaScript-Warning alert. By default, this alert suppresses multiple triggers within the same one-minute interval. If you are using this function to record critical information, you may wish to adjust this behavior. Handling of alerts can be configured in the Alerts tab.

Util.PrintClient

Summary

A debugging function that sends a text message to the browser, skipping the normal response.

Arguments

- A string containing the message you want to send to the browser

Notes

You can call this several times in the same event function to append more data. However at the end of the current event, the data will be sent to the browser and the response will be ended.

This is a somewhat crude "print" function designed for debugging. To send html to the client you can use [ServeFile](#).

Util.Redirect

Summary

Sends an HTTP 302 response to the browser, instructing it to redirect to a different URL.

Arguments

- A string containing an fully qualified URL such as:

`http://www.collectivesoftware.com/Products`

Or, a server-less URL which will be interpreted relative to the current server:

`/Products`

Notes

The redirect function can qualify a server-less URL so you don't have to construct the whole thing if you are just trying to redirect to a different path on the same web server.

If any response bytes have been sent to the client already, the redirect won't work.

Util.SendResponse

Summary

Sends (the beginning of) a completely custom filter-generated response to the browser.

Arguments

- The HTTP response line (without trailing `\r\n`)

Notes

To add headers and body to the response use the [Util.AddResponseHeaders](#) and [Util.AddResponseBody](#) functions.

Example

```
-- re-prompt when the user "GuestUser" authenticates
-- because that user is a shared logon account
-- IE will automatically use it, but we can prompt IE
-- to try again and allow the user to log in as someone else
user = ISA.GetServerVariable("AUTH_USER")
if user == "GuestUser" then
    Util.SendResponse("HTTP/1.1 407 Proxy Authenticate")
    Util.AddResponseHeaders("Proxy-Authenticate: Negotiate\r\n")
    Util.AddResponseHeaders("Proxy-Authenticate: Kerberos\r\n")
    Util.AddResponseHeaders("Proxy-Authenticate: NTLM\r\n")
    Util.AddResponseHeaders("Connection: Keep-Alive\r\n")
    Util.AddResponseHeaders("Proxy-Connection: Keep-Alive\r\n")
    Util.AddResponseHeaders("Pragma: no-cache\r\n")
    Util.AddResponseHeaders("Cache-Control: no-cache\r\n")
    Util.AddResponseHeaders("Content-Type: text/html\r\n")
    Util.AddResponseBody("<html>You need to authenticate</html>")
end
```

Util.ServeFile

Summary

Reads an html (or supporting type) file from the path:

`[ISA/TMG Folder]\Collective Software\IsaScript\HTMLFiles`
and sends it to the browser.

Arguments

- Name of the file to send.

Notes

The file *must* be in the HTMLFiles folder. For security, no other folders are checked.

For images, javascript, css, html, and txt files, the mime type will get set automatically. For other file types the mime type is set to "text/plain".

If you specify a file that's not found, a 404 response will be sent to the browser.

Util.SetGlobal

Summary

Stores a value to persistent memory (see [GetGlobal](#))

Arguments

- Variable name to set
- The value to be stored

Util.SetLDAPAttributeValues

Summary

Like [Util.GetLDAPAttributeValues](#), this function allows you to operate on the LDAP attributes of a user object. This function replaces the string value(s) of one or more attributes with one or more string values you specify. That is, for each attribute you can specify one or more values. (Not all LDAP attributes can store more than one string value)

Arguments

- A string in the form "domain\username", which specifies the user of interest.
- A string specifying the user's password. This is used to authenticate to the LDAP connection. Ideally leaving this blank would tell it to use default credentials; alas this is not the case currently. You have to specify creds here.
- A string specifying the user's distinguished name. Technically this can refer to a different user than the one whose credentials you supply, as long as the credentials are sufficiently powerful to have access to write to this other user's object.
- A list (table) of items to set. Each item in the list is a pair (table) whose first member is a string containing the name of the LDAP attribute to set. The second member is a list (table) of string values that will be set into that attribute.

Notes

To get the distinguished name of a user object, you can either fabricate it using knowledge of how your users are stored in AD (which OU's etc.) or you can do a GetLDAPValues and ask for the distinguishedName attribute.

