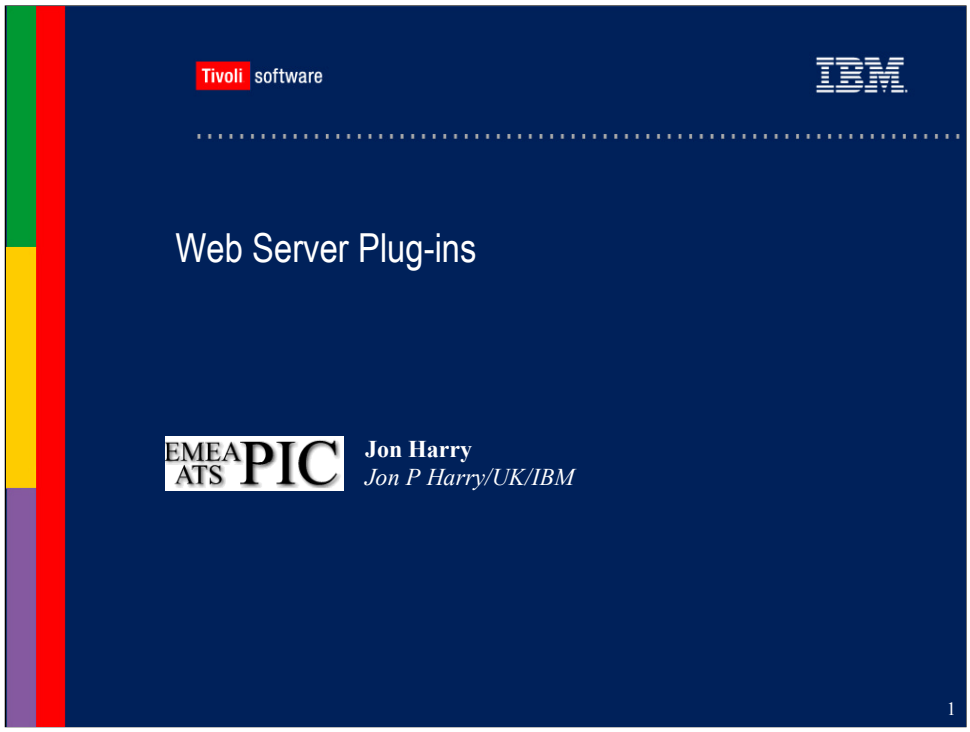
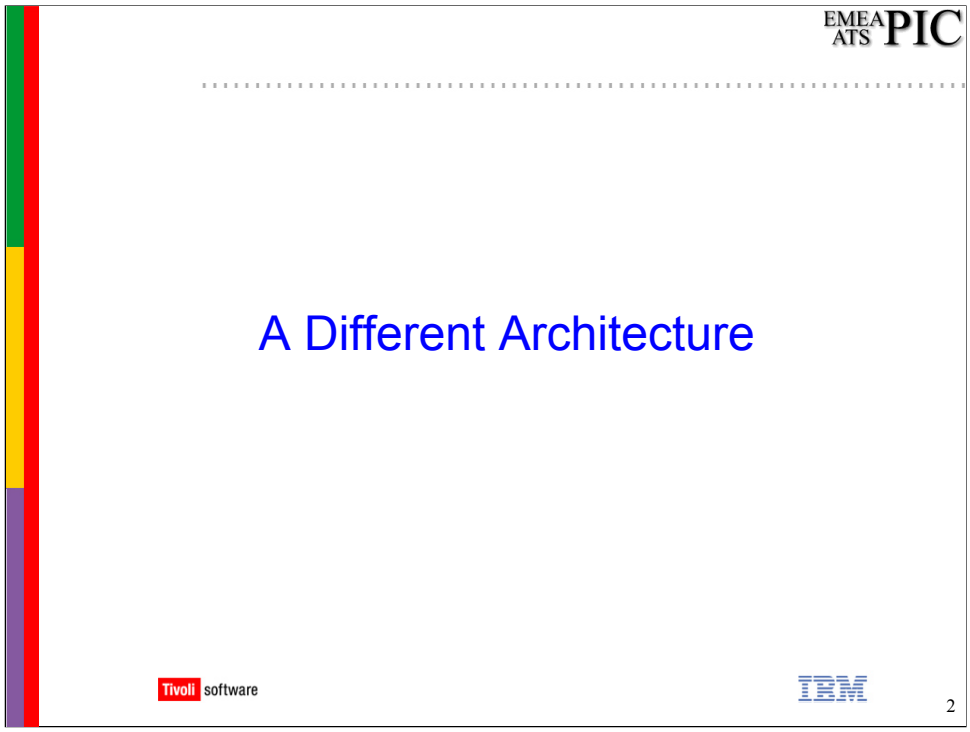
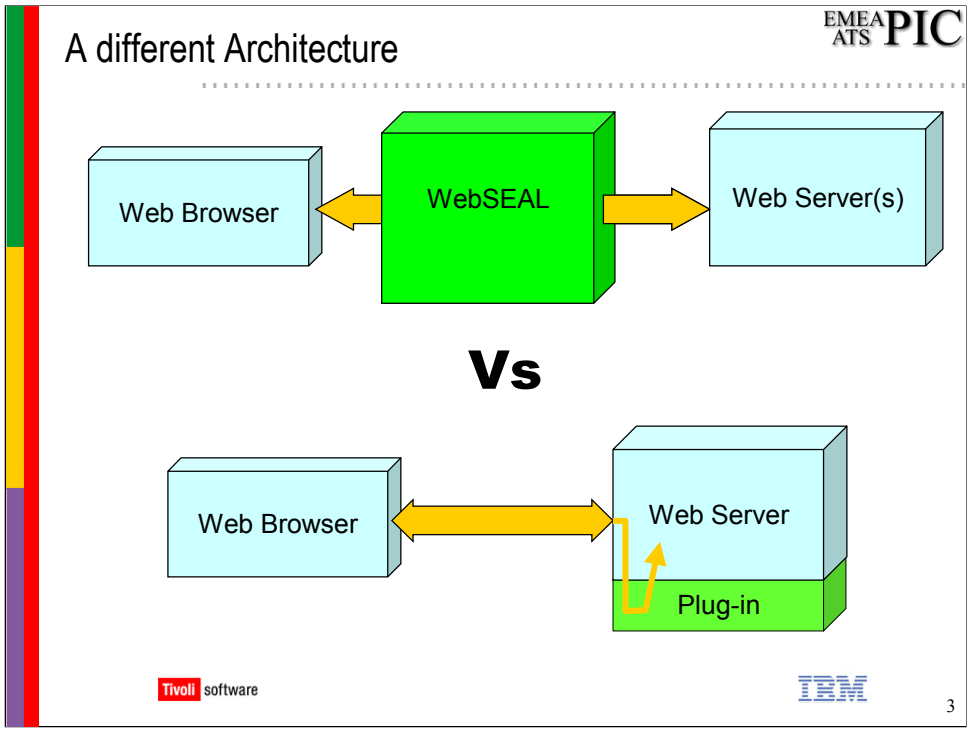


This presentation introduces Web Server Plug-ins – a new feature in Access Manager v3.9.

It describes the architecture and configuration.

Web Server plug-ins perform similar tasks to WebSEAL and many features are common. This presentation does not attempt to explain the details of each function; it shows how to configure and notes any differences.


EMEA ATS PIC
A Different Architecture
Tivoli software
IBM
2



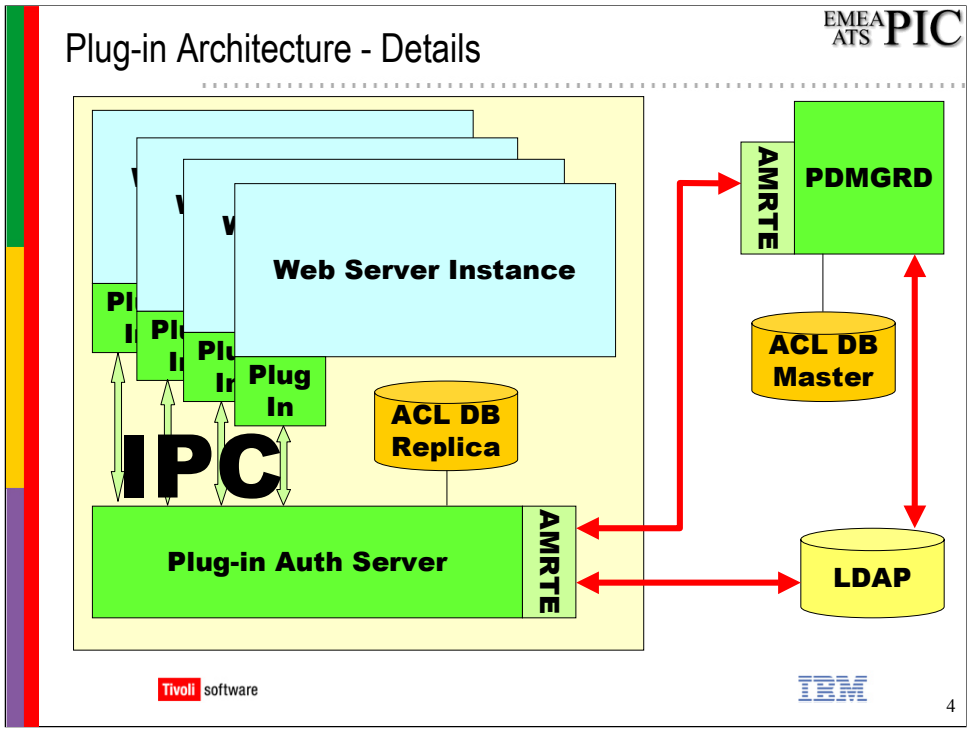
The diagram above show, at a high level ,how an architecture with a Web Server Plug-in differs from one that uses WebSEAL.

In a WebSEAL environment, the user communicates directly with WebSEAL, treating it as a Web Server. WebSEAL examines each request and, if authorized, passes it on (acting as a client) to the back-end Web Server.

In a Plug-in environment the user communicates directly with the Web Server but all requests are internally channelled through the plug-in code. The plug-in gets a chance to reject the request or to modify it before passing it onto the Web Server code.

The advantage of the Web Plug-in environment is that all communications are handled by the Web Server. This means that there are much less compatibility issues than when every request must be received and interpreted by WebSEAL.

The disadvantage of using Web Plug-ins is that there is no longer the same level of insulation from the end user that WebSEAL provides. The end users can now access the web server directly without (initially) providing any authentication. It is more likely that they can exploit the Web Server from this position.



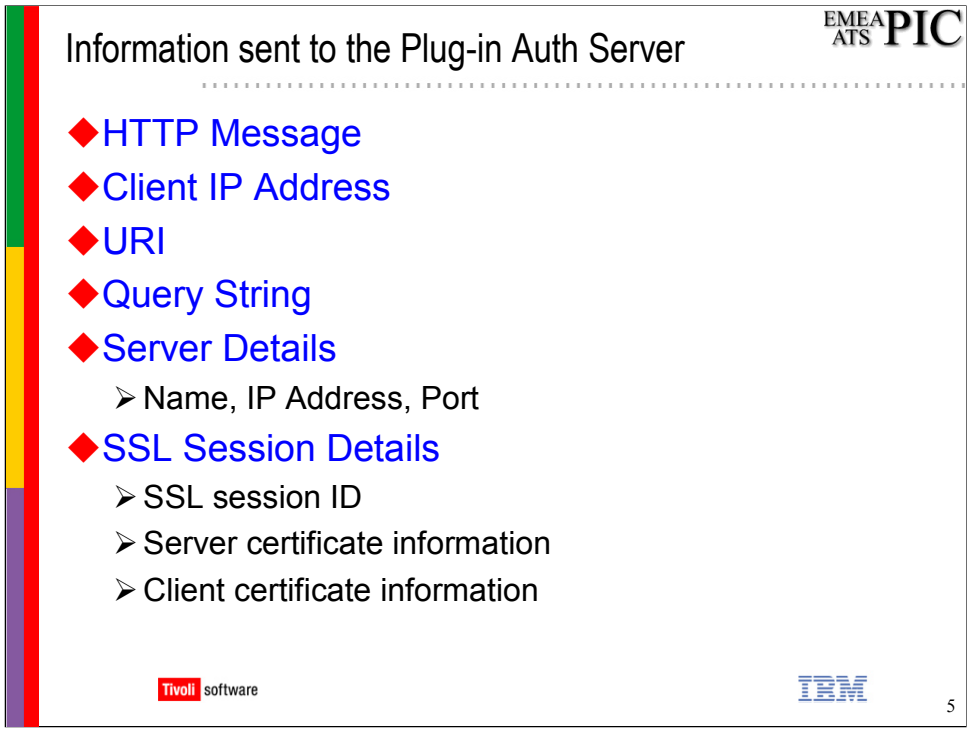
The slide above shows a more detailed view of the Web Server Plug-in architecture.

In most Web Server environments there are multiple server threads in operation on the machine. These might be different threads of the same Web Server instance or threads of different Web Server instances. Having a distinct authorization engine for each thread would be inefficient but would also mean that session information would have to be shared between them somehow.

The architecture used contains two parts:

Interceptor – This is the real "plug-in" part of the solution. Each Web Server thread has a plug-in running in it which gets to see and handle each request/response that the thread deals with. The interceptor does not authorize the decisions itself.; it sends details of each request (via an Inter-Process Communication interface) to the Plug-in Auth Server.

Plug-in Auth Server – This is where authorization decisions are made and the action to be taken is decided. There is a single Plug-in Auth Server on each machine and it can handle requests from all plug-in types. The Plug-in Auth Server is a local AZNAPI application which handles authentication and authorization for the plug-ins. The Auth Server receives intercepted requests from the Plug-ins and responds with a set of commands that tell the plug-in how to handle the request.

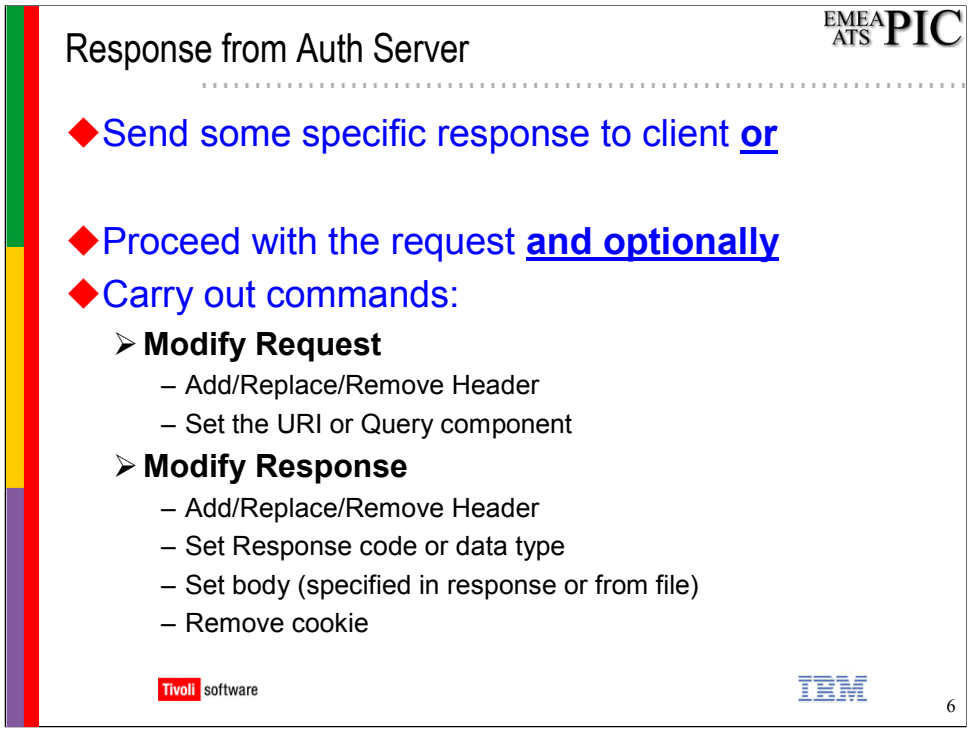


The list above shows the information about each request that is sent to the Plug-in Auth Server by the Plug-ins. This information is enough for the Plug-in Auth Server to identify the current session, authenticate the end user when required (using a variety of different methods) and make authorization decisions.

This may seem like a lot of information to be passing around for every request but it is important to remember that the communication between the plug-in and the plug-in Auth Server is very efficient because the processes are communicating directly – which is very much quicker than communication over network sockets.

Not all the information above is always available. For example, SSL certificate information is only available when SSL communication is being used.

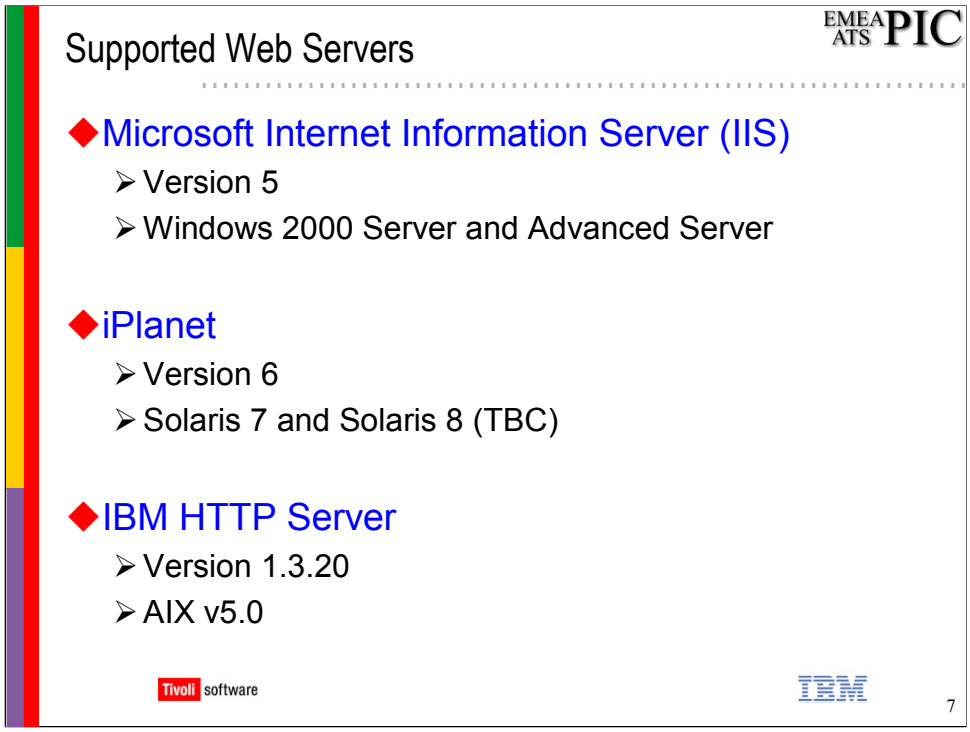
In addition, some Web Severs do not provide all of this information on their plug-in interface. Microsoft IIS, for example, does not pass the SSL Session ID to plug-ins. This means that this is not available as a means of session identification in this environment.



Based on the information that the plug-in Auth Server receives from the plug-in it can respond in a variety of ways. The responses are commands that tell the plug-in how to handle the request.

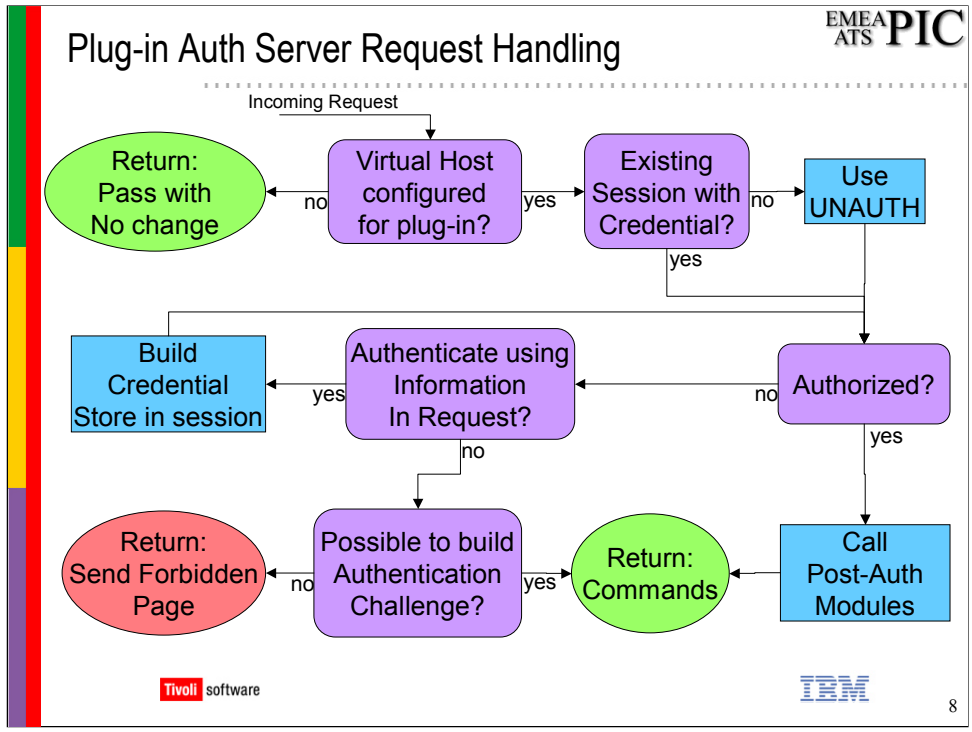
In the first case the Auth Server can instruct the plug-in to simply send some response to the client. In this case the Web Server code never sees the request – it is dealt with directly by the plug-in. This response could be used to send an authentication challenge or error message to the client.

