How to Specify PERMIS Policies for Controlling Access to Web Based Resources

Version	Date	Author
0.1	4 April 2006	Sassa
0.2	6 April 2006	David
1.0	26 Jun. 06	David

In this document we show how to specify a Web-Resource access control policy for PERMIS.

Generally speaking, policy specification for controlling access to web based resources is no different to specifying any other PERMIS policy. However, a web resource policy has to include the specifics of the web-server protocol used by web-browsers (i.e. HTTP), and the specific way that web resources are named. This affects the way that the both Resources and Actions are specified in PERMIS Policies (e.g. as created by the Policy editor),

1. Specifying the Resources

All the resources hosted by an Apache web-server are named with URLs. You as the policy writer need to identify these URLs, and any subdirectories which may have access restrictions on them. You then need to group your resources into domains that have the same access restrictions. As a general principle, domains with stricter access controls should be subordinate to domains with more relaxed access controls. Your policy should then include only those URLs that require the specific access, and exclude those that have stricter access permissions.

For example, assume you have one set of resources that has public access (the Public Domain), another set that has access restricted to say members of staff (the Restricted Domain), another set that has very restricted access, say to administrators only (the Top Secret Domain), and a final set that is dynamic content accessed via scripts (the Scripts Domain). Assume your web site is called http://www.mysite.com then you should place your publicly accessible resources in this location and its sub directories. Your Restricted Domain resources and Scripts Domain should be placed in subordinate directories of this, say starting at http://www.mysite.com/restricted, and http://www.mysite.com/cgi-bin respectively, and your Top Secret resources should be placed in a subdirectory of the Restricted Domain, say starting at http://mysite.com/restricted/secret (see Figure 1).

When specifying the PERMIS policy, you will need to specify 4 resources (as a minimum), one for each domain. The Public Domain resource should include http://www.mysite.com/ and should exclude http://www.mysite.com/restricted and http://www.mysite.com/cgi-bin. When specifying the Restricted Domain resource you should include http://www.mysite.com/restricted and exclude http://www.mysite.com/restricted/secret. When specifying the Top Secret Domain resource you should include http://www.mysite.com/restricted/secret.

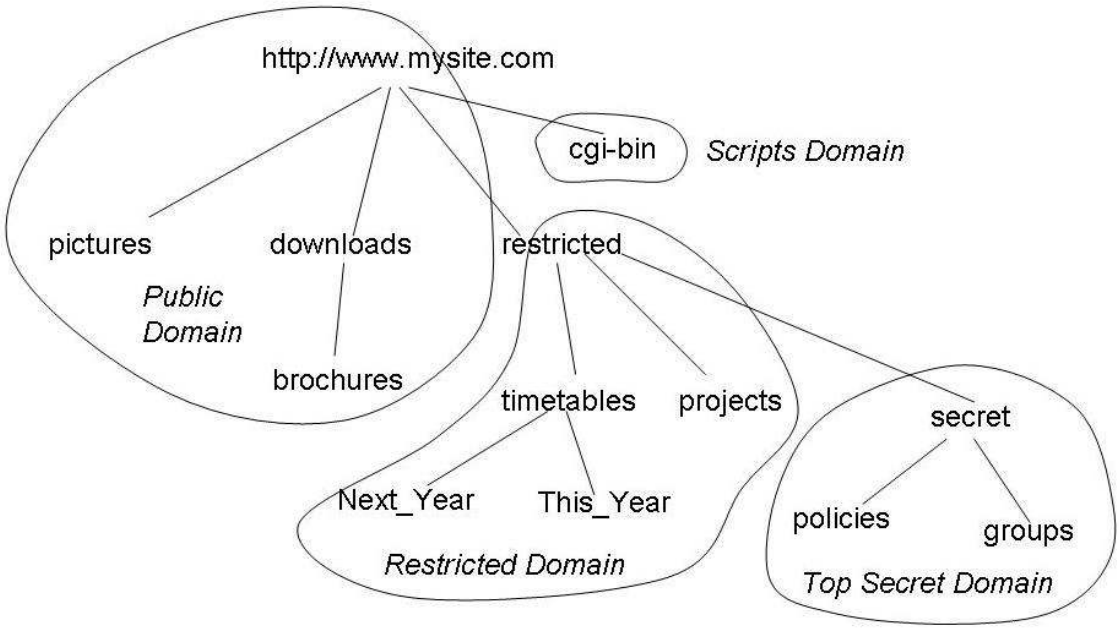

Figure 1. An example set of web resources in three different security domains

2. Specifying the Actions

The actions are defined at the level of the web-browser to web-server interaction protocol, HTTP. What might look as an innocent click on the link or a button, may generate different types of actions that the browser asks the server to perform. Often the user does not seem to do anything, yet the browser sends off action requests to the server, for example, when images are loaded as part of the web page.