Example

```
setvals = {"physicalDeliveryOfficeName", {someVariable} }  
errN, errS = Util.SetLDAPAttributeValues(username, pass, dn, setvals);
```

Util.URLDecode

Summary

Decodes a string in the URL-encoded format, such as URLs and simple form post bodies.

Arguments

- A string to decode

Util.URLEncode

Summary

Encodes a string into its URL-encoded format.

Arguments

- A string to encode

Content Matching

The web proxy's native API functions for accessing the body of HTTP messages are very rudimentary. Proper handling and processing of body data requires a lot of sophistication in the filter code. Buffering and content length control are all left up to filter developers to implement correctly, which places a heavy burden on any filter that wishes to modify body content.

So, in addition to the *Header functions provided through event functions, IsaScript exposes HTTP message (header and body) contents through a system of content matching callbacks. In this system, you specify a part of the HTTP conversation you are interested in, and provide a matching expression and the name of a callback function. IsaScript examines the HTTP data stream using your expression, and calls your function when a match occurs. The function is free to modify the data, and any change will be correctly merged back into the HTTP stream. This enables powerful content control, while internally abstracting away all the difficult details.

SetupContentMatching

Summary

In order to match an expression and be notified, your filter must define a function called SetupContentMatching in the global script area. Inside this function, you will tell the filter about what you intend to match, and what function should be called for each match. SetupContentMatching is called at the beginning of each proxy session.

Arguments

- SetupTextMatch: The [SetupTextMatch](#) callback function.

SetupTextMatch

Summary

This is a function passed in during [SetupContentMatching](#). You call it with various arguments to set up matches and actions the filter will take during the session. This callback is only available to be used within the scope of the [SetupContentMatching](#) function. You can call this several times with different arguments in order to set up many matchers during the same session.

Arguments

- Side: Either SIDE_CLIENT or SIDE_SERVER. The ISA/TMG API allows filters to intercept data at either "side" of the proxy.

Data coming from the browser is considered to be on the client side *first*,

then the server side *after* it passes through ISA. Data coming from the server is considered to be on the server side *first*, then the client side *after* it passes through ISA.

- Mode: Either MODE_REQUEST or MODE_RESPONSE. If your filter is to match traffic from browser to server, choose MODE_REQUEST. If you want to match data returned from the server to the client, use MODE_RESPONSE.
- Part: Either PART_HEADER or PART_BODY. If you want to match header strings use PART_HEADER, otherwise select PART_BODY to match body bytes.
- Pattern: This is a string containing a perl-compatible regular expression that will be used on your selected portion of the HTTP stream. Whenever this expression matches part of the stream, the matching portion will be sent to your callback function.
- Callback: This is a string containing the name of a function you will define that gets called upon each match.
- Can modify?: Either *true* or *false*. This value should be set to *true* if your PART_BODY callback intends to change the *length* of the data in the HTTP stream.

Notes

Side: Your filter exists within an ecosystem of other filters and hooks. There's no simple rule that can tell you which side to choose; it depends on whether you want to filter data that has already been processed by ISA/TMG and other filters, or to get it beforehand, whether it's going to be request or response data, and what priority the IsaScript filter is set at, relative to other filters.

Pattern: A tutorial of how to use [regular expressions](#) is beyond the scope of this document. However there are a few points important to filter efficiency:

- Use [non-greedy \(or 'lazy'\) quantifiers](#) whenever possible, to match the narrowest possible amount of text. Using the quantifier `".*"` is dangerous because it can match all bytes.
- Craft your expressions to include well-known leading and trailing bytes, a string always known to be beyond your desired match bytes. This will help the filter efficiently plan how much of the stream to "hold" in memory before deciding it does not match. You can do this implicitly as in the example below where the beginning matches `"<title>"` and the end matches `"</title>"`, or explicitly using a [positive lookahead and lookbehind](#).
- If you want case insensitive matching, the first characters of your expression should be `"(?i)"`
- There is currently no facility to pass "sub matches" to your callback function. However, lua string matching functionality is usually sufficient to parse the matching bytes and acquire any sub-parts you need.
- When using the quote(`"`) character in your expression, you need to escape it with a backslash so the lua script can pass it correctly.

- When passing regex escape entities such as `\s` or `\w`, you must (additionally) escape the backslash character so lua does not interpret it. So `\s` for example becomes `\\s`.
- When you want to actually match a literal backslash (`\`) character, you must **double** escape it, to `(\\)`. Yes, that's 4 in a row. This seems excessive, but it is necessary because lua will parse it down to `(\\)` and then that sequence is what the regular expression processor will interpret as "match a literal backslash".