An alternative response can instruct the plug-in to pass control back to the Web Server but to modify the request and/or response. Modifications to the request might include adding headers or cookies to the request for single sign-on.. It might replace the request with a different one. Modifications to the response are normally to set cookies on the browser (or to remove cookies set by the Web Server).



In this first release of Web Plug-ins only the dominant Web Server on each platform is supported – as shown above.

Although the interceptor plug-ins are quite different for each Web Server, the plug-in Auth Server code (which contains almost all the function) is very similar and so, once installed, the majority of the configuration of the Web Server Plug-ins is the same on each platform.



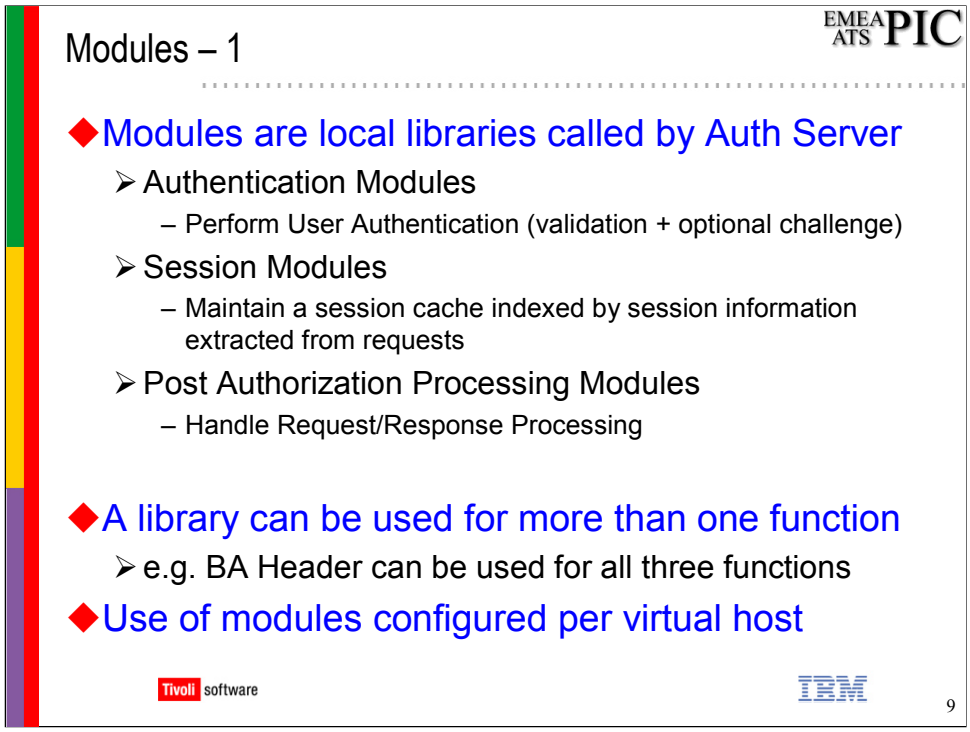
The diagram above shows how the Plug-in Auth Server handles requests that it receives via the plug-ins.

The first decision is whether the virtual host that the request is for is configured for protection. If not then the response to the plug-in is to pass it without change.

If the virtual host is one configured for protection then the first thing to do is to identify the user making the request. If possible this is done by identifying the request as part of an existing session that already has a credential assigned. In this case authorization can be done using that credential. If no credential can be found then the request is authorized with the UNAUTHENTICATED credential.

If the request is authorized then the Auth Server needs to determine whether any modifications need to be made to the request or response. This processing is done by post-authorization modules that build a list of commands to send back to the plug-in.

If the request is not authorized using the current credential then the Auth Server attempts to find authentication information (e.g. BA Headers) in the request to build a new credential. If it succeeds then this can be used to reattempt authorization. If there is no authentication information then the Auth Server attempts to build an authentication challenge response for the plug-in to send to the end-user. If this is not possible either then the forbidden page must be returned.

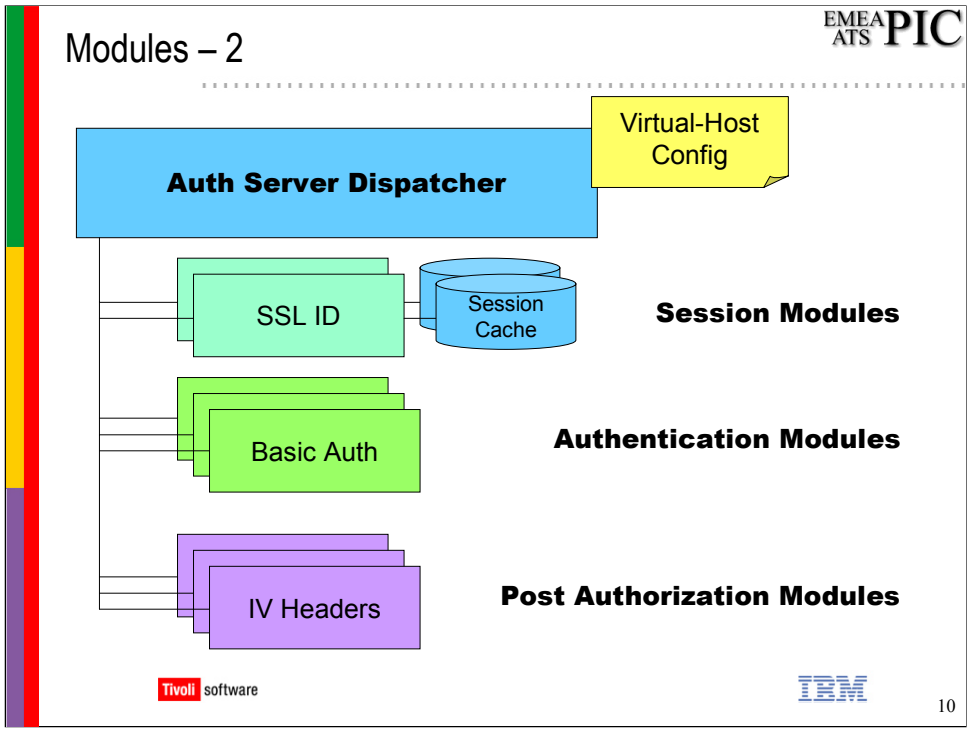


The Plug-in Auth Server performs the tasks shown on the previous page by calling modules that can provide these functions in different environments. The three module types are:

Session Modules: These are modules that can extract session information from a request and use it as an index to store/retrieve session information (e.g. user credential) in a session cache. This session information might be the SSL Session ID or the value in a session-cookie.

Authentication Modules: An Authentication module can perform two functions. First it must be able to extract authentication information from a request and validate it. This might be userID/password information from a form or the DN from an X.509 certificate. Second, it **may** be able to produce an authentication challenge that can be passed back to the end-user via the plug-in. Not all authentcation modules can produce a challenge.

Post-Authorization Modules – Post Authorization modules are used to provide additional functions. They are called after a request is authorized and have the chance to instruct the plug-in to modify the request or response. They might add IV Headers to the request for SSO or they might intercept a "special" URL and kick off processing for (for example) change password.

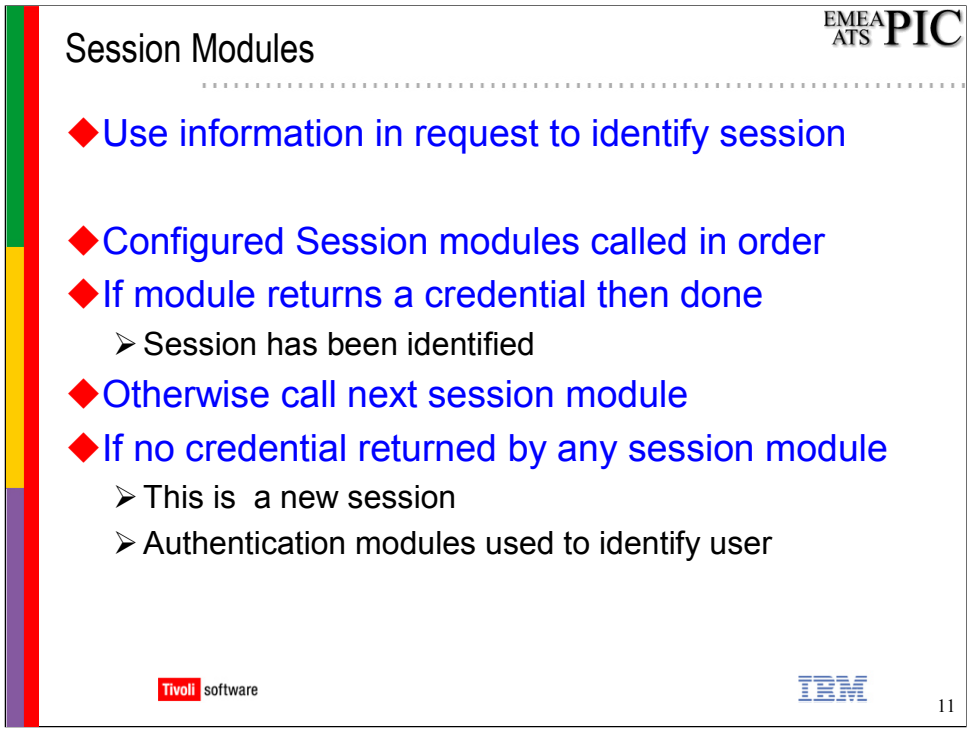


The modules used by the Plug-in Auth Server are loaded at start-up by a dispatcher and are then called as needed during request processing.

The modules used for a request are determined by the Plug-in Auth Server configuration. This can be specified on a per virtual-host basis so that different virtual hosts use different authentication mechanisms, use different session maintenance techniques and support different post-authorization features (failover, LTPA cookies etc)

In addition to specifying which modules will be used for a given virtual host the configuration also specifies the order in which the modules will be used. This is important in the case of authentication and session modules because the first one that succeeds is used.

A single library can contain code to perform more than one of the above module functions. For example, the Basic Authentication header can be used for both Authentication and for identifying a session. A single library implements both of these functions and can be called as both an authentication module and a session module.

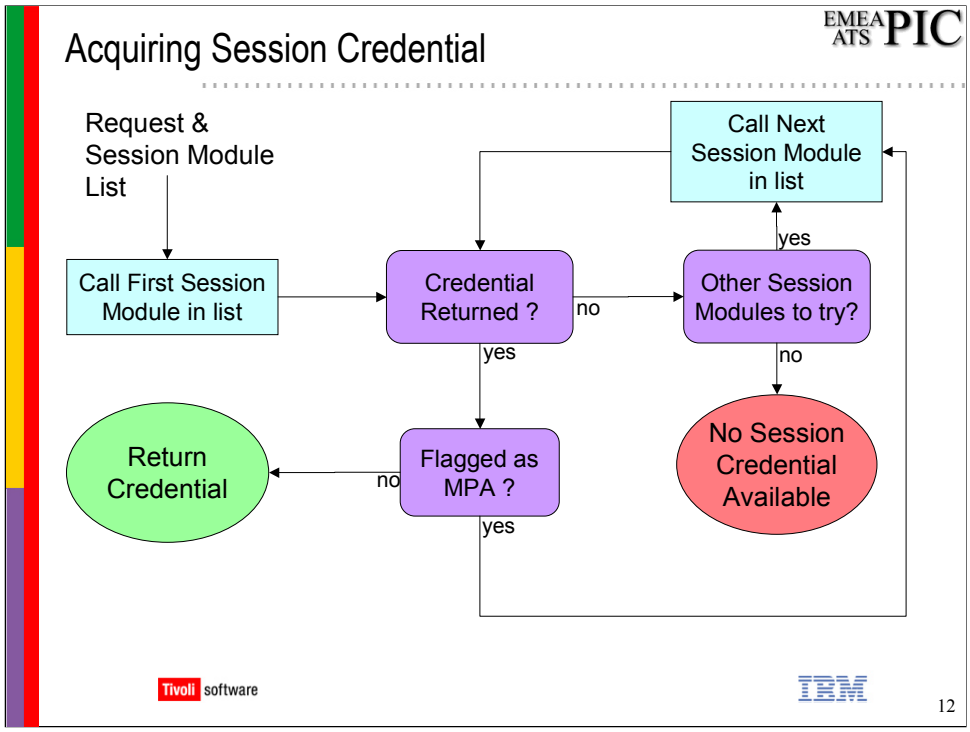


Session Modules are used as a way to insulate the Plug-in Auth Server from the different ways that a request can be identified as being part of an existing session.

The Plug-in Auth Server can call the session module to either create a new session, indexed by information in the request, or to use information in the request to identify an existing session and return the associated user credential from the session cache.

In both cases the session module may succeed or fail. If the module fails (because the information it needs is not present in the request or if there is no existing session) then the Plug-in Auth Server will simply call the next session module.

Each session module maintains its own session cache. This is because the information used to index the sessions differs for each session module type.



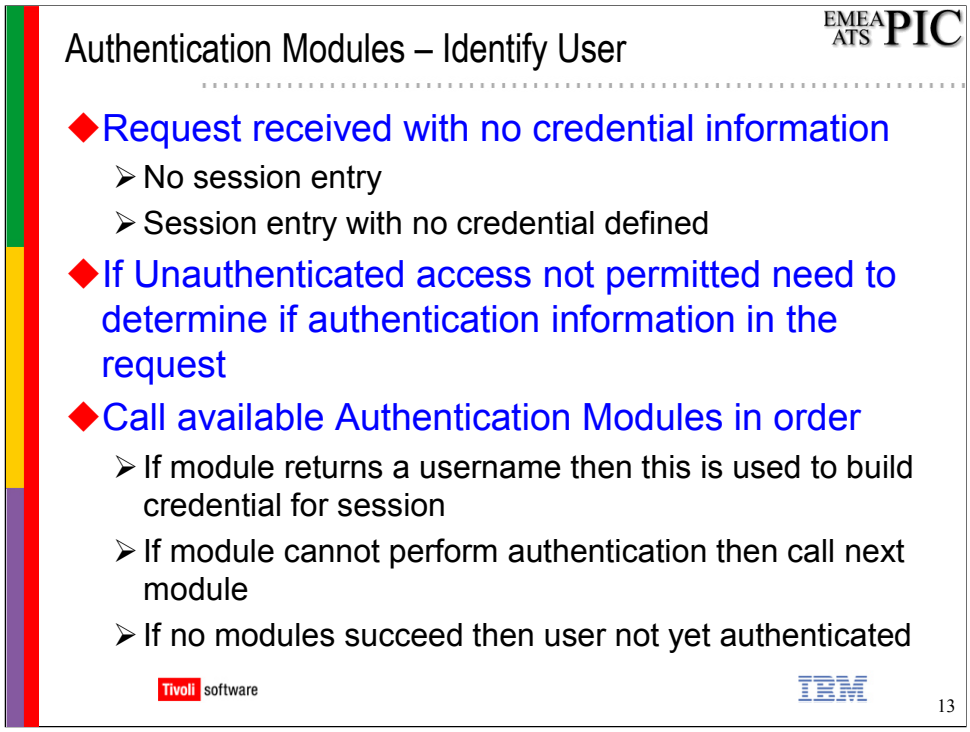
The flowchart above shows how the session module is used to simplify the process (from the Plug-in Auth Server point of view) of getting acquiring a user credential from a request for use during authorization.

The Plug-in Auth Server calls each of the configured session modules in turn. It doesn't care how the session module works – it just needs to know if a credential is returned or not. If not then it simply calls the next session module.

If a credential is returned then it is checked to see if it references an Multiplexing Proxy Agent. If it is a Proxy Agent then another session must exist for the real end user. To find this other session the Plug-in Auth Server continues to call the rest of the session modules.

If a user credential is returned then this means that an existing session was found for which user authentication has already taken place. This credential is used to authorize the request.

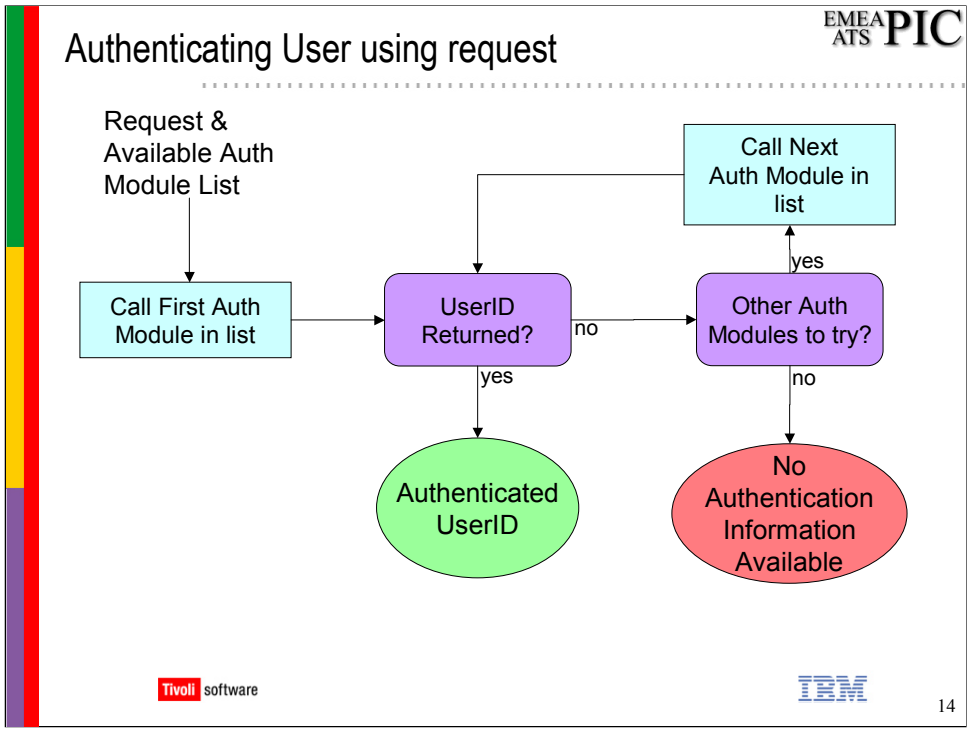
If all the session modules are called but none returns a user credential then this means that this is either a new session or a session for which there is not yet a credential.



When a request is received for which no session can be identified the request is initially authorized using the unauthenticated credential. If this fails then the Authentication modules are called, in order, to see if they can identify the user who sent the request from information contained in the request (e.g. Basic Authentication headers, DN from client certificate).

If an authentication module is able find authentication information and validate it then it will pass back a Access Manager UserID that can be used to build a credential for the session.

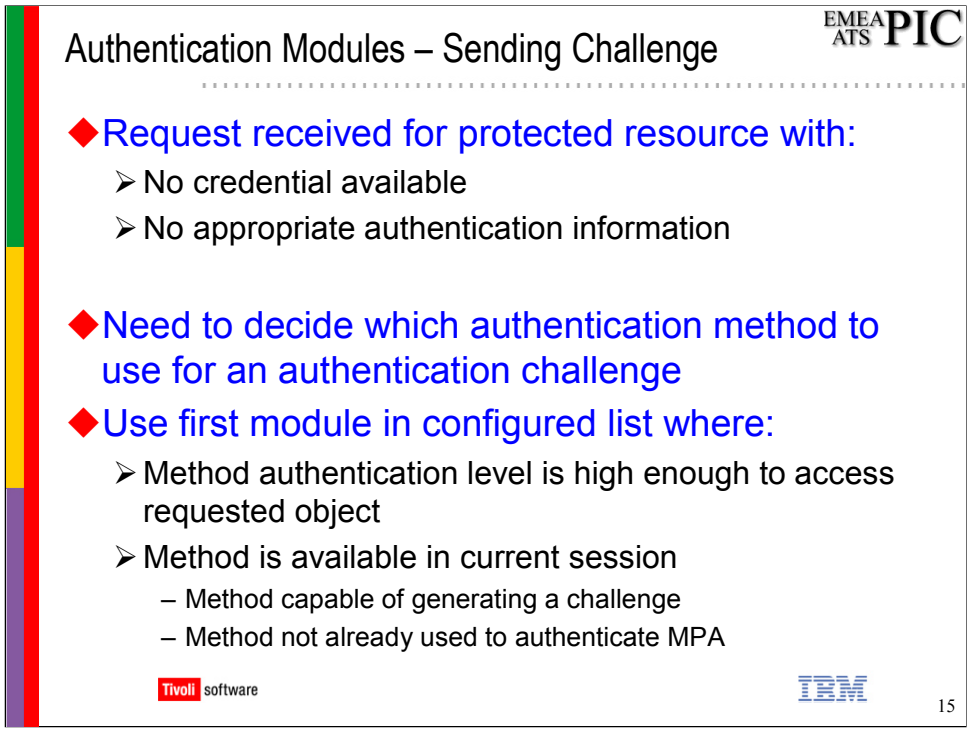
If none of the authentication modules are able to provide a validated Access Manager User then an authentication challenge will have to be sent back to the user so they can provide the information required.