There are 8 actions that directly correspond to the HTTP methods, defined in section 9 of RFC 2616 [1]:

- OPTIONS
- GET
- HEAD
- POST
- PUT
- DELETE
- TRACE
- CONNECT

The web server may run certain modules that extend the set of actions that the server may be asked to perform. For example, actions for Distributed Authoring (WebDAV) are defined in RFC2518, and their semantics are described there. This document will focus only on the standard actions used by web-browsers.

When the web server is configured to use PERMIS authorisation[1], then the PERMIS PDP will be queried by the web server in order to determine whether the requested access should be allowed or not. The web server will pass the URI from the request as the PERMIS Target Resource name, and the HTTP method name as the PERMIS Action name. Because the URI from the method

[1] When using an Apache web server, the httpd.conf file uses the LoadModule mod_permis command to load PERMIS and the PermisAuthorization command in each <Location> that is to be protected by PERMIS. See [2] for more details.

becomes the PERMIS Target Resource name, this means that in the case of dynamically generated content, the Target Resource name will be different for nearly every user request. Because the HTTP method name becomes the PERMIS Action, this means that the actions do not contain any arguments. The net result of this is that PERMIS policies cannot contain conditions based on arguments of the action, but should contain rules for many different targets or target domains.

All the HTTP method names are provided by the web server as is, and the PERMIS policy treats these Actions as case-sensitive strings, so that the Actions in the policy should always be specified in upper case letters (RFC2616 [1] requires the browsers to send them in upper case). Don't forget that if an Action is not mentioned in the PERMIS policy, it will be denied to everyone.

2.1 OPTIONS

The OPTIONS method is defined in section 9.2 of RFC2616:

> *The OPTIONS method represents a request for information about the communication options available on the request/response chain identified by the Request-URI. This method allows the client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating a resource retrieval.*

Since this method is primarily designed to query a resource about its capabilities and requirements, we can have very relaxed authorisation rules, because no resource retrieval or resource action is performed. Consequently, the OPTIONS method can usually be made available for public access, unless your server has any information about its capabilities/requirements that you do not want to make publicly available. Note that the RFC allows an "*" to be specified for the URI with this method, which refers to the server as a whole, and not to any specific resource.

2.2 GET

The GET method is defined in section 9.3 of RFC2616:

> *The GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URI. If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process.*

This is the most common method used by web browsers. It is used to retrieve any HTML pages, stylesheets and images used by the pages, download files, create dynamic content from databases, etc. So you should consider this is like "read" access to your resources. Note that retrieving a web-page may cause several GET requests, as individual resources (images, applets, CSS style-sheets, Java-scripts, Flash scripts, audio content, etc.) referenced on the web-page require individual GET requests. You need to make sure that the access controls to these elements of the web page are specified correctly. E.g. if you specify the page as a restricted resource but do not specify the content as a restricted resource, it will be possible to access the content directly, without accessing the page. On the other hand, if the access restrictions are too tight, a valid user may be able to see the page, but will not be able to see parts of it (images won't load, or the applet won't start, etc).

Sometimes this method, instead of POST, is used by browsers to send the completed contents of a form, for example, when dynamically created content is being requested from a database. Such forms can be easily identified. The "method" attribute of the form is set to "GET" E.g.

<FORM name="tel no query" action="/cgi-bin/retrieve.pl" method="GET">

Note that in a HTML FORM command the "action" attribute specifies what resource on the web server is the recipient of the form data. In terms of the PERMIS policy it is not an action, but rather it is the Resource being accessed by the user.

Such dynamic content queries can also be identified by the URL that appears in the location bar of the browser, when the data is submitted. If the URL contains a query string (everything after the question mark is the query string) e.g.

http://www.mysite.com/cgi-bin/retrieve.pl?name=David& surname=Chadwick&town=Salford

then the HTTP query that submitted the data used a GET method.

So, in conclusion, when specifying the access control rules for the GET method, regard it as "read" access to your files on your web server, but also in some cases e.g. when a form uses a GET method to submit some data to a script, then it may also be regarded as "write" access. When forms use a GET method to submit some data to a server-side script (CGI, PHP, ASP, JSP etc.) it is better to separate these scripts into a different target domain (*and exclude them from all other domains*), so that the access control rules can be specified cleanly and precisely for this domain. Note that if you forget to exclude these scripts from the other target domains, someone may have unexpected "write" access to your resources[2].

2.3 HEAD

The HEAD method is defined in section 9.4 of RFC2616.