Can modify: This is an advisory value only, and does not *prevent* you from modifying the data if set to *false*. However, incorrectly specifying the value as *false* and then changing the length of the HTTP body **will** lead to HTTP errors, since the HTTP header may specify the wrong body length.

When *can modify* is set to *true*, IsaScript will force HTTP messages with bodies to use chunked encoding. This is necessary because during the header processing, there's no way to know in advance what the ultimate content length of the message will be. For most purposes, you can ignore this detail. However it should be noted that technologies such as download/upload progress bars often depend on a knowledge of the content size in order to display meaningful data. Using chunked encoding can interfere with these heuristic indicators.

If your matcher needs to be:

- `SIDE_CLIENT` with `MODE_REQUEST`, or
- `SIDE_SERVER` with `MODE_RESPONSE`,

then it will operate *before* ISA/TMG has a chance to parse the HTTP headers (see [OnPreprocHeaders](#) and [OnReceiveResponseHeaders](#)). If you need to deal with header information in this configuration, you'll have to make content matching callbacks for the `PART_HEADER` and do the parsing yourself.

Text matching callback function

Summary

This is a user-named and user-defined function that will be called when a match occurs in an expression set by a [SetupTextMatch](#) call. The name of the function must be the same as what you specified in the *Callback* argument to [SetupTextMatch](#) .

Arguments

`matchText`: A string containing the matching portion of the HTTP stream. If you modify this string, the change will be merged into the stream.

DisableTextMatch

Summary

This is an advanced function. If you are in a `PART_HEADER` callback and

discover that you don't need to do any PART_BODY processing for this message even though you set up one or more matchers for it, you can call the function `ISA.DisableTextMatch` to cancel those matchers.

Body matching can be expensive because every byte of the body flows through the web filter and your lua code. If you only need to process certain bodies, you can use this function to increase performance. The FBA.lua code uses this approach. If it does not detect the URL as `"/CookieAuth.dll"` it disables body matching so it will skip processing any request bodies other than FBA POSTs.

Do not use this to disable matching on the SIDE/MODE/PART you are currently running. Doing this will very likely lead to the filter eating part of the data stream and the session will stall. It is only sensible to disable a matching type that has not yet been entered into.

Once called, all matchers for that SIDE/MODE/PART will be cancelled.

Arguments

- Side: Either `SIDE_CLIENT` or `SIDE_SERVER`.
- Mode: Either `MODE_REQUEST` or `MODE_RESPONSE`.
- Part: Either `PART_HEADER` or `PART_BODY`.

Content matching Example

```
-- A toy example that records the title tag of each page with the
-- content type of text/html, into the ISA/TMG log's "Filter
Information"
-- field, and preserves other filters' data in that field too.
function SetupContentMatching(args)

    args.SetupTextMatch(
        SIDE_SERVER,
        MODE_RESPONSE,
        PART_BODY,
        "(?i)<title>.*?</title>",
        "MyTitleCallback",
        false
    )
end

myTitle = ""
myContentType = ""

-- make sure vars
-- start empty on each request
function Initialize()
    myTitle = ""
    myContentType = ""
end

function MyTitleCallback(args)
    myTitle = args.matchText
end

function OnReceiveResponseHeaders(args)
```

```
myContentType = args.GetHeader("Content-Type:")
return SF_STATUS_REQ_NEXT_NOTIFICATION
end

function OnLog(args)
  -- don't quibble about case, force to lower for testing
  if string.find( string.lower(myContentType), "text/html" ) then
    -- eat the end tags: first 7 chars and last 8
    args.pszParameters =
      string.sub( myTitle, 8, -9) ..
      "; " ..
      args.pszParameters
  end
  return SF_STATUS_REQ_NEXT_NOTIFICATION
end
```

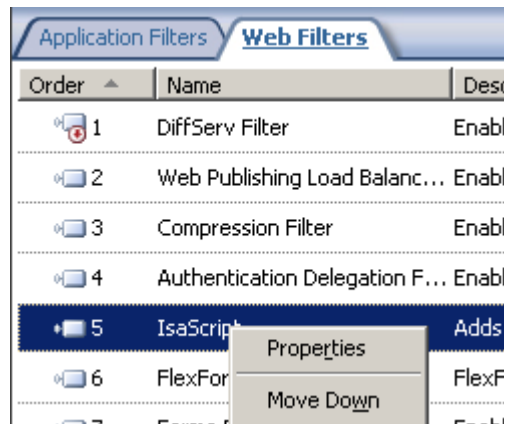
Even more features

If you see something missing, or you're not sure whether IsaScript can solve a particular filtering task, please let us know.