The diagram above shows how the use of authentication modules insulates the Plug-in Auth Server from the different ways in which a user can be authenticated.

The plug-in simply calls each authentication module in turn. As soon as one of the modules returns a AM UserID then this is used to build a credential.

If none of the modules are able to provide a validated AM User ID then this is reported back and will mean that an authentication challenge will have to be sent to the user.

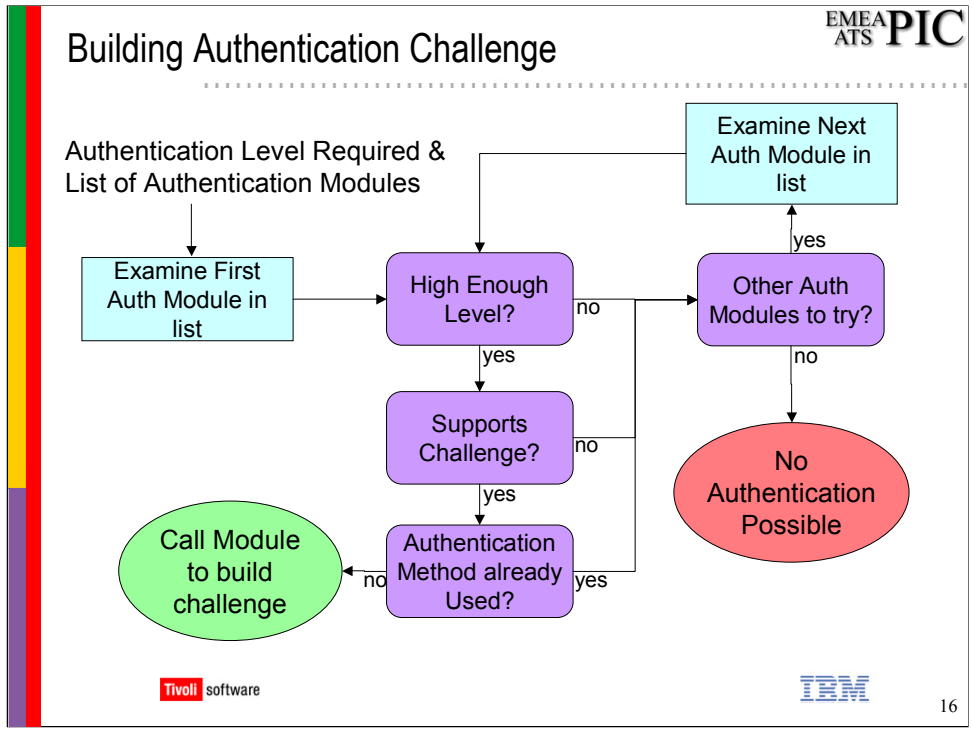


In addition to validating authentication information received in a request, an authentication module may also be able to generate an authentication challenge that can be returned to the user in order to prompt them to provide the authentication information required. Not all authentication modules can generate a challenge. For example, there is no challenge to request HTTP Headers – these are either in the request or not.

If an authentication challenge is required then the first suitable authentication module from the configured list is called to generate the commands needed (to be sent to the plug-in) to produce the challenge. As previously stated, not all authentication modules can generate a challenge. In addition, an authentication module may be unavailable because it is already being used to identify a proxy agent who is forwarding the requests.

The most common authentication mechanisms that can generate a challenge for the user are Basic Authentication (a BA challenge is sent to the user) and form-based authentication (a login form is returned to the user).

If no authentication method is available then the user cannot be authenticated. The plug-in will be instructed to return a forbidden page.

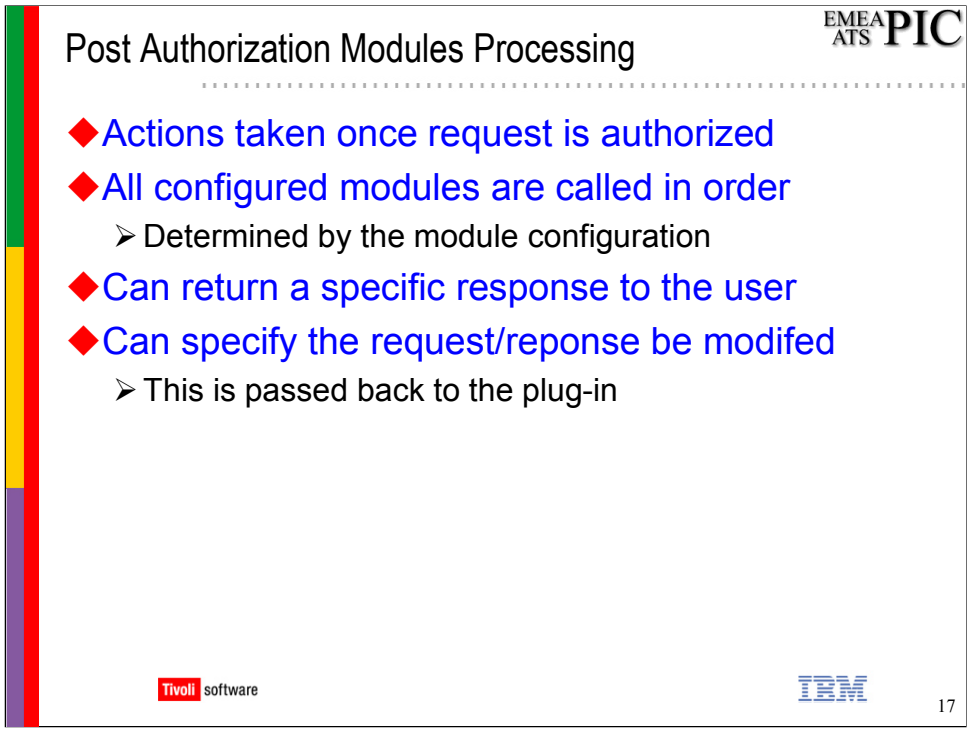


The flowchart above shows how the plug-in selects an authentication module for sending a challenge to the user.

It examines each configured authentication module, in order, until it finds one that provides the required level of authentication, is able to support sending a challenge and has not already been used to identify a proxy agent.

As soon as a module is found that meets these criteria it is called to build the challenge that will be sent back to the user.

If none of the configured authentication methods are suitable then no authentication is possible. The Plug-in Auth Server will have to return a "Forbidden" page to the user since they do not have the permissions required to access the requested resource and there is no possibility to send them a challenge to authenticate at the required level.



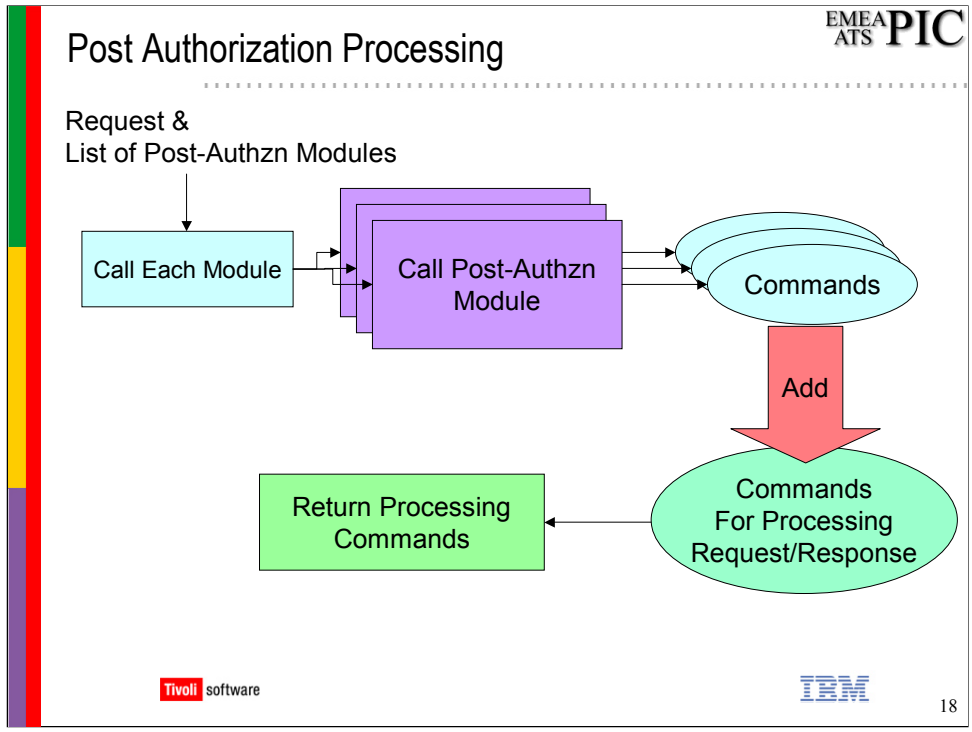
The final set of modules defined in the Plug-in Auth Server are Post Authorization Modules. These are called after a request has been authorized to determine if any other action needs to be taken before the request is passed back to the plug-in for processing by the Web Server. All of the configured Post Authorization Modules are called to see if they need to take any action.

Post Authorization modules are mainly of three types:

Modifying Request for SSO – These Post Authorization modules add information (cookies or headers) that will be used by the web application to identify the user without needing a second authentication.

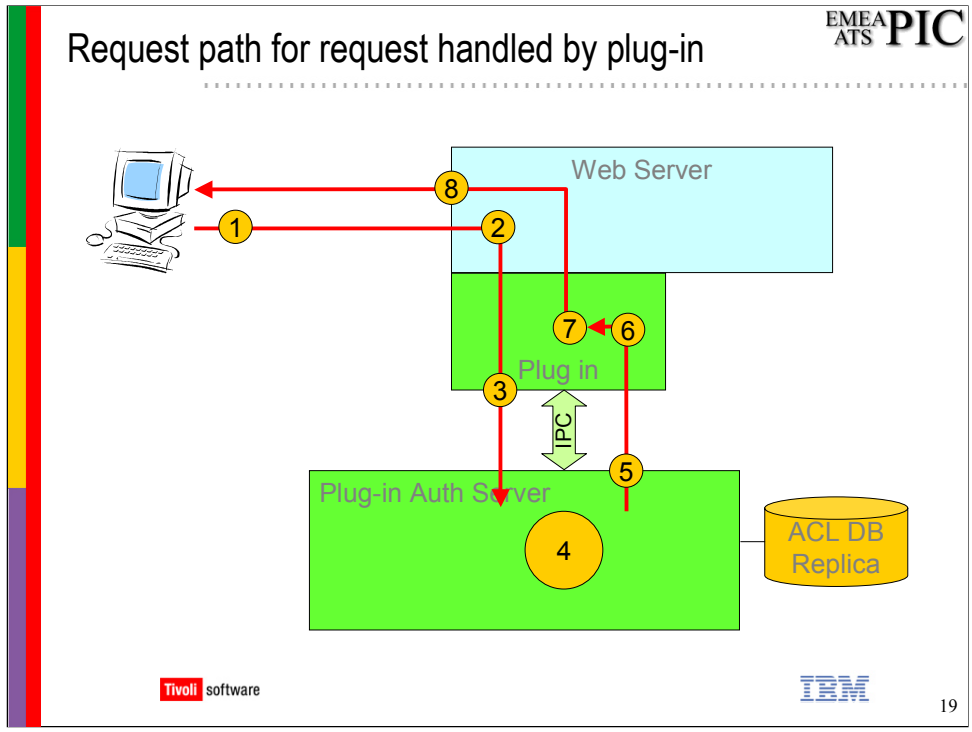
Modifying Response – These Post-Authzn modules don't modify the request but specify that the response be altered – normally by adding headers or cookies to it. For example, the failover module adds a failover cookie to responses.

Special Function – These Post-Authorization modules recognise the URI being requested as the trigger for some special function. This usually means that the request will be handled by the Plug-in Auth Server. E.g, eCSSO "vouchfor" request.



After a request has been authorized it is passed to all of the configured post-authzn modules so that they can decide if they need to take any action.

All of the commands that are returned from these modules are put together and will be sent back to the plug-in for action.



The diagram above shows the path taken by a request that is eventually handled directly by the Plug-in Auth Server (login challenge required for example)

*1 User makes request

*2 Request is received by the Web Server and passed to the plug-in code

*3 Plug-in forwards the request details over the IPC link to the Plug-in Auth Server

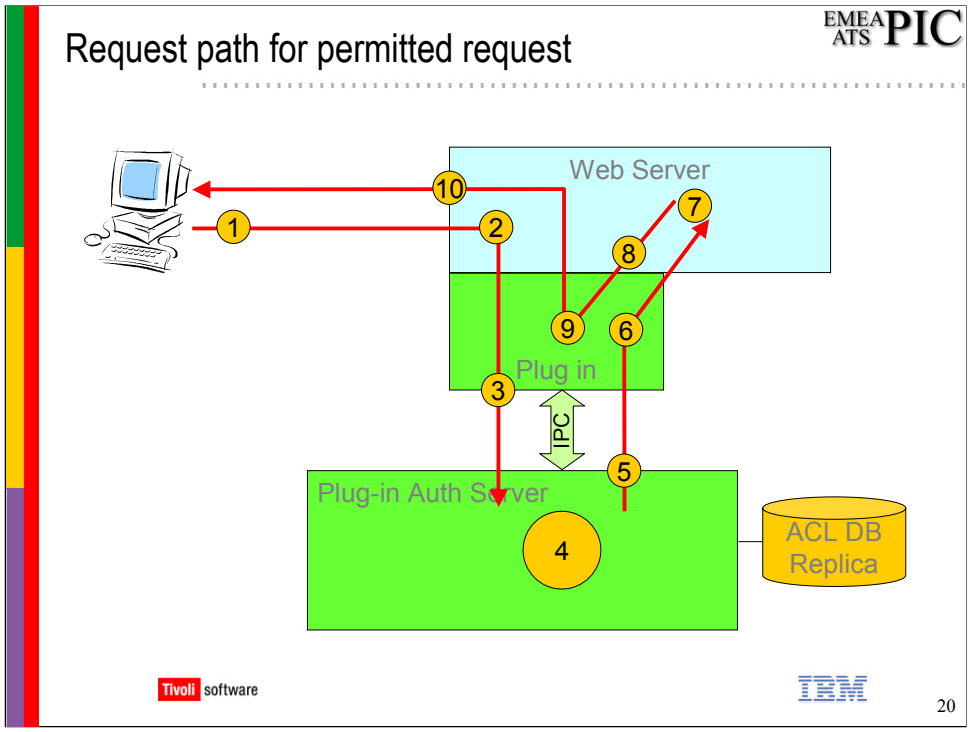
*4 The Plug-in Auth Server processes the request. It determines that a response needs to be sent to the user without going to the Web Server.

*5 The Plug-in Auth Server returns a set of commands to the plug-in instructing it to forward a fixed response to the user

*6 The plug-in receives the commands. It does not need to consult the Web Server so it forwards the request for outbound processing

*7 Any cookies and headers specified by the Plug-in Auth Server are added to the response. This is forwarded to the Web-Server for return to the user

*8 The Web Server returns the response to the User.



The diagram above shows the path taken by a request that is eventually passed to the Web Server for processing. This is likely to be a standard request for a WEB resource that resides on the Web Server.

*1 User makes request

*2 Request is received by the Web Server and passed to the plug-in code

*3 Plug-in forwards the request details over the IPC link to the Plug-in Auth Server

*4 The Plug-in Auth Server processes the request. It determines that the request is authorized and is for a resource on the Web Server.

*5 The Plug-in Auth Server returns a set of commands to the plug-in instructing it to forward the request to the Web Server.

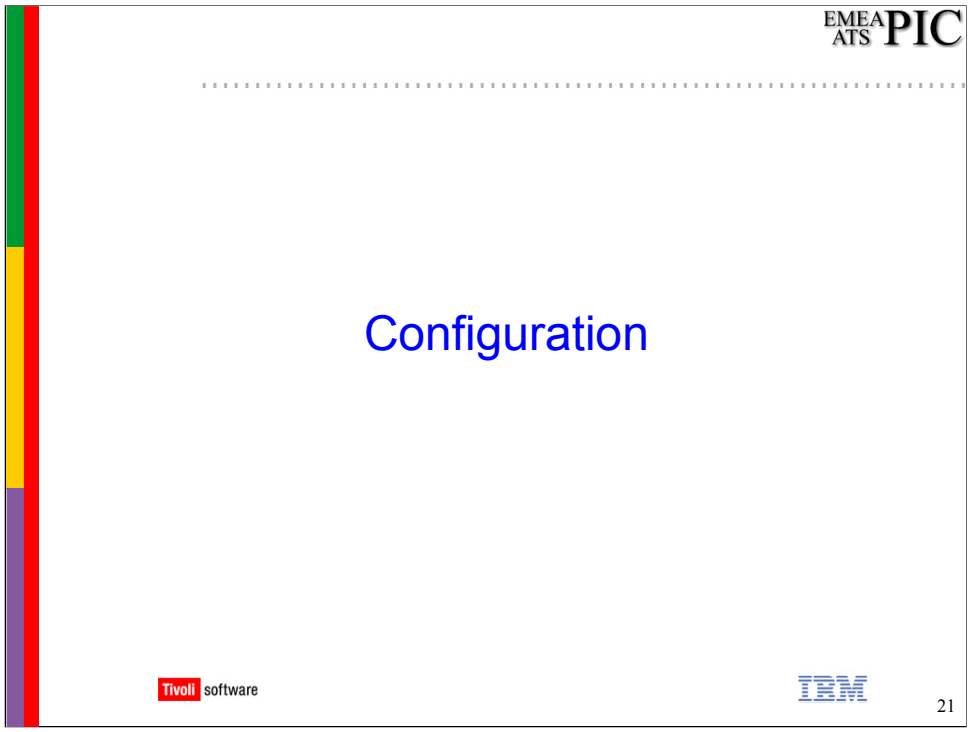
*6 The plug-in receives the commands. It makes any changes to the request that are specified and hands the request to the Web Server

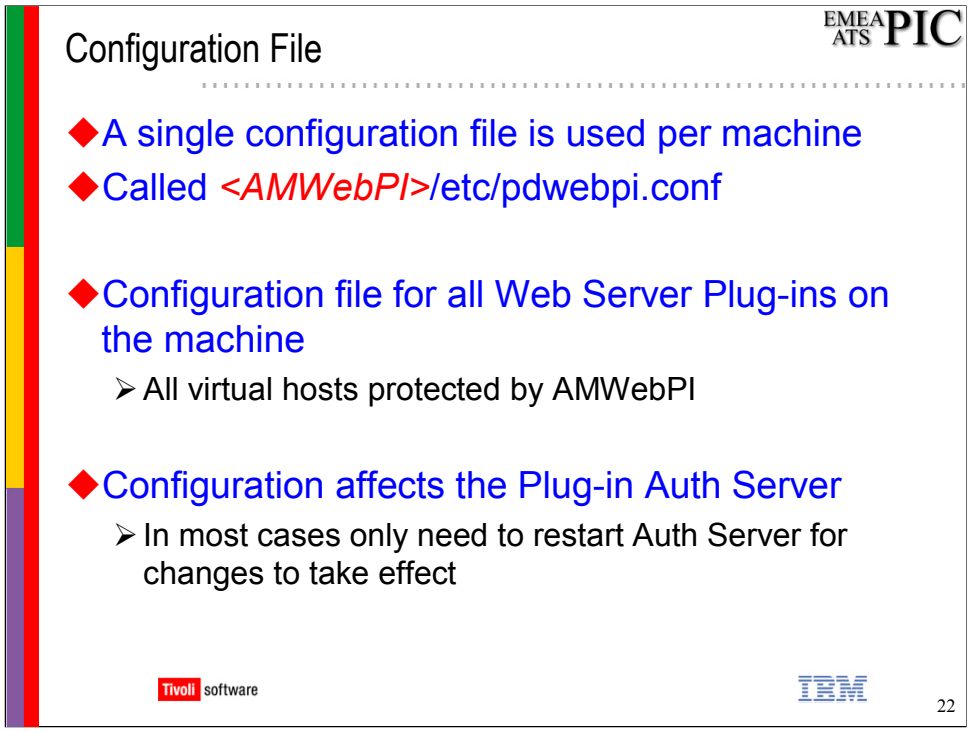
*7 The Web Server processes the request

*8 The Web Server passes its response to the plug-in.

*9 Any cookies and headers specified by the Plug-in Auth Server are added to (or removed from) the response. This is forwarded to the Web-Server for return to the user

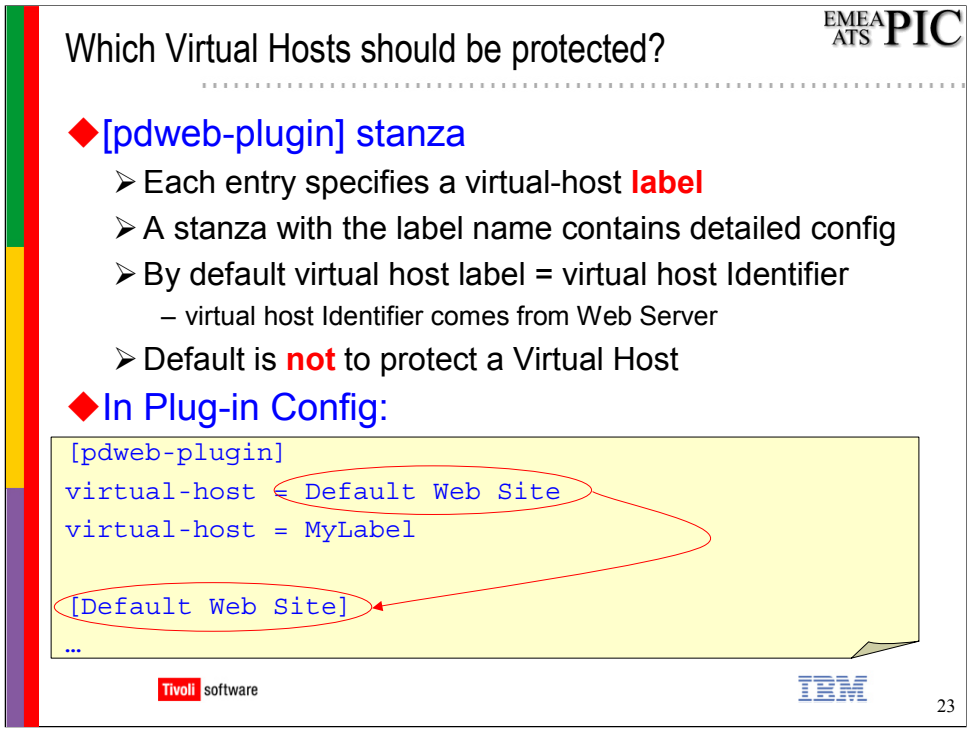
*10 The Web Server returns the response to the User.


EMEA ATS PIC
Configuration
Tivoli software
IBM
21



Only one AMWebPI configuration file is used per machine. This configuration file is used for all AMWebPI components on the machine.

Most of the configuration options only affect the Plug-in Auth Server – since it is doing almost all of the processing. In most cases only the Auth Server will have to be restarted for configuration changes to take effect – the WEB servers can continue running.

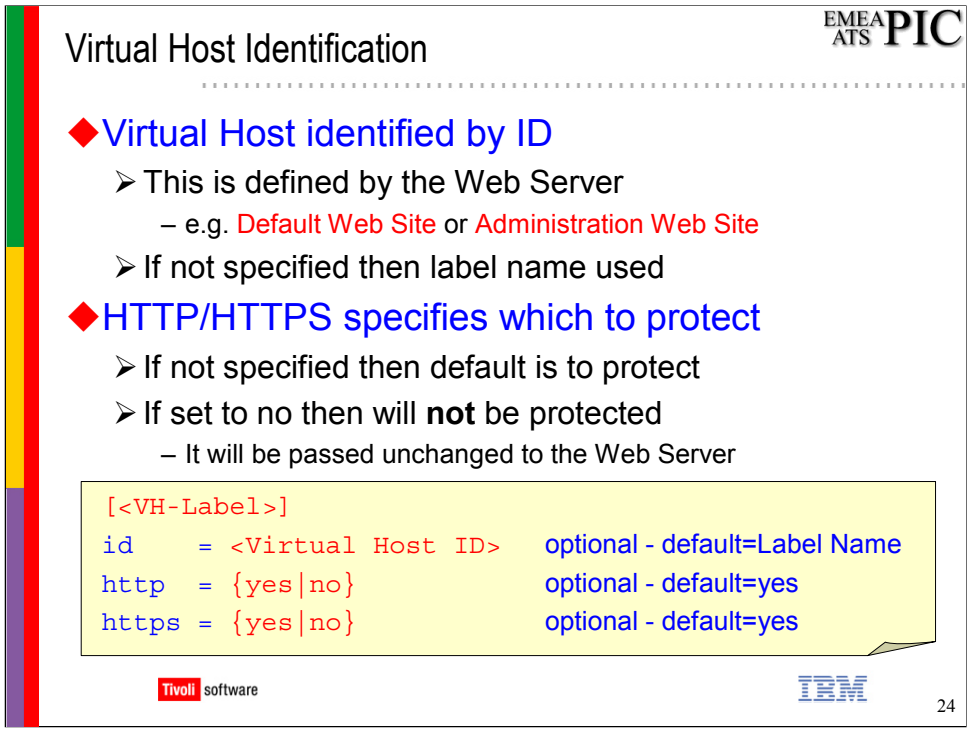


The AMWeb Plug-ins will only protect virtual hosts that are specifically configured. The default, if no configuration is found, is not to protect a virtual host – in this case requests are passed unchanged and AMWebPI offers no protection.

To protect a virtual host it must be identified in the AMWebPI configuration file. There are two parts to this operation. The first, shown above, is to create a *virtual-host* parameter in the *[pdweb-plugin]* stanza that gives a label that is used to identify the virtual host everywhere else in the configuration file.

One location it points to is another stanza (with the name specified by the virtual host label) that contains information on how to determine if requests are being made to this virtual host.

By default the virtual host label will be set to be the same as the virtual host Identifier sent by the Web Server. However this is not required; the identifier can also be specifically configured (see next page).

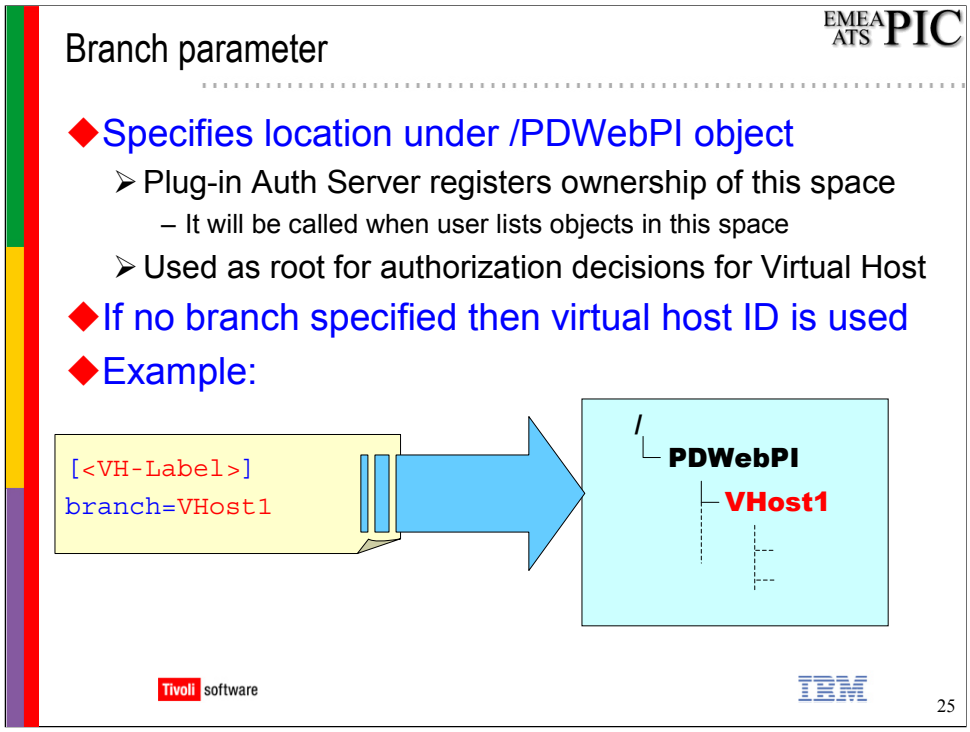


In the stanza named with the virtual host label three parameters determine which requests will be processed with the configuration for this virtual host.

The *id* parameter specifies the virtual host identifier that is set by the Web Server for requests to this virtual host. Normally this parameter is not present and the VH-Label is used instead. In this case the label must match the identifier that is sent by the Web Server. If the *id* parameter is set then the VH-Label can be set to any value.

The *http* parameter is used to specify if traffic received over HTTP should be protected. If set to no then HTTP traffic will not match this virtual host configuration and will **not be protected unless another virtual-host configuration matches it**. If the *http* parameter is not set or is set to yes then HTTP traffic will match this virtual host configuration.

The *https* parameter is used to specify if traffic received over HTTPS should be protected. If set to no then HTTPS traffic will not match this virtual host configuration and will **not be protected unless another virtual-host configuration matches it**.. If the *https* parameter is not set or is set to yes then HTTPS traffic will match this virtual host configuration.

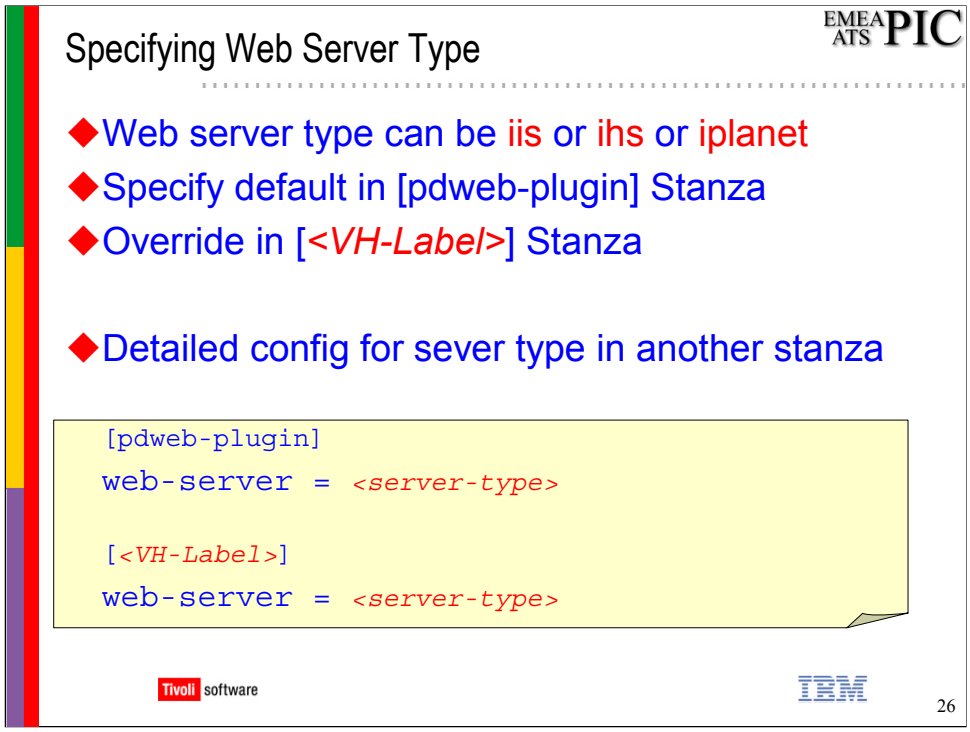


When Web Plug-ins are configured on the first machine in the AM domain an objectspace is created called */PDWebPI*. In this objectspace, entries are created for each of the virtual hosts that are being protected. Notice that there are no objects to represent servers – only virtual-hosts are listed.

The objectspace under a virtual-host object is owned by the Plug-in Auth Server that performs authorization for that virtual-host. It is registered as a dynamic objectspace by the Auth Server at start-up which means that the Auth Server will be called (through the server admin API) whenever a list of the objects in that space is required. This part of the objectspace is used by the Auth Server when authorization decisions are made for resources on the virtual host.

By default the branch of the objectspace used for a virtual-host takes its name from the Virtual Host ID. If a different branch of the /PDWebPI objectspace is to be used then the *branch* parameter is used to specify this.

Note: If the branch is changed then an object will have to be created in the with the new name. Any ACLs attached under the old branch will stay attached to the now non-existent objects under the original branch.



The *web-server* parameter is used to specify which type of Web Server a virtual host resides on. This is required in order for the output from directory queries to the server to be interpreted correctly for "query_contents" type operations.

By default the *web-server* parameter is set in the *[pdweb-plugin]* stanza which specifies this sever type for all virtual hosts on the machine. This is OK for the current version (where only one server type is supported on each machine) but when multiple server types are permitted this can be over-ridden by specifying the server type in the virtual-host stanza (stanza named with the VH-label).

URL Properties - 1

EMEA ATS PIC

- Windows Filesystem
  - Disallow DOS shortnames (e.g. progra~1)
  - Prevent access via W2K Short Paths
- Case Sensitive Filenames ?
  - Always convert URL to lowercase before checking
- Set globally and overridden by Virtual Host

```
[pdweb-plugin] or [<vh-label>]
windows-file-system = true
case-sensitive = false
```

Tivoli software

IBM

27

URL Properties - 2

EMEA ATS PIC

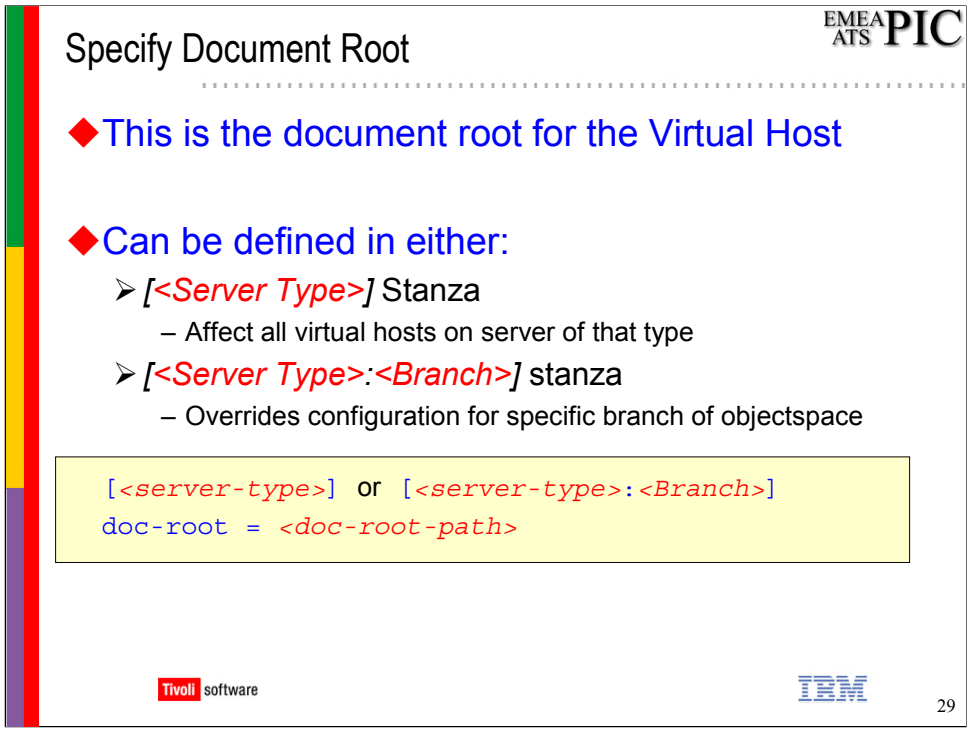
- How Are URLs encoded?
  - UTF-8 - Must be converted to local codepage
  - Non UTF-8 - Assumed to be in local codepage
- Can configure for either or set to “auto”
  - Assumed UTF-8 unless invalid characters present
- Set globally and overridden by Virtual Host
  - Default is TRUE

```
[pdweb-plugin] or [<vh-label>]
utf8-url-support-enabled = auto
```

Tivoli software

IBM

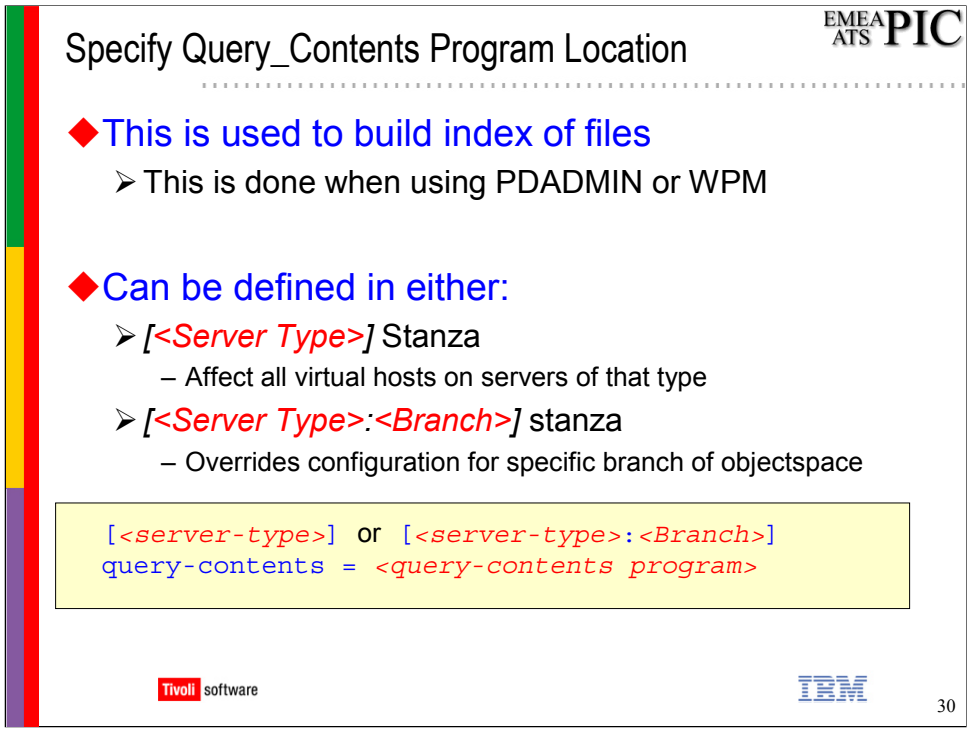
28



For each virtual host you need to configure the document root. This is used by the Plug-in Auth Server to respond to requests from the management server to list the objects that it is protecting (for display in the objectspace).

The *doc-root* parameter can be defined for all virtual hosts on a given server by specifying in the <*server-type*> stanza (e.g. [ihs]) or can be specified for a particular objectspace branch by specifying in a stanza qualified with the virtual-host label (e.g. [ihs:MyBranch])

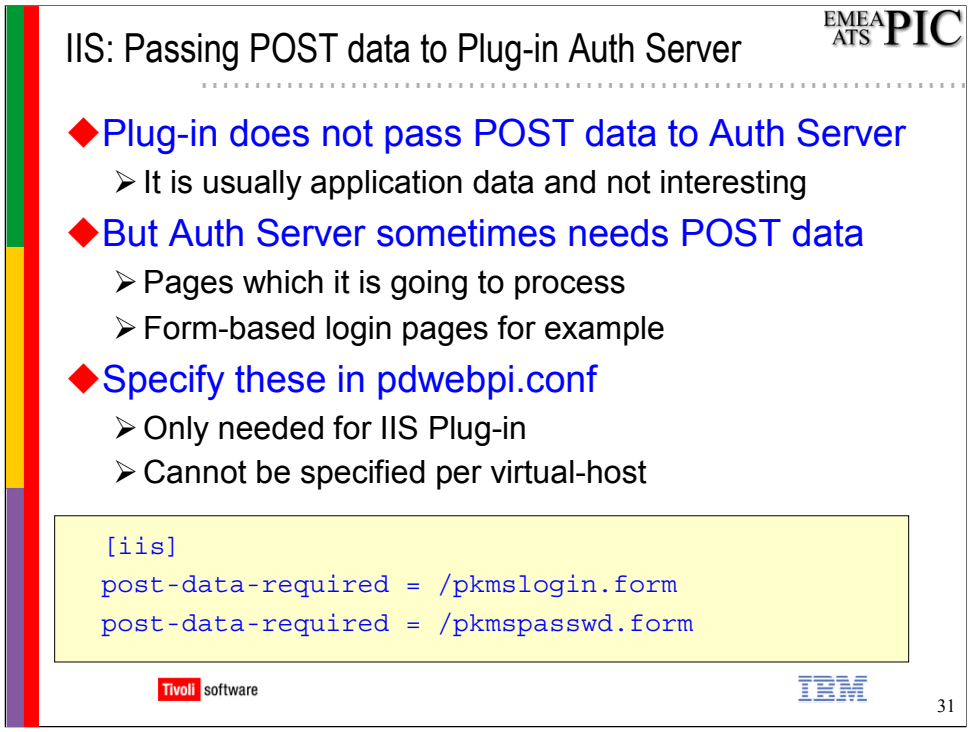
Note: The document-root parameter is only required for IHS and iPlanet Web Servers.



For each virtual host you can configure an application that will be called by the Plug-in Auth Server to retrieve a list of the web-server objectpace. This is rather like the "query_contents" application that WebSEAL uses except that it is run locally rather than being invoked via HTTP. This application is called by the Plug-in Auth Server to respond to requests from the management server to list the objects that it is protecting (for display in the objectspace).

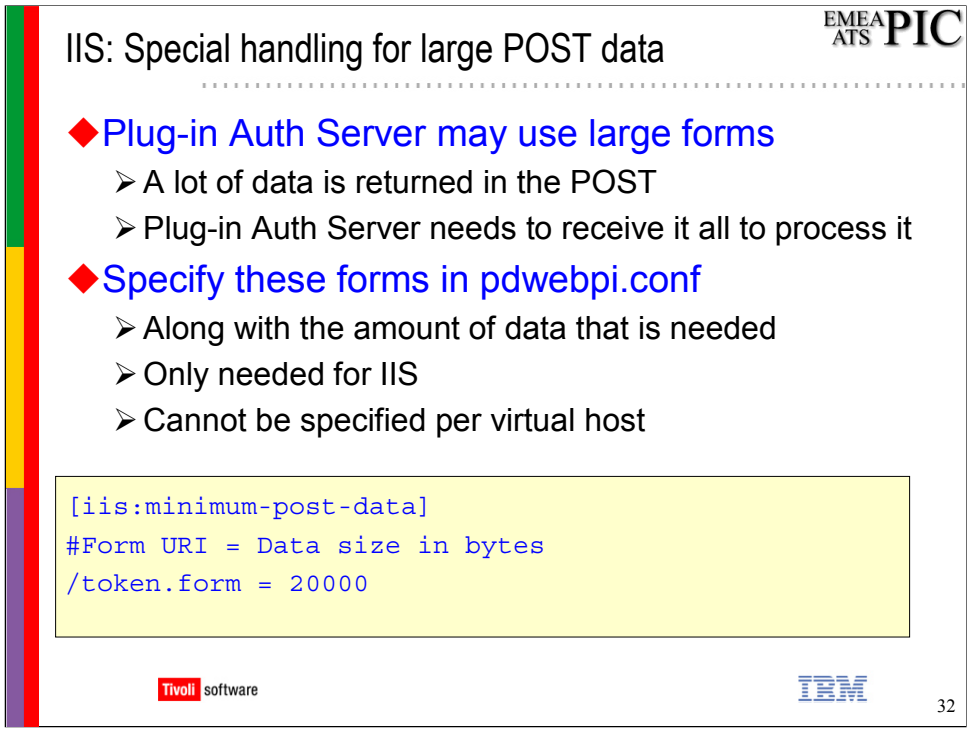
A custom application can be provided by the Customer if required.

The *query-contents* parameter can be defined for all virtual hosts on a given server by specifying in the <*server-type*> stanza (e.g. [ihs]) or can be specified for a particular objectspace branch by specifying in a stanza qualified with the virtual-host label (e.g. [ihs:MyBranch])



The configuration item shown above is specific to IIS servers. By default the IIS plug-in does not forward POST data to the Plug-in Auth Server. This information is needed in some cases (e.g. when it is a login form POST) and so it is possible to define a list of URLs that indicate a page where the body of the request should be forwarded to the Plug-in Auth Server.

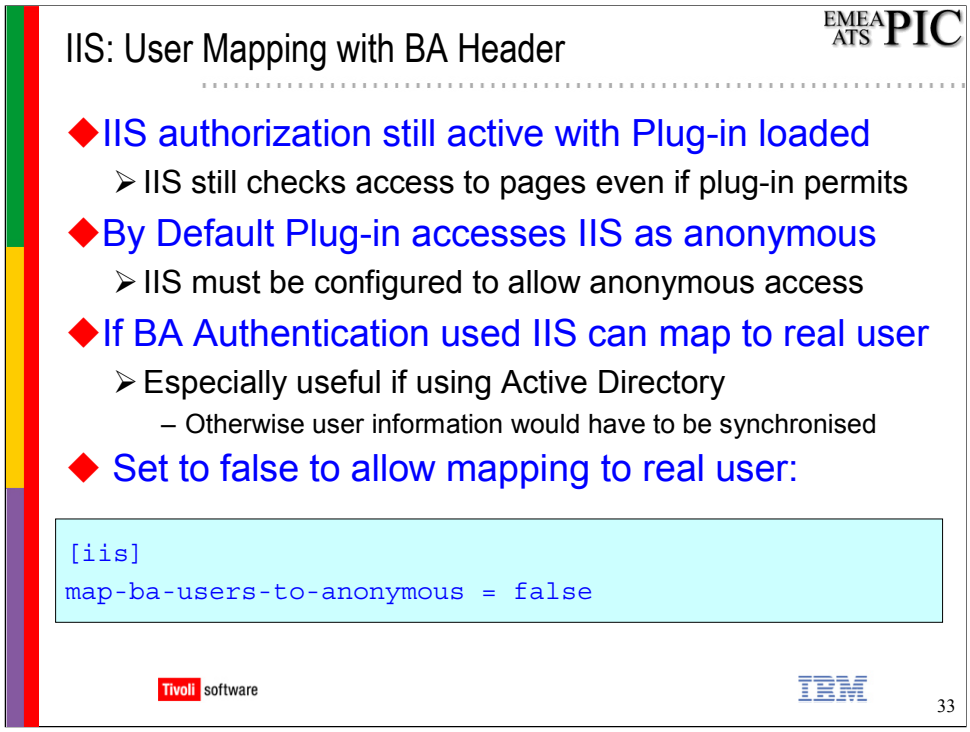
Note: In IHS and iPlanet this configuration must be done in the Web Server configuration files. The configuration changes required for out-of-the-box forms based login (including token login) are performed during configuration of the AMWeb Plug-in on the machine.



The configuration item shown above is also specific to IIS servers. In the case where a large amount of POST data is required by the Plug-in Auth Server in order to authenticate a user (for example) it is possible to specify this in the configuration. This configuration tells the plug-in that when it sees a POST to the URL configured it needs to send at least the amount of POST data configured to the Plug-in Auth Server.

This configuration option is configured in a stanza called *[iis:minimum-post-data*]. It is not configurable on a per virtual-host basis.

Note: In IHS and iPlanet this configuration must be done in the Web Server configuration files. The configuration changes required for out-of-the-box forms based login (including token login) are performed during configuration of the AMWeb Plug-in on the machine.



Even with the AM Plug-in loaded, IIS still has to authorize requests.

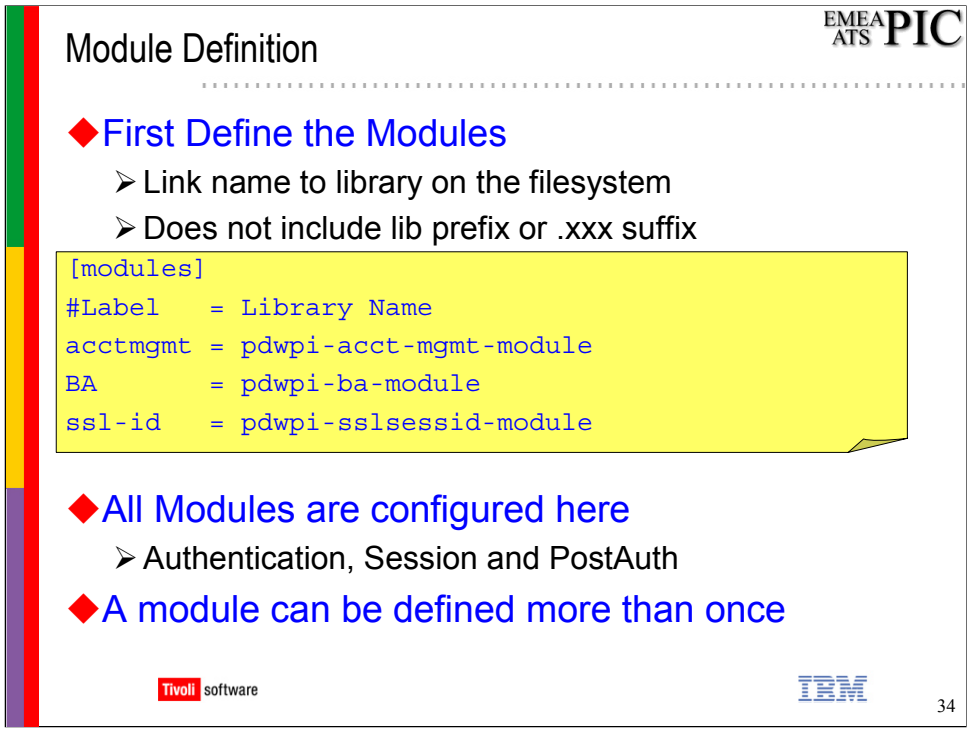
By default, the plug-in passes all requests to the IIS code as anonymous. This means that IIS must be configured to allow anonymous access which, in turn, means that all authorization is handled by Access Manager.

Unauthenticated access can be configured in IIS either during configuration of a new virtual host or in the Directory Services tab of a virtual host or the Master WWW properties.

If IIS authorization is still required then it is possible (for BA authentication) to have the plug-in pass the BA header to the IIS code so that it can authenticate (and authorize) the actual user making the request. This might be particularly desirable in an Active Directory environment where IIS and Access Manager are sharing the same user registry.

To enable the plug-in to send the BA information on to the IIS code turn off the default behaviour of mapping all users to unauthenticated in the [iis] stanza of pdwebpi.conf as shown above.

This is a global configuration parameter - it cannot be set per virtual host.



There are three steps to configuring modules. The first is to define the modules (of all types) that are available. This is a global configuration that provides a label to represent each module library.

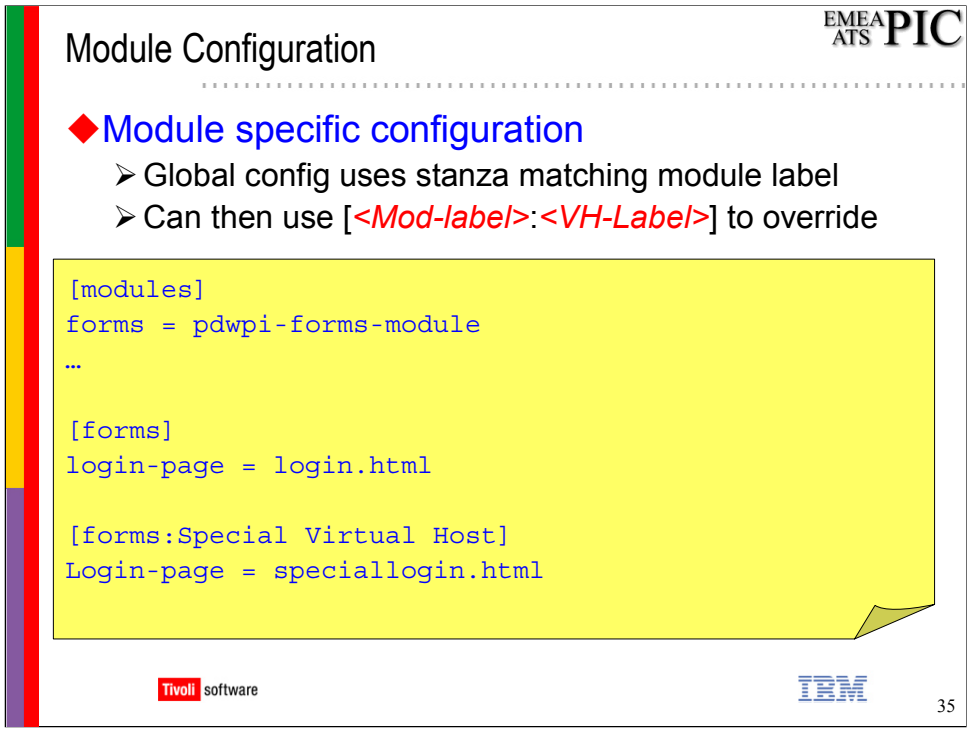
Modules are defined in the *[modules]* stanza and each definition consists of a label and a library name. The libraries must exist in the *<pdwebpi>/lib* directory and the filename given is with any OS specific prefix (e.g. lib) and any OS specific suffix (e.g. .dll) removed.

In the example above the BA module library is given as *pdwepi-ba-module*. On Windows the Plug-in Auth Server will look for a file called *pdwebpi-ba-module.dll*. On Solaris it will look for a file called *libpdwebpi-ba-module.so* and on AIX it will look for a file called *libpdwebpi-ba-module.a.*

The label names defined here are used in the following configuration to identify the modules.

A library can be defined more than once with different labels. This allows it to have different configurations for use in the same virtual host.

Note: It is possible to configure the path that is used when searching for module libraries. This is the *path* parameter in the *[module-mgr]* stanza.

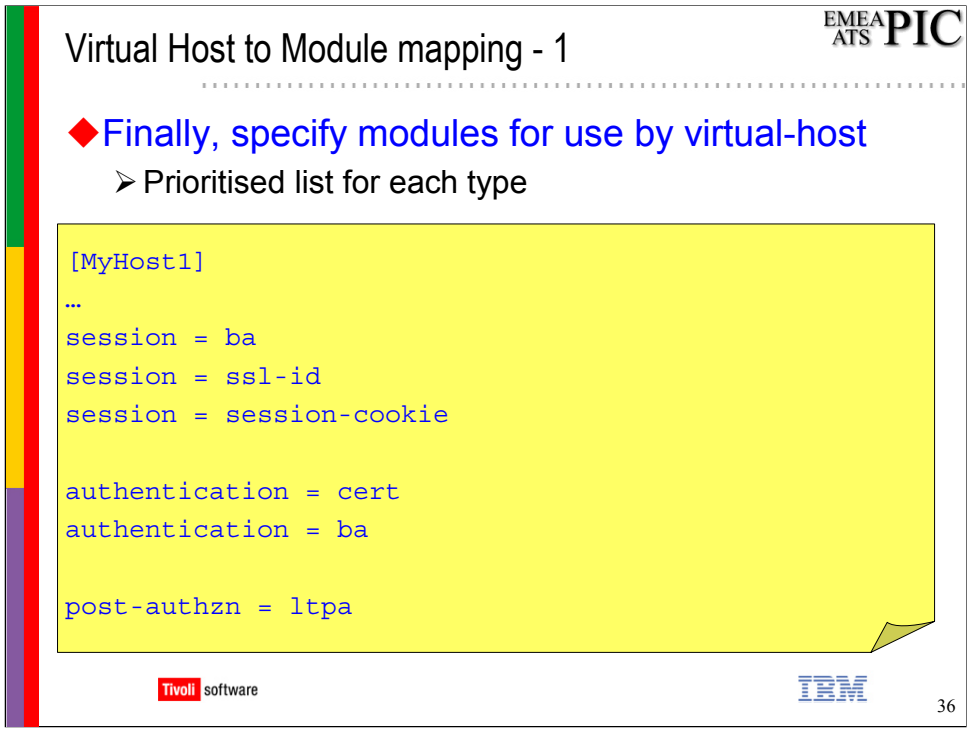


The default configuration options specific to a module are configured in a stanza that is named using the module label described on the previous page. This configuration affects that module no matter which virtual-host it is called from.

If special configuration is required on a per virtual-host basis then the default configuration can be overridden by using a stanza that qualifies the module label with a virtual-host label. This seen in the example above.

If any virtual-host except the one with label “Special Virtual Host” uses forms-based login then the user will receive login.html.

If the virtual-host with label “Special Virtual Host” uses forms-based login then the user will receive speciallogin.html



The last step for module configuration is to specify which modules will be used (in preference order) for each virtual host. This can be done in a variety of ways.

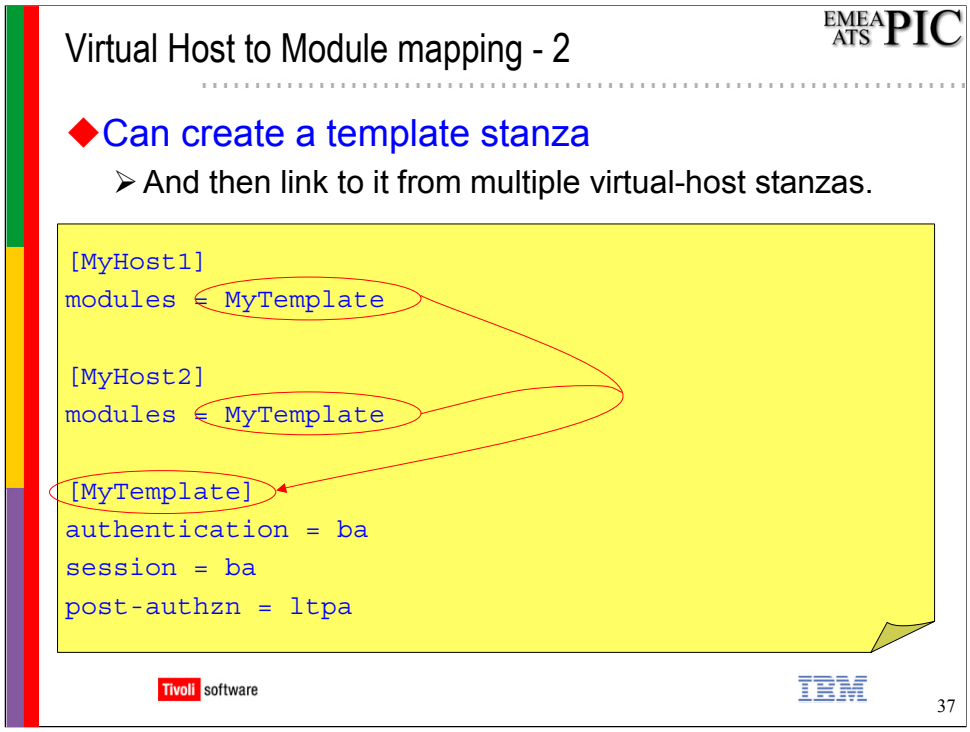
The first way, shown above, is to specify the modules directly in each virtual-host stanza.

Modules are specified in preference order by type as shown above. The syntax of each entry is:

<module-type> = <module label>

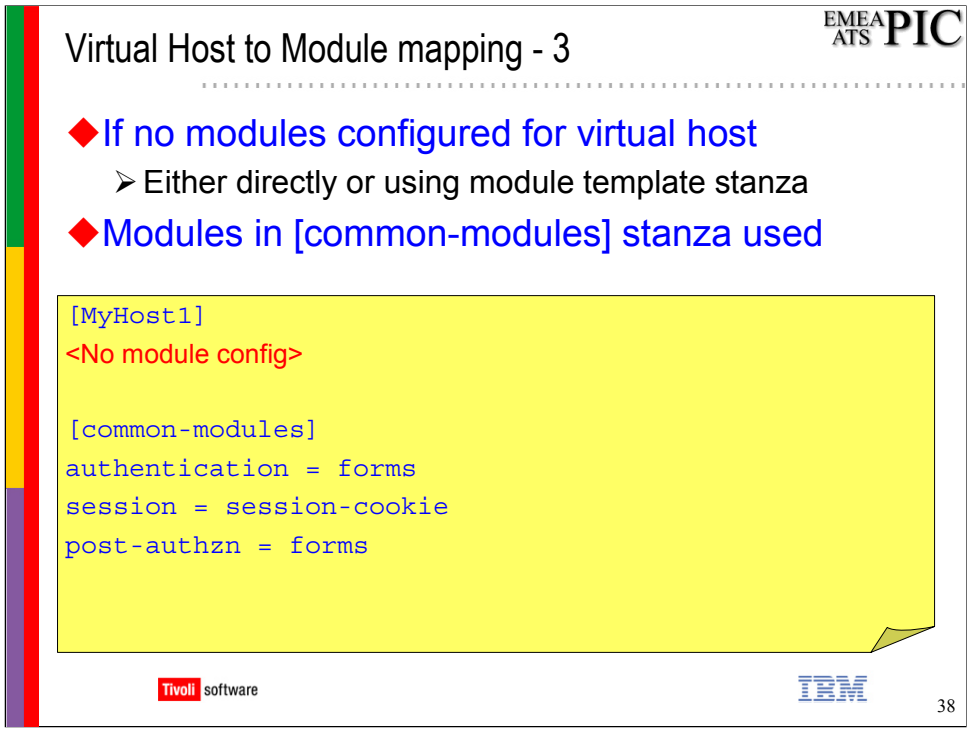
Where *<module-type* is *{session|authentication|post-authzn}*

Some modules require that other modules are present. For example if *forms* is specified as an authentication module it must also be specified as a post-authzn module. If mismatches occur then errors will be logged when the Plug-in Auth Server attempts to start.



An alternative way to specify virtual-host to module mapping is to specify a stanza for module configuration in the virtual-host configuration. This allows multiple virtual-hosts to share a module configuration.

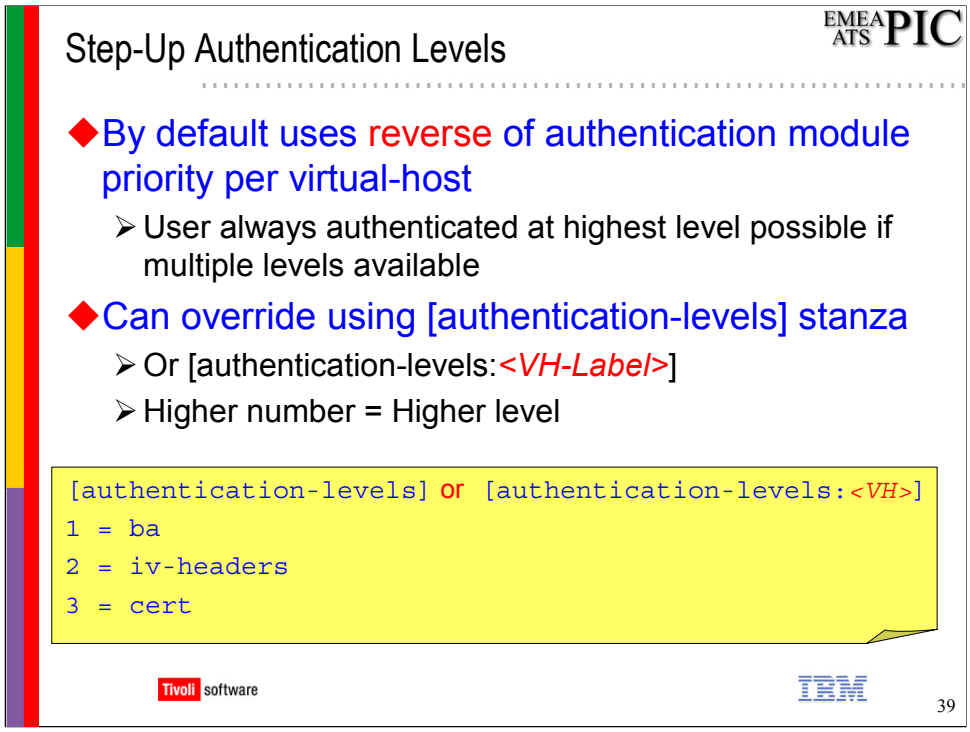
The module configuration stanza is specified by the *modules* parameter in the virtual-host stanza as shown above.



The default out-of-the-box configuration is that there is no virtual host configuration in any of the virtual-host stanzas.

In this case virtual-host to module mapping for all virtual-hosts is taken from the *[common-modules]* stanza. In this case changes to the *[common-modules]* stanza will affect all virtual-hosts.

Even some virtual-hosts are modified to have modules directly configured or others specify a module template stanza, any virtual-host that has no module configuration will continue to take its mapping from the *[common-modules]* stanza.



The Web Plug-ins support Authentication Levels and authentication step-up in the same way as WebSEAL.

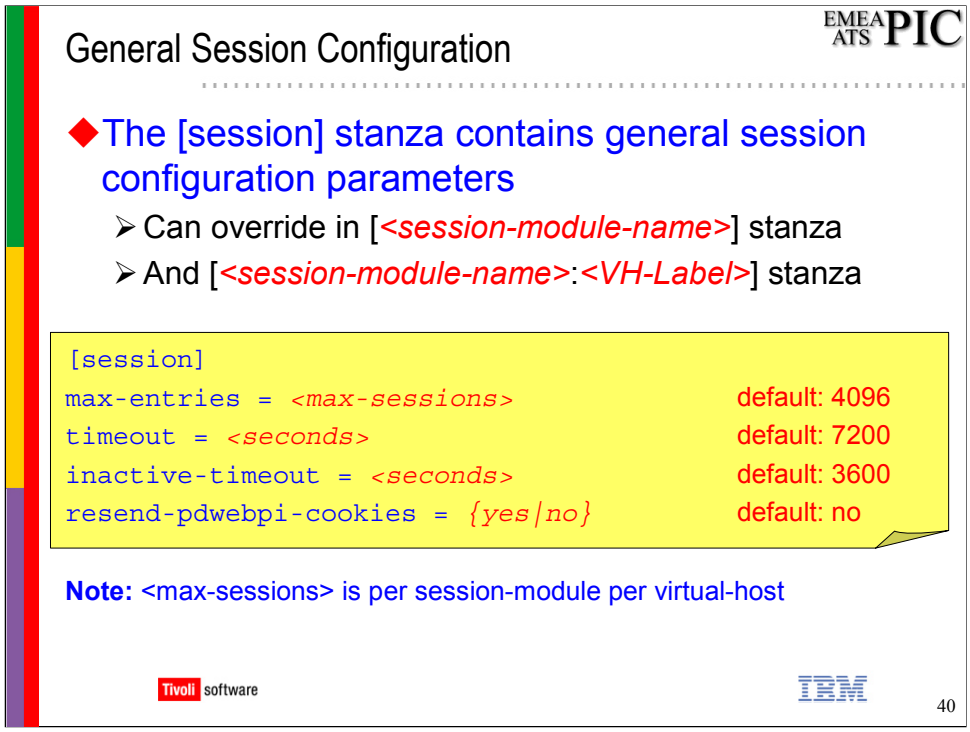
By default, if no authentication-level configuration is given, the authentication levels for a virtual-host will be set to the **reverse** of the authentication module priority. The means that the user will always initially authenticate at the highest possible level. The only reason a user will not authenticate at a high level is if that method is not available (e.g. Certificates not exchanged at SSL negotiation).

If this order is not what is required then the authentication levels can be specified globally and on a per virtual-host basis as shown above. The syntax for an entry is:

<level> = <authentication module label>

The higher the value of *<level>* the higher the authentication level.

Step-up can only be supported to authentication modules that are capable of generating a challenge. This means that step-up is supported to password and token authentication.



As mentioned previously, each session module maintains its own session cache. This means that each session type can have its own configuration in terms of timeouts.

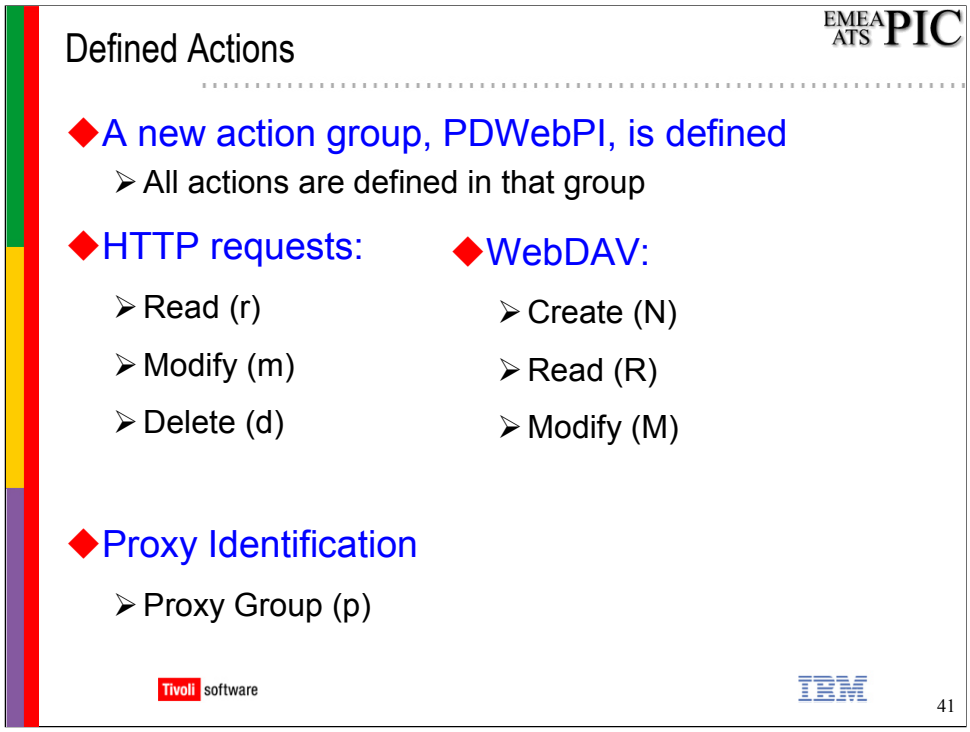
By default all session modules share a common configuration defined in the *[session]* module as shown above. This can be overidden per session module type (in a stanza such as *[forms]*) and from there by virtual-host (in a stanza such as *[forms:MySite]*).

max-entries - defines the number of entries in the session cache per virtual-host. This means it defines the maximum number of concurrent sessions. Note that this is per session module and per virtual-host.

timeout - This is the maximum length of a session. When this time expires the user must authenticate again and a new credential is built.

inactive-timeout – This is the amount of time a session can be idle before it is deleted from the session cache. The user must authenticate again.

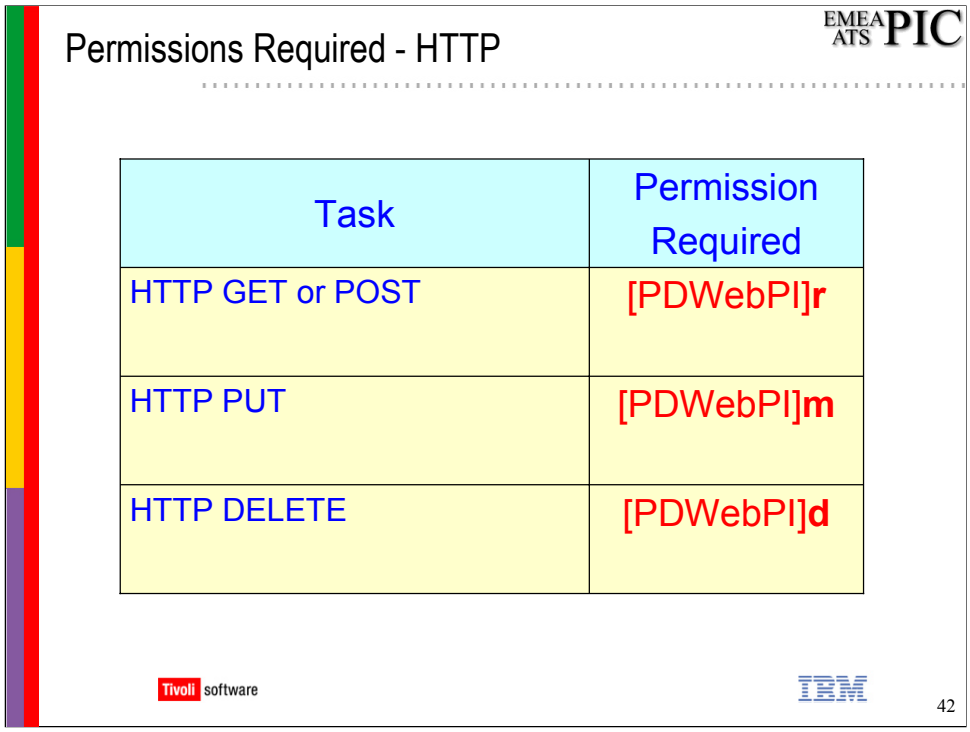
resend-pdwebpi-cookies – This indicates whether session cookies (when used) should be sent with every response to ensure they do not age out of the browser.



AM Web Plug-ins define a new action group called [PDWebPI] that contains 7 new actions.

The [PDWebPI] action group and these actions are created the first time that Web Plug-ins are configured on a machine.

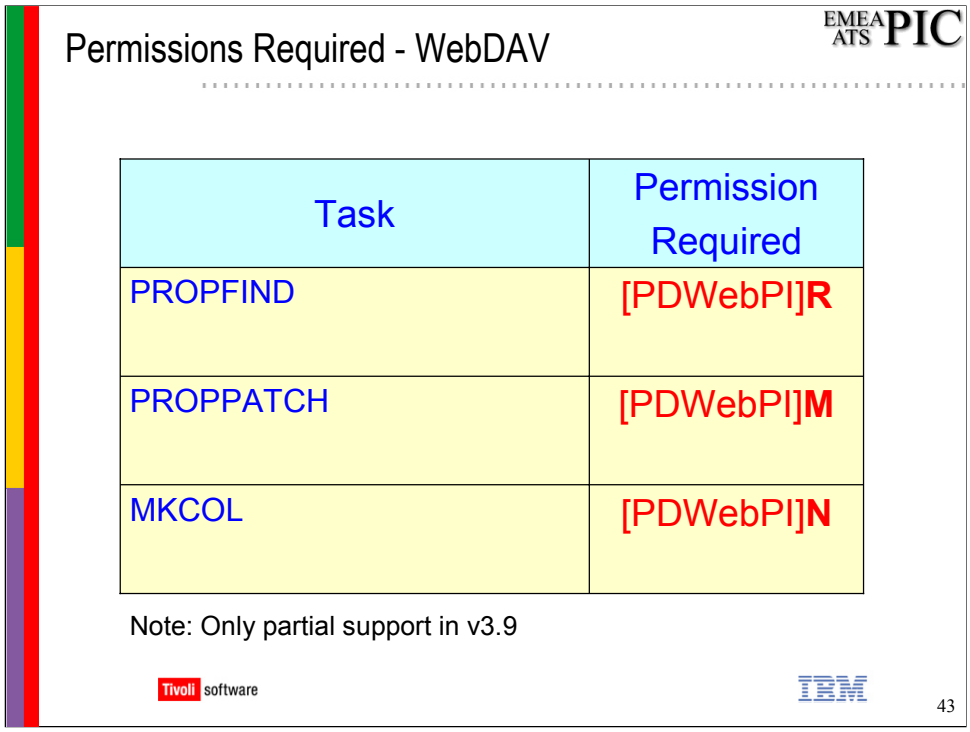
The permissions are described in detail on the following pages



Any HTTP GET or POST request requires the **[PDWebPI]r** permission. There is no specific "list" permission for requesting a directory listing (A GET of a URL ending in /) – this is also checked with the **[PDWebPI]r** permission.

An HTTP PUT request requires the **[PDWebPI]m** permission.

In order to execute an HTTP DELETE command the **[PDWebPI]d** permission is required.

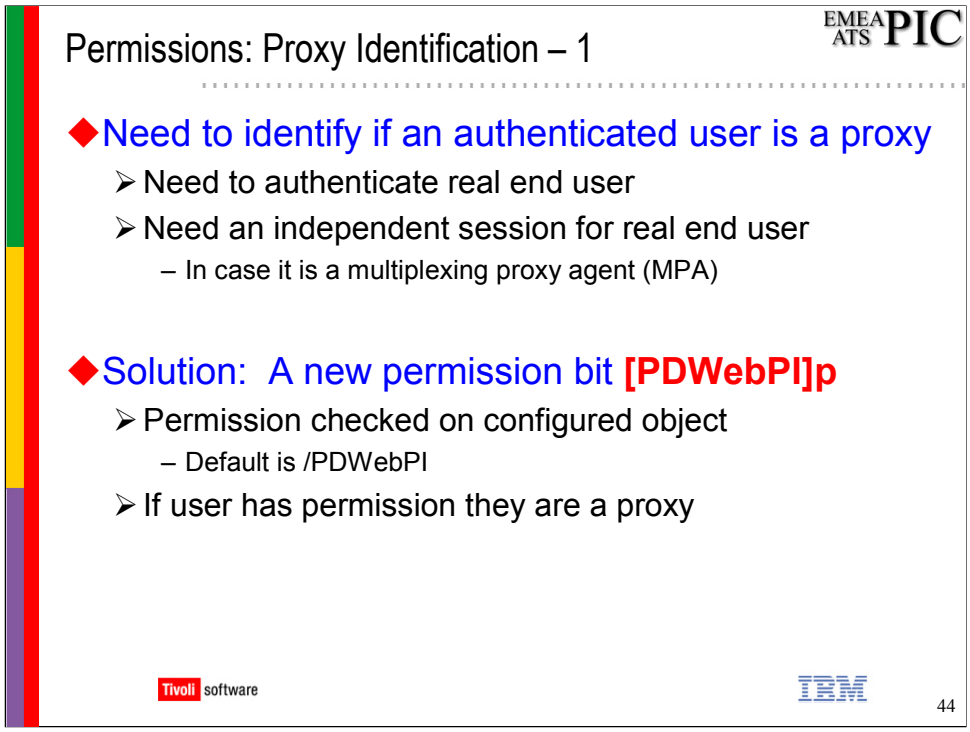


AMWebPI also supports WebDAV operations as shown above. WebDAV operations are authorized based on the request URI – not on individual members of a collection.

In addition, some other WebDAV operations are partially supported:

COPY: Requires **[PDWebPI]R** on the collection to copy from in order to read it. Permissions for the destination are not checked.

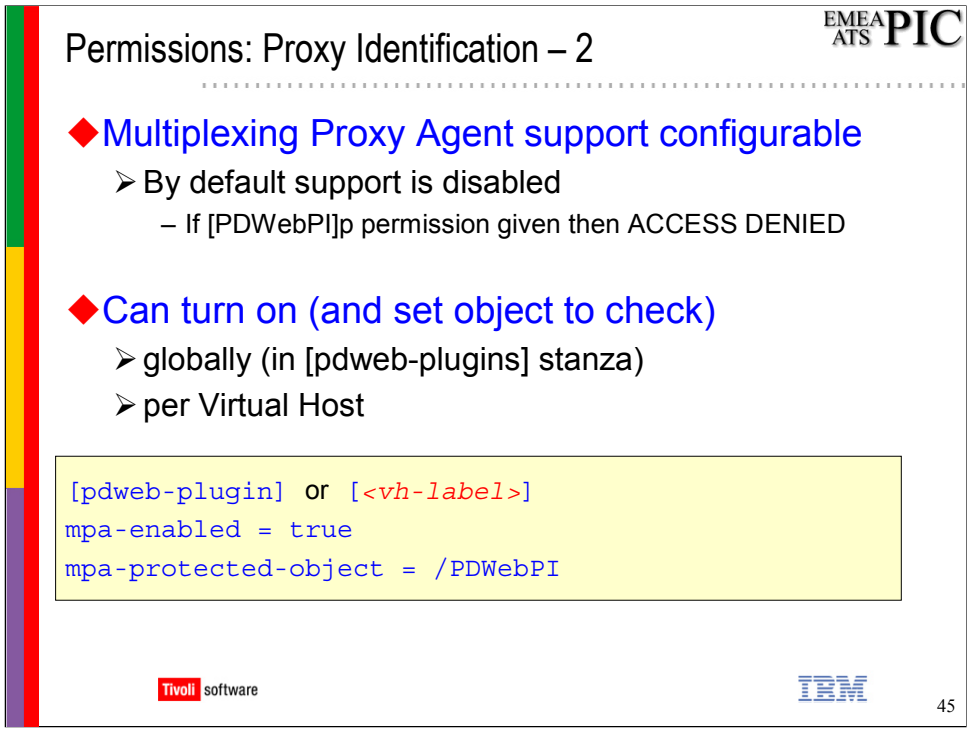
MOVE: This is considered a copy then a delete. Requires **[PDWebPI]Rd** on the collection to move from. Permissions for the destination are not checked.



The Plug-in Auth Server needs to be able to identify when an authenticated entity is actually a multiplexing proxy agent (MPA). This is important because they may be many real end-users behind that single proxy with all requests coming over the same session. The Auth Server needs to be able to authorize access based on the end-user – not on the proxy.

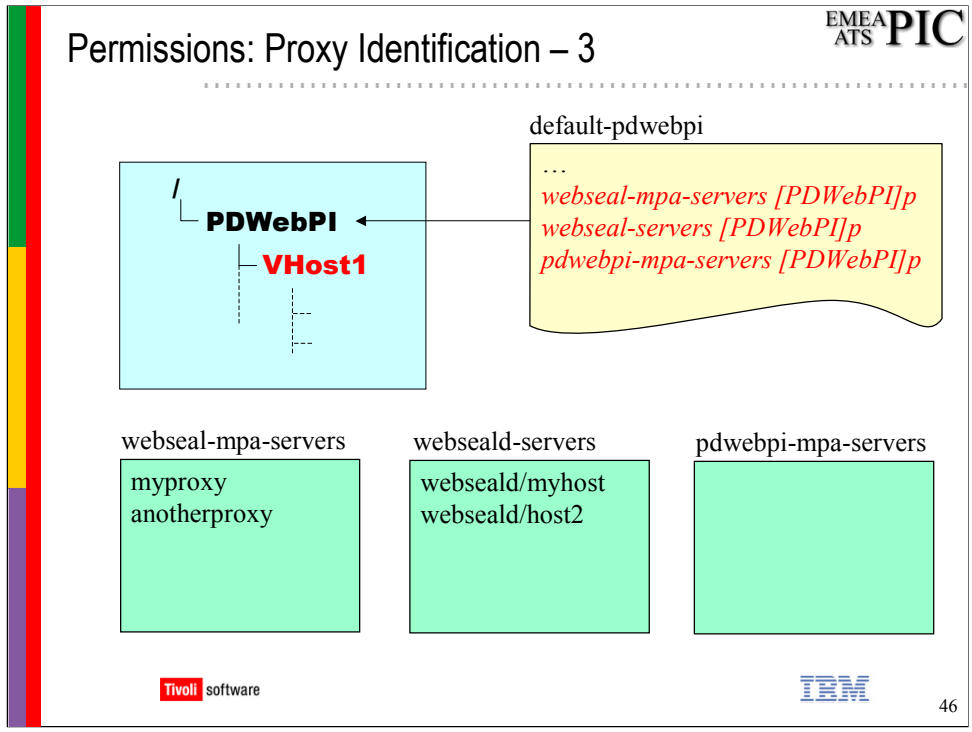
When a new user authenticates their permission to perform [PDWebPI]p action on configured object (default is /PDWebPI) is checked. If they are authorized then this user is identified as a proxy and the appropriate action can be taken.

If a user turns out to be an MPA then first of all it has to be decided if access via MPA is allowed (see next page). If MPA access is permitted then all access is authorized using the unauthenticated credential. If a protected page is requested then a secondary authentication is performed that is directed to the real end-user. This might mean sending the end user a form-based login request or it could mean that the end user is identified using some header information set by the proxy (this requires that the proxy be trusted to perform authentication on our behalf).

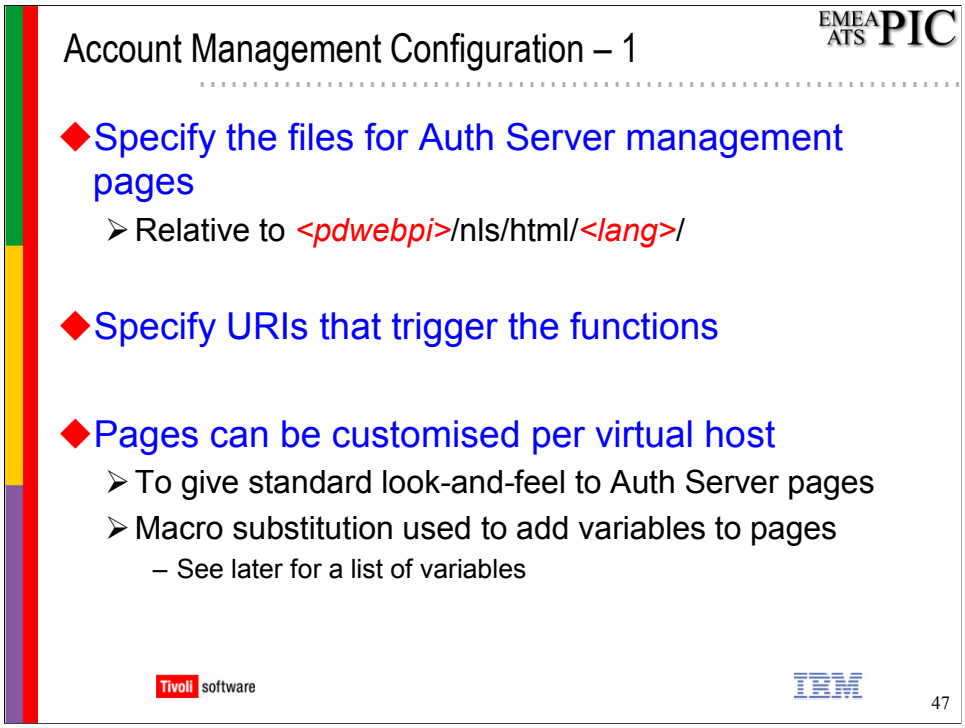


By default support for Multiplexing Proxy Agents (MPAs) is disabled. This means that if a proxy connects (identified by the [PDWebPI]p permission) then it will be denied access.

The configuration allows proxy access to be granted globally or per virtual host and allows the object to use for proxy checks to be specified (also globally or per virtual host).



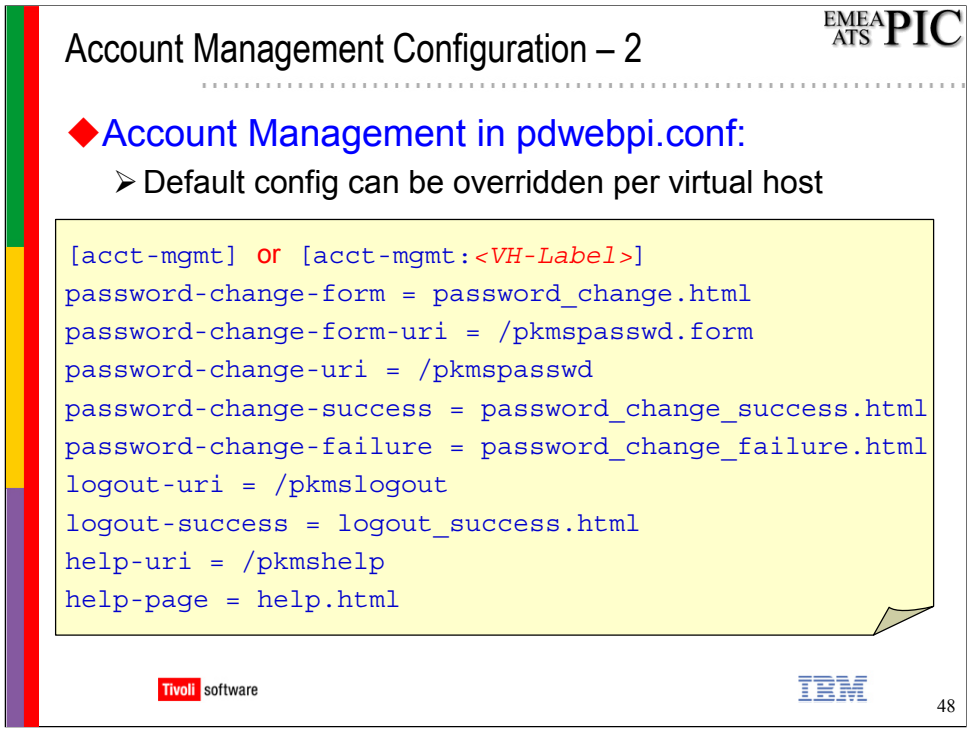
The diagram above shows how the [PDWebPI]p permission is used to identify a proxy. It shows the default configuration which means that members of the *webseald-mpa-servers*, *webseal-servers* and *pdwebpi-mpa-servers* groups are considered proxies.



Account Management can be customised by changing the URIs that trigger the account management functions and changing the files that are returned to the user for some of these functions.

This customisation can be done per virtual-host so each virtual-host can have management pages that have the appropriate look and feel.

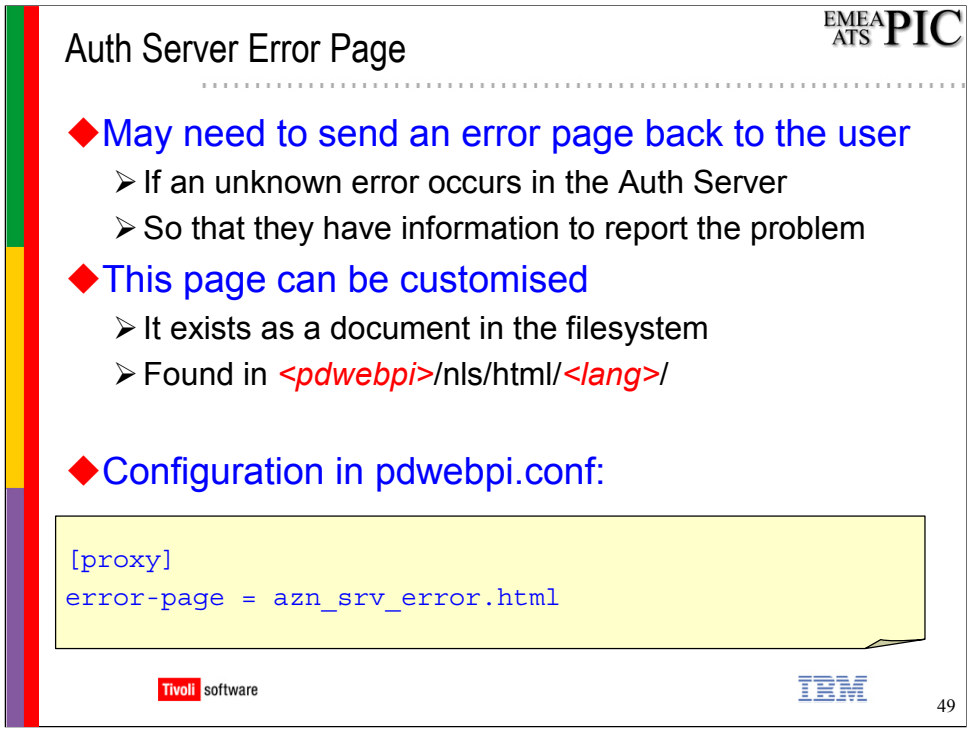
Macro substitution is used to include appropriate information into the pages before they are returned to the user. See later for a list of variables that can be used in the pages.



The configuration options shown above are those that control account management. These can be configured globally and overridden on a per virtual-host basis.

All of the filenames given specify files that are found relative to the path: <*pdwebpi*>/nls/html/<*lang*>/

The <*lang*> variable is taken from the NLS configuration. In an US English installation this will be set to '*C*'.



If the Auth Server encounters an error that it cannot handle which prevents it from processing a request then it needs to return an error page to the user. This error page can be customised and it's location is specified in the *[proxy]* stanza as shown above.

This file must be located in the <*pdwebpi*>/nls/html/<*lang*>/ directory.

The <*lang*> variable is taken from the NLS configuration. In an US English installation this will be set to '*C*'.

Variable substitution can be used in this page to provide more information on the error to the user so that the y can report it more effectively. Variables available are shown on the next page.

There is also *acct-locked-page* and *retry-limit-reached-page* pointers in this stanza that determine the message sent to the user if the user account has been locked out or suspended due to excessive login failures.

Macro Substitution

EMEA ATS PIC

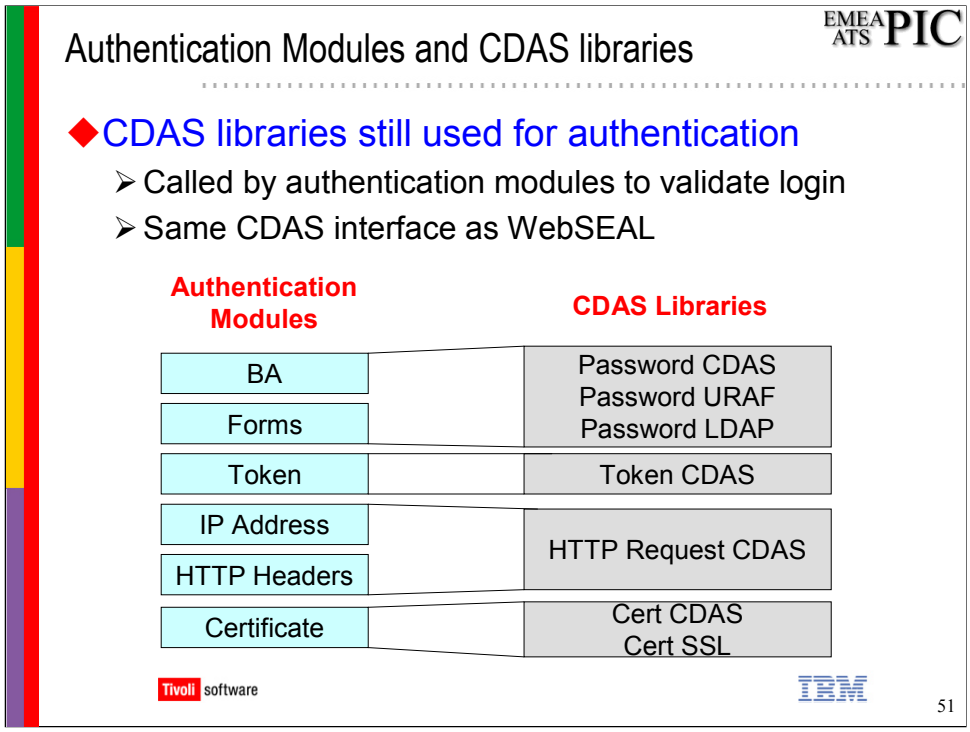
Macro	Description
%USERNAME%	The name of the logged in user
%ERROR_CODE%	A numeric error code associated with an error
%ERROR_TEXT% %ERROR%	Error text associated with an error
%URL%	The URL requested by the client
%HOSTNAME%	Fully qualified hostname
%HTTP_BASE%	Base HTTP URL of the server: http://<host>:<tcpport>/
%HTTPS_BASE%	Base HTTPS URL of the server: https://<host>:<sslport>/
%REFERER%	The value of the referrer header from the request, or "Unknown" if none
%BACK_URL%	The value of the referrer header from the request, or "/" if none
%BACK_NAME%	The value "BACK" if a referrer header is present in the request, or "HOME" if none.

Tivoli software IBM

50

The table above shows the variables that can be used in WEB pages to be served up by the web plug-ins. Variable substitution is done by the plug-in before the page is passed back to the user (via the Web Server).

Note: Not all variables are available for substitution in all pages. Each page has a list of the available variables in the sample page.

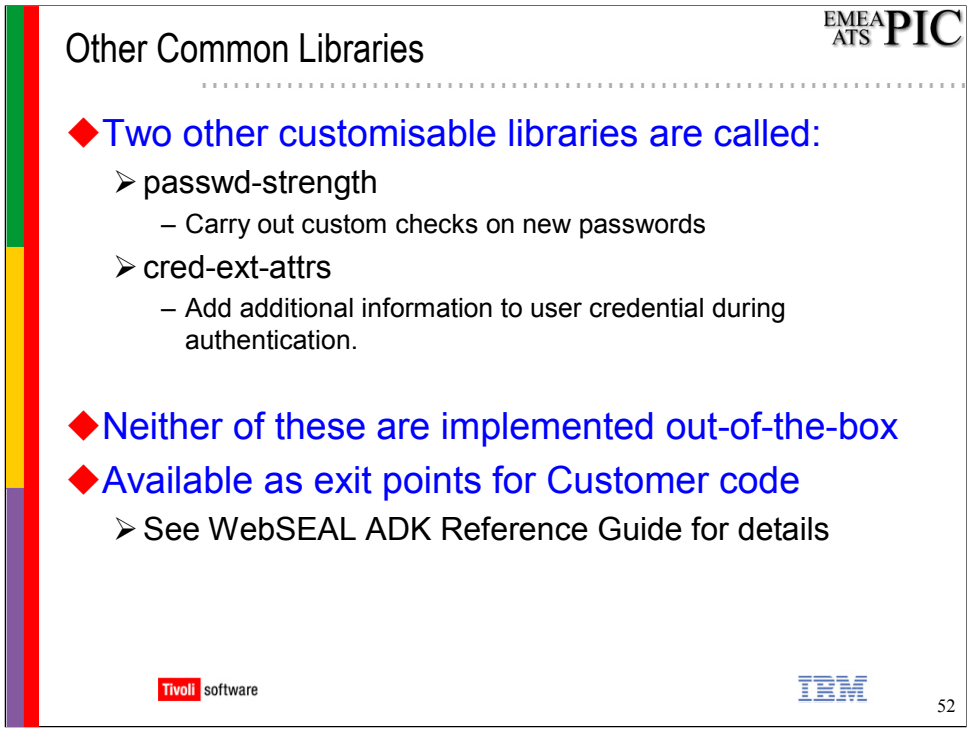


Although the authentication modules handle the process of extracting the authentication information from request standard CDAS libraries are called to validate the authentication information. This allows custom CDAS libraries written for WebSEAL to be used with the AM Web Plug-ins.

The chart above shows which libraries are used by which authentication modules.

Although in some cases multiple libraries are shown only one is ever used. The preference order is as shown above.

For example if a password-cdas and password-ldap library are both defined then user authentication (from form and BA login modules) will be done using the password-cdas library.

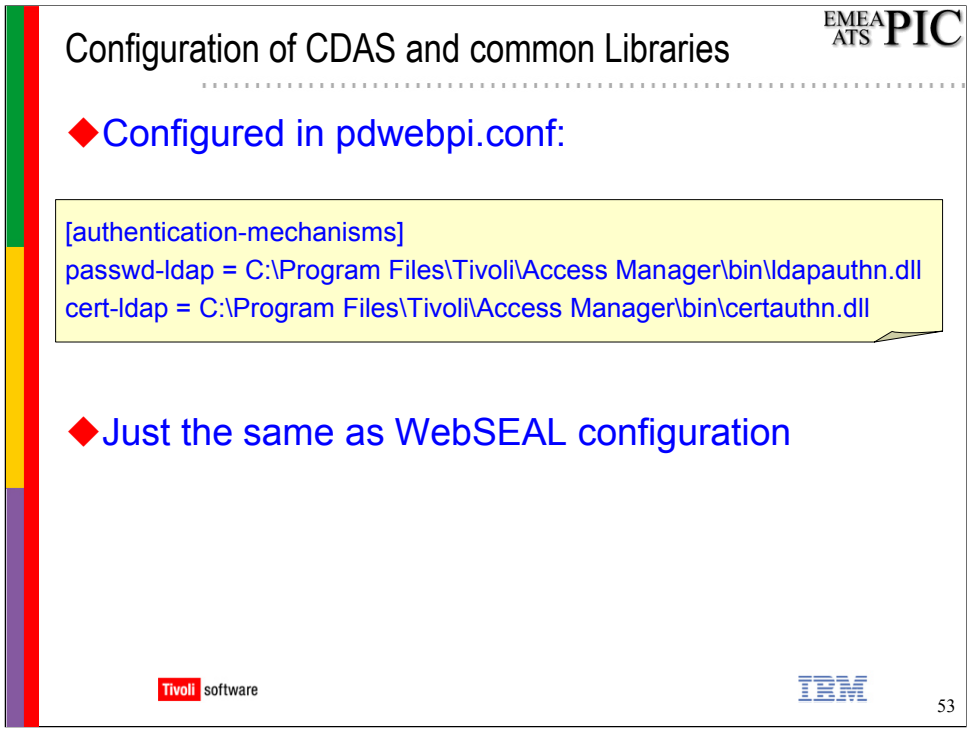


In addition to the authentication libraries there are two other standard AM libraries that can be used in the AMWeb Plug-ins:

passwd-strength – If defined then this library is called to check new passwords entered on the password change form of the AMWeb Plug-ins.

cred-ext-attrs – If defined then this library is called before a credential is built to allow custom attributes (name/value pairs) to be specified for inclusion in the credential.

See WebSEAL ADK Reference Guide for details on these libraries.



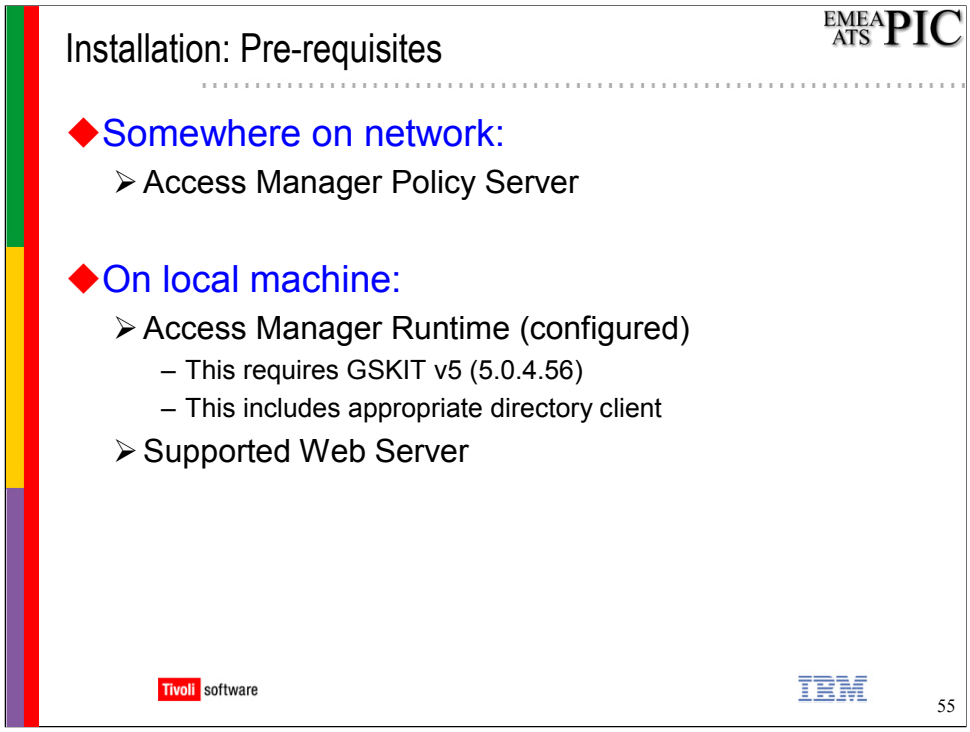
The configuration of authentication-mechanisms (and other common libraries) is exactly that same as for WebSEAL.

The *[authentication-mechanisms]* stanza defines the libraries using a keyword and a library as shown above. Valid keywords are shown in the configuration file. They are:

passwd-cdas, passwd-ldap, passwd-uraf, token-cdas, cert-ssl, cert-cdas, http-request, passwd-strength, cred-ext-attrs

Note: Just as in WebSEAL an authentication library is needed for some kinds of authentication to be used which is not defined by default. For example HTTP-Header and IP Address authentication both require that an http-request library be defined to perform the header->AM User mapping.

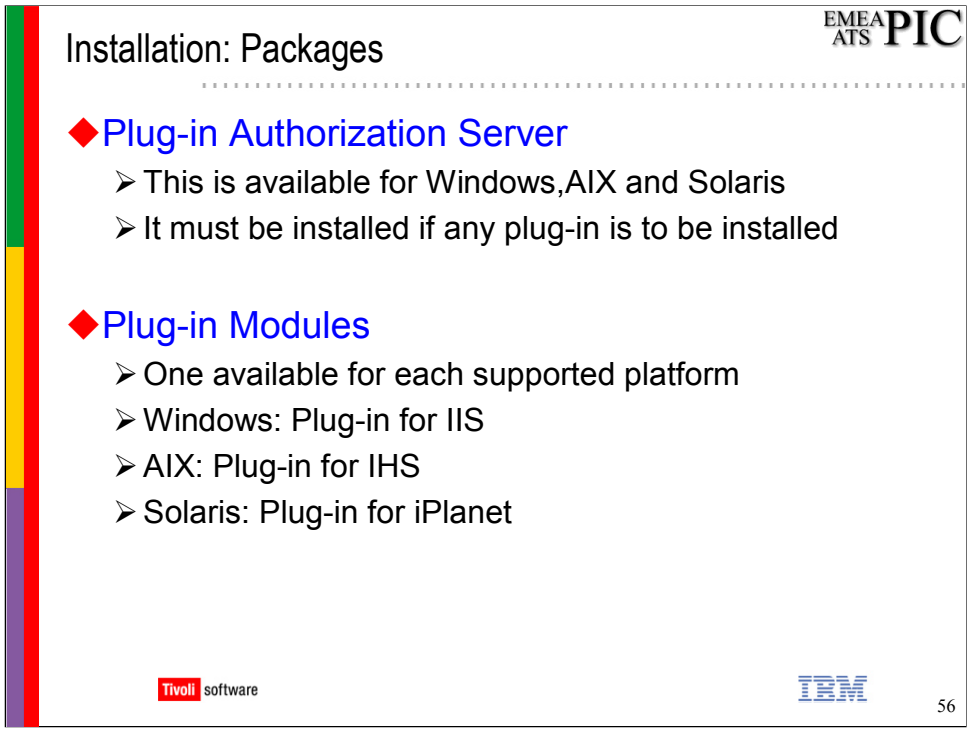

EMEA ATS PIC
Installation
Tivoli software
IBM
54



Since Web Server Plug-ins are an Access Manager component they require that an Access Manager Policy Server (new name for PD Management Server) be installed and configured somewhere on the network.

On the local machine Access Manager Runtime must be installed and configured so to enable communication with the Policy Server. To check this is ready to receive the Plug-ins, make sure that the PDADMIN utility works.

You'll also need to have installed and configured a supported Web Server.

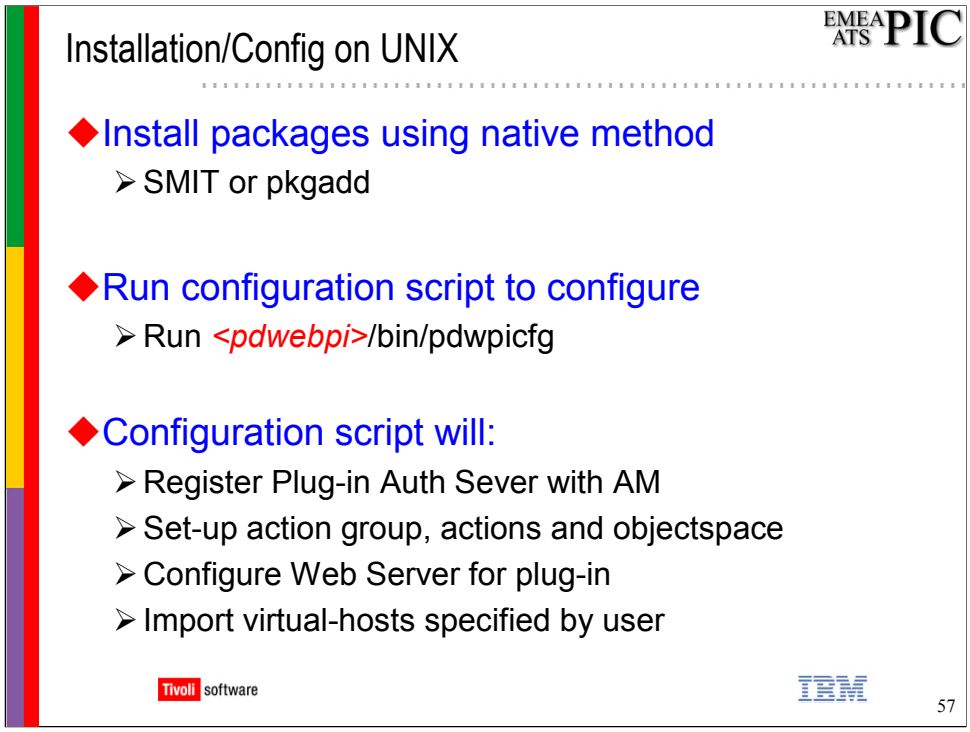


There are two installation packages for each platform.

The Plug-in Authorization Server is required on all platforms. It is always required.

In addition to the Authorization Server the appropriate plug-in package is required as shown above.

Installing the packages does not do any configuration. This is done by running the pdwpicfg utility after installation (see next page).



 # /opt/pdwebpi/pdwpicfg

Access Manager Web Plug-In Configuration

Please enter 'u' for unconfiguration or 'c' for configuration : **c**

Gathering the necessary configuration information...

Enter the full path name to the directory containing the IHS Web Server configuration file [/usr/HTTPServer/conf] :

Which virtual hosts are to be protected:

1. yourhost
2. anothervirtualhost

Menu choice [?,??,all]> **1,2**

Enter the Access Manager Administrator ID : **sec_master**

Enter the Access Manager Administrator password :

Enter the port number to listen on for AZN updates [7237] : **7937**

Do you want to enable SSL communication between the

Access Manager server and the LDAP server (y/n) [y] :**n**

Configuring the Web Plug-In (this may take a few minutes)...

Starting the server.

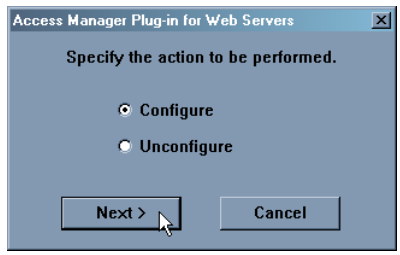
Note: The Web server must be restarted before the changes will take affect.

The plug-in configuration was successful.

Installation/Configuration on Windows – 1

EMEA ATS PIC

- Install with Installshield
- Use Start Menu to launch configuration GUI
  - Start->Programs->Access Manager Plug-in for Web Servers->Configuration



Tivoli software

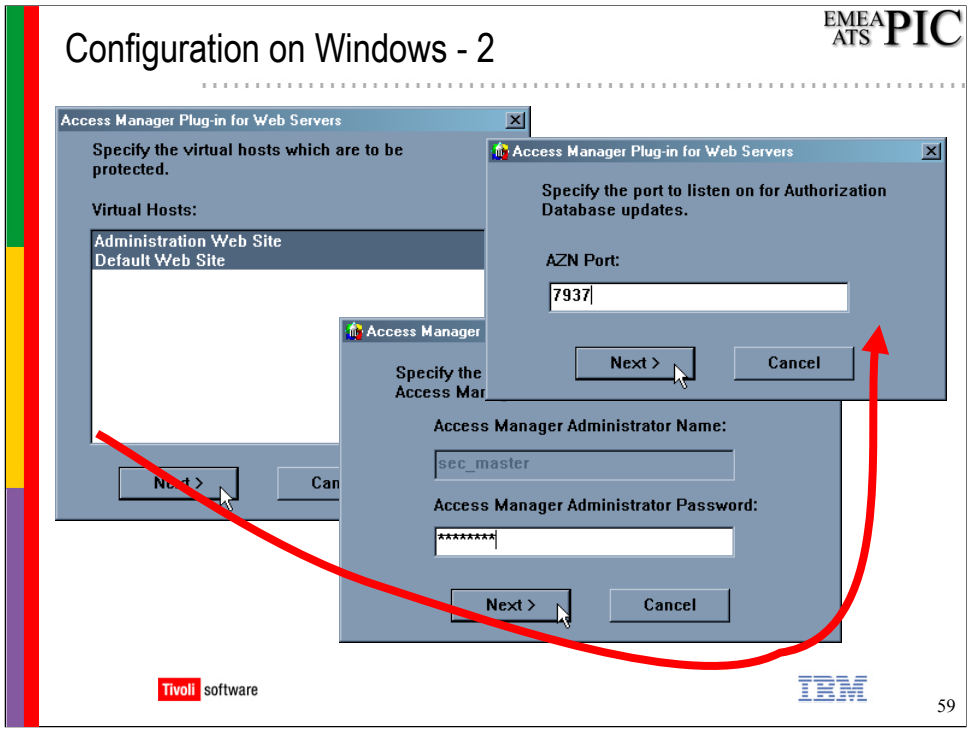
IBM

58

On Windows, The Web Plug-ins provide a Graphical User Interface for configuration/unconfiguration.

Once the Plug-in Auth Server and Plug-in for IIS are installed with installshield, launch the GUI from the Start menu as shown above.

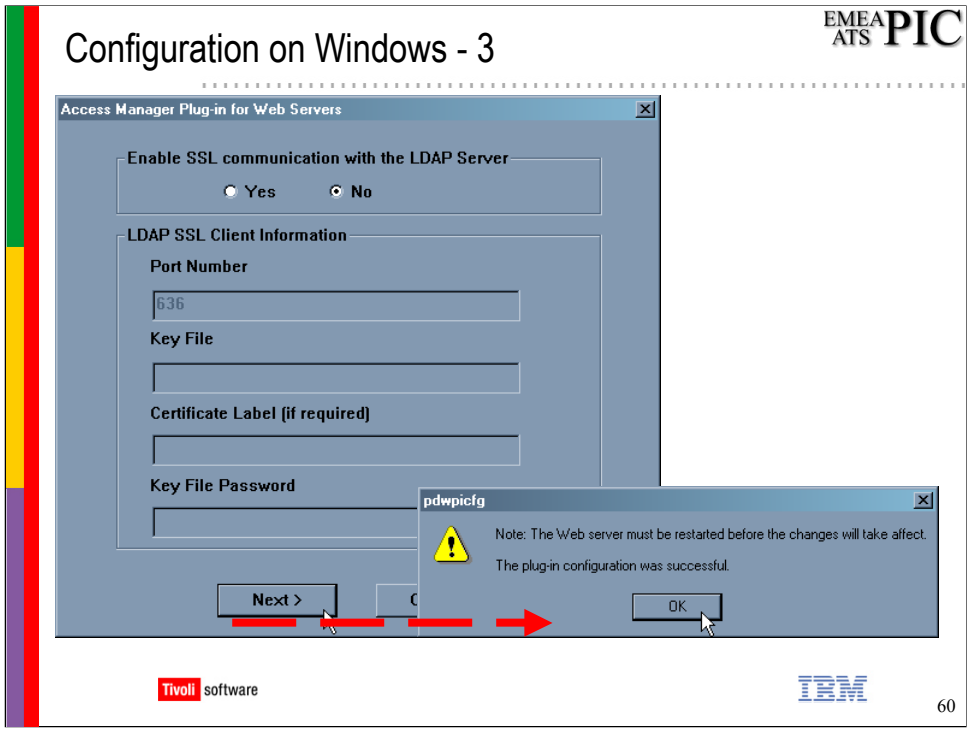
On the first screen select “Configure” and click Next.



On the next screen, select the virtual hosts that you wish to protect and click next.

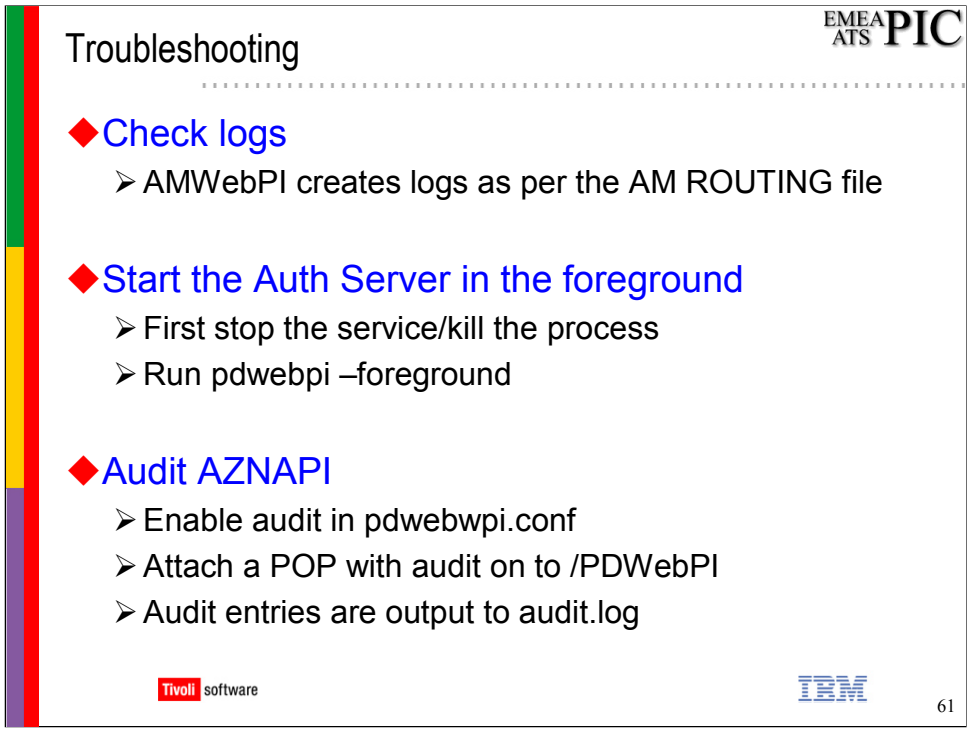
Enter the sec_master password and click next

Enter a port for the Plug-in Auth Server to listen on for requests from PDMgrd. If you have WebSEAL intsalled on the same machine then make sure you're using a different port.



If you are using SSL to communicate with the LDAP then fill in the details. In the screenshot above SSL to LDAP is disabled.

After the configuration has completed you will see the following message if configuration was successful. You will need to re-start IIS in order for protection by the plug-in to be active.

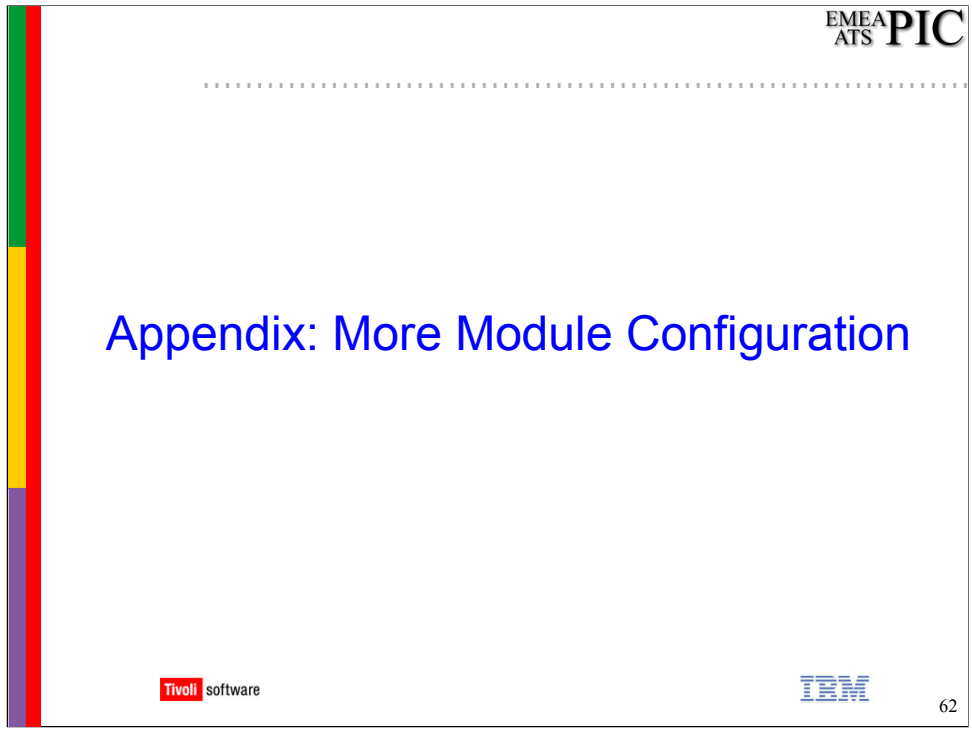


If you are having trouble with the Web Plug-ins then the three methods shown above can give some clues as to where the problem lies.

Initially, check fatal.log, error.log and warning.log for error messages. This could point you to the problem.

If you need a realtime view of the error messages then try starting the Auth Sever in the foreground. Use the –foreground option to achieve this.

If you need to check what objects the Auth Server is checking permissions against then turn on auditing globally for the Auth Server in the pdwebpi.conf file and then use a POP to specify which parts of the tree you require audit records for.



The appendix includes a brief description of each of the module configurations. This is for completeness because configuration is, in almost all cases, the same as the configuration for the equivalent feature in WebSEAL.

The keyword used in virtual-host to module mapping is shown at the bottom of each slide. In some case a module can be specified as more than on type. In this case both are shown.

Session Module – SSL Session ID

EMEA ATS PIC

- Recognise session using SSL Session ID
  - Session ID set during SSL session initialization
  - Included with every packet until session ends
- Only valid for HTTPS connections
  - HTTP sessions do not use SSL
- Cannot be used with IIS Server or IE Browser
  - IIS Server does not pass SSL ID to plug-in
  - IE resets SSL Session ID every 60 seconds

```
session = ssl-id
```

Tivoli software

IBM

63

Session Module – Session Cookie

EMEA ATS PIC

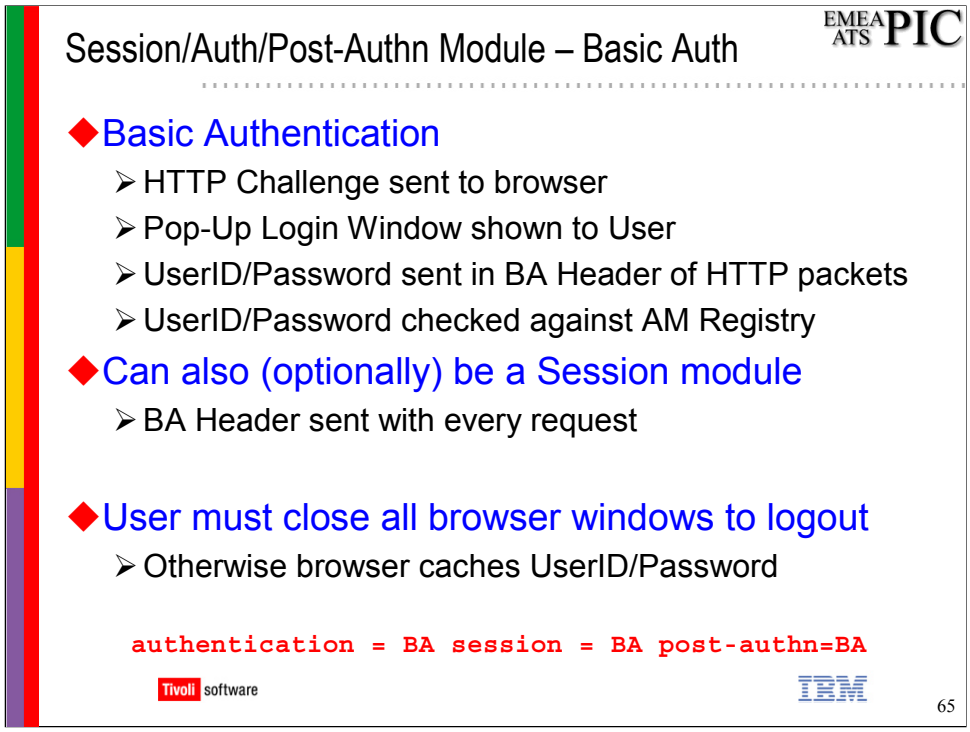
- Recognise session using a cookie
  - Cookie set in browser by first response
  - Cookie contains a large random number
  - Browser sends cookie with all subsequent requests
- Browser must have cookies enabled to use
  - This is the default in most cases
  - Hard to use the internet today without cookies
- Normally used as a “Last Resort”
  - Since we like to avoid using cookies

```
session = session-cookie
```

Tivoli software

IBM

64



As an authentication module BA module uses BA header for authentication.

As an session module BA module uses BA header for indexing session.

As a post-authzn module, BA strips BA headers provided by the user so that they cannot be spoofed.

On Windows, IIS is configured, by default, to attempt integrated windows login if it gets a connection from an IE browser. This causes problems for the Plug-in because it means that the user never gets prompted for BA authentication. In order to allow BA authentication from IE to IIS you must disable Integrated Windows Login. This can be done per virtual host in the "Directory Security" tab of the virtual host properties or for all virtual hosts in the Master WWW properties (right click on the host name in Internet Services Manager and click properties).

If you have trouble with BA using IE and you suspect this problem, try Netscape to confirm this is the problem you are seeing.

Basic Authentication Configuration

EMEA ATS PIC