> *The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response. The meta information contained in the HTTP headers in response to a HEAD request SHOULD be identical to the information sent in response to a GET request. This method can be used for obtaining meta information about the entity implied by the request without transferring the entity-body itself. This method is often used for testing hypertext links for validity, accessibility, and recent modification.*

As can be seen from the definition, this is almost the same as GET. However, it does not reveal the content of your resource. This is more like "list the content of the directory and return the information about the files, without returning the content of the files". In simple terms this means that the browser may see the date of last modification of the web page, or the checksum of the web page, which will help it manage its cache of web pages. However, if you think this information is as sensitive as the content itself, the HEAD action must have the same access restrictions as the GET action.

You need to design your access control rules correspondingly. These rules may be very relaxed, if you don't think the information about files is in any way confidential. E.g. if you want to make the contents of a project report available to the project members only, but you don't mind anyone knowing that the report exists, then you may not need to have strict rules for the HEAD method.

---

[2] This is true only if the scripts modify something on your server or in your database.

However, if you want to hide the existence of the report from everyone except the project members, then you will need to have strict access rules and only grant project members access to the HEAD method.

2.4 POST

The POST method is defined in section 9.5 of RFC2616.

> *The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line. POST is designed to allow a uniform method to cover the following functions:*
> - *Annotation of existing resources;*
> - *Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles;*
> - *Providing a block of data, such as the result of submitting a form, to a data-handling process;*
> - *Extending a database through an append operation.*

As can be seen from the definition, this method is used to update the state of the resources on your web server. So you should consider this is like "write (append)" access to your resources.

This action is used by browsers to send some content to web servers, e.g. the contents of a filled in web form. Usually this means that a server-side script is executed to process the content. You will need to specify who (which roles) is allowed to execute such scripts in the various resource locations. (See also comments on the GET action.) Note that it makes sense to specify the same access restrictions to the web page displaying the web form as to POSTing the form, because it usually does not make sense that a user is able to see the form (GET action is permitted on this web page), but is not allowed to submit it (POST action is forbidden).

If you are running web services, they may be accessed using the POST method. Even though the web services may have their own access control enforced, the web server will be the first gateway where the user of the web service can be authorised or rejected. Another time the POST method is used is when a Shibboleth SP sends a SAML Request to a Shibboleth IdP.

2.5 PUT

The PUT method is defined in section 9.6 of RFC2616.

> *The PUT method requests that the enclosed entity be stored under the supplied Request-URI. If the Request-URI refers to an already existing resource, the enclosed entity SHOULD be considered as a modified version of the one residing on the origin server. If the Request-URI does not point to an existing resource, and that URI is capable of being defined as a new resource by the requesting user agent, the origin server can create the resource with that URI.*

From the definition of the method it can be seen that it allows the user to submit new content to your web server. So you should consider this is like "write (replace) or write (create)" access to your resources. In effect, this allows the user to upload new HTML files, images and scripts onto your web server. This is exactly the method used by some Internet browsers to publish the content that you edit on your computer. This method will normally be forbidden to anyone, unless you want the users of your system to actually upload new web content and you are prepared for them to overwrite existing content. If you do not intend web publishing to be part of

the functionality of your web site, you should deny any access to the PUT action, i.e. do not create any Resource Access rules that talk about the PUT action.

2.6 DELETE

The DELETE method is defined in section 9.7 of RFC2616.

> *The DELETE method requests that the origin server delete the resource identified by the Request-URI. This method MAY be overridden by human intervention (or other means) on the origin server. The client cannot be guaranteed that the operation has been carried out, even if the status code returned from the origin server indicates that the action has been completed successfully. However, the server SHOULD NOT indicate success unless, at the time the response is given, it intends to delete the resource or move it to an inaccessible location.*

Basically, this complements the PUT method by allowing the users to remove the content they created. You should treat this as a "delete" access. As with the PUT method, this method will normally be forbidden to anyone, unless you want to let your users use web publishing tools to manage the contents of the web server.

2.7 TRACE

The TRACE method is defined in section 9.8 of RFC2616.

> *The TRACE method is used to invoke a remote, application-layer loop-back of the request message. The final recipient of the request SHOULD reflect the message received back to the client as the entity-body of a 200 (OK) response. ... A TRACE request MUST NOT include an entity.*

Basically, this is a method that allows you to diagnose your system from a remote computer. This method does not return any content from your web server, it only returns the headers received by the server from the browser. This method may be allowed public access rights with little harm to your system, but if you are extremely security conscious then you may want to disable it altogether.

2.8 CONNECT

The CONNECT method is defined in section 9.9 of RFC2616. It is defined as a method reserved for use by proxies and is not received by the end-point web servers. Therefore this command should never be seen by the PERMIS PDP.

References

[1] RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1 http://www.faqs.org/rfcs/rfc2616.html
[2] David W. Chadwick, Wensheng Xu, Alexander Otenko, Zhiqing Wu. "PERMIS SAAM Installation Cookbook"