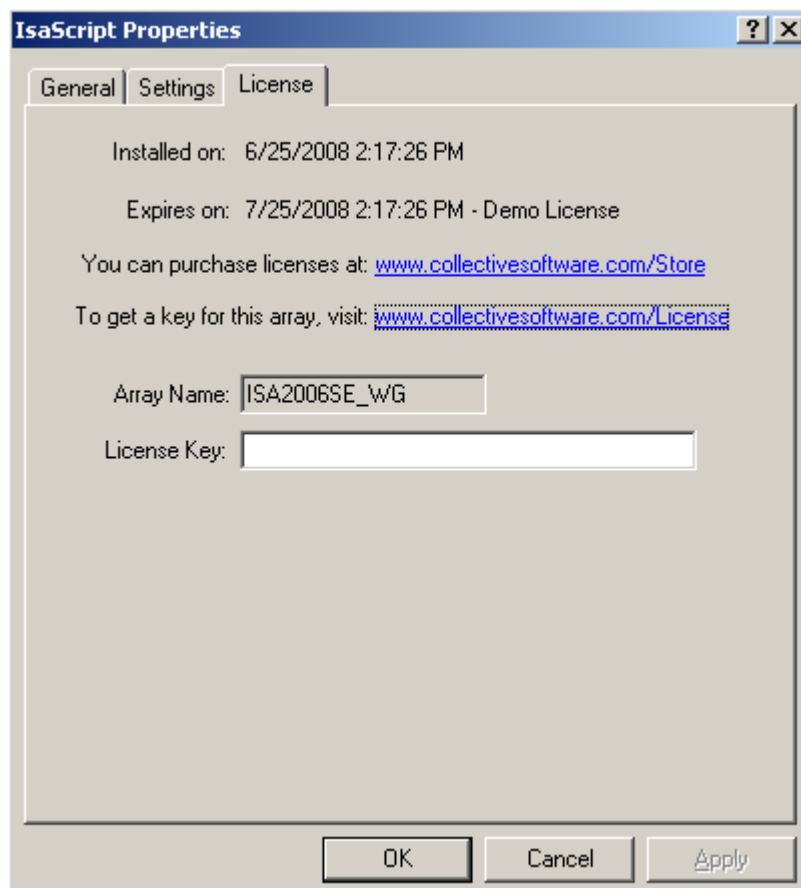
IsaScript was developed initially for our internal prototyping use, but based on a high demand for affordable custom filters, we have made it into a retail product. We will continue to add features as we need them internally, and as customers request them.

Filter licensing

To view your evaluation period or enter a key, go to Add-ins, Web Filters, and select IsaScript properties:



and select the License tab:



The License tab is used to check how long remains in the evaluation period, and to activate a permanent license.

To be eligible for a license key, you need to purchase license(s). You can do this on our [web store](#) or by [contacting us](#).

Once you have available license(s) you can request a key for your array (or single server) at our [licensing page](#). When requesting a license key, you will need to tell us the name of the ISA/TMG array, which is indicated on this dialog. The exact name is important, because it will be used to validate the key. Please note that for TMG systems, the characters “-TMG” will be appended to this name. Please tell us the name exactly as shown in your dialog.

The license key is sensitive to the number of servers in the array. For example if you begin with only 2 servers in the array but plan to have 4, you can purchase 4 licenses and request a license key for a 4-server array. Then as you bring future servers online, they will be licensed automatically.

Warning: if you install more servers than you have licensed then the license key will be seen as invalid, and the servers will begin to operate in [demo/lab mode](#). So if you need to add more servers to a live array then you should acquire and apply your new license key in advance, so this behavior does not take place.

Demo/Lab mode

When the evaluation period expires (after 30 days) or when an invalid license key is used, the filter runs in demo/lab mode. In this mode the filter will work normally for a period of 2 hours from the starting of the Firewall Service, and then stop working after that time. This mode is meant to be useful for test labs where you don't wish to purchase licenses but still want to be able to run meaningful test setups. After 2 hours, you can restart the firewall service and the lab timer will reset again.

Troubleshooting

The first place to look if something seems to be working incorrectly is the ISA/TMG alerts tab in the Monitoring section. Often this will directly indicate the cause of the problem. This information will also be required in almost all cases if you need support.

Support for IsaScript

Collective is proud to offer support for IsaScript, whether you need help getting a script working, find a bug, or just have a feature question.

Support is available from our web site at <http://www.collectivesoftware.com/Support/>

- *Knowledge Base:* When our staff answers questions that will apply to the whole community, they will often create a permanent KB item to disseminate this knowledge. There is a Search feature here; you can also easily browse by topic. To get fast answers to FAQs (frequently asked questions) the knowledge base is the best place to start.
- *Support ticket:* We are always happy to help you get set up and working. If you have questions or need assistance understanding/configuring/testing a Collective product, you can get in touch with our support staff quickly and easily. For the most up-to-date information, please see our Support page.

Appendix A: Other Examples

Strip domain part out of forwarded Basic Authorization

For this example to work, the IsaScript filter must have a higher priority than the Authentication Delegation Filter.

It also simplistically assumes, for UPN cases, that the username you want to forward is the part of the string left of the '@' symbol.

Global script

```
-- Control variables
-- Leave these declarations as-is here,
-- but copy the variable into per-rule scripts with the value /true/
-- if you want to perform that action in that rule
trimBasic = false

-- reset variables at the start of each request
function Initialize()
    trimBasic = false
end

function SetupContentMatching(args)
    args.SetupTextMatch(
        SIDE_SERVER,
        MODE_REQUEST,
        PART_HEADER,
        "\r\nAuthorization:\\s*Basic\\s+[^\\r]+",
        "TrimBasicAuth",
        false)
end

function TrimBasicAuth(args)
    if trimBasic then
        auth = string.match(args.matchText, "Basic%s+([^\r]+)")
        dAuth = Util.Base64Decode(auth)
        name, pass = string.match(dAuth, "([^:]+):(.*)")
        -- is name in domain\\user format?
        if string.find(name, "\\") then
            name = string.match(name, "[^\\]+[\\](.*)")
        end
        if string.find(name, "@") then
            name = string.match(name, "[^@]+")
        end
        newDAuth = name .. ":" .. pass
        newAuth = Util.Base64Encode(newDAuth)
        replaceText = "\r\nAuthorization: Basic " .. newAuth
        args.matchText = replaceText
    end
end
```

Per-rule script

```
trimBasic = true
```

Read data posted to the FBA logon form

For this example to work, IsaScript must have a higher filter priority than the "Forms-based authentication" web filter.

It makes use of the script "FBA.lua" included with the filter distribution. This script uses content matching functions and other event callbacks discussed above, to determine whether a request is an FBA post. If so, it forwards the name/value pairs in the form to a function you define.

This is useful to sniff the password entered into the form, or if you have edited the default FBA form to include other information you want to be passed into the script.

It is important to put this script into the Listener scope so you aren't content matching all traffic going through the proxy (i.e. forward proxy traffic)

Per-listener script

```
require "FBA"

username = ""
pass = ""
city = ""
function Initialize()
    -- call the initialization in the FBA script
    FBAInitialize()
    -- then do any other init needed here
    username = ""
    pass = ""
    city = ""
end
function SetupContentMatching(args)
    -- set up FBA content matching
    FBASetupContentMatching(args)
end
-- this function gets called by the FBA logic
function OnFormAuthentication(nvpairs)
    -- form data is passed as a table of name/value pairs
    if(nvpairs["username"]) then
        username = nvpairs["username"]
    end
    if(nvpairs["password"]) then
        pass = nvpairs["password"]
    end

    -- here, city is an <input> element we added to the form
    if(nvpairs["city"]) then
        city = nvpairs["city"]
    end
    return true -- if you return false it will force the auth to fail.
end
-- note that FBA posts skip many normal events, and the browser
-- closes the session right away. So if you want to use these
-- variables do it right away or store them as globals.
-- FBA.lua sets the global var "cookievalue" which can be used as a
-- unique key to set/get globals for this session. The example
-- listener.lua makes extensive use of this.
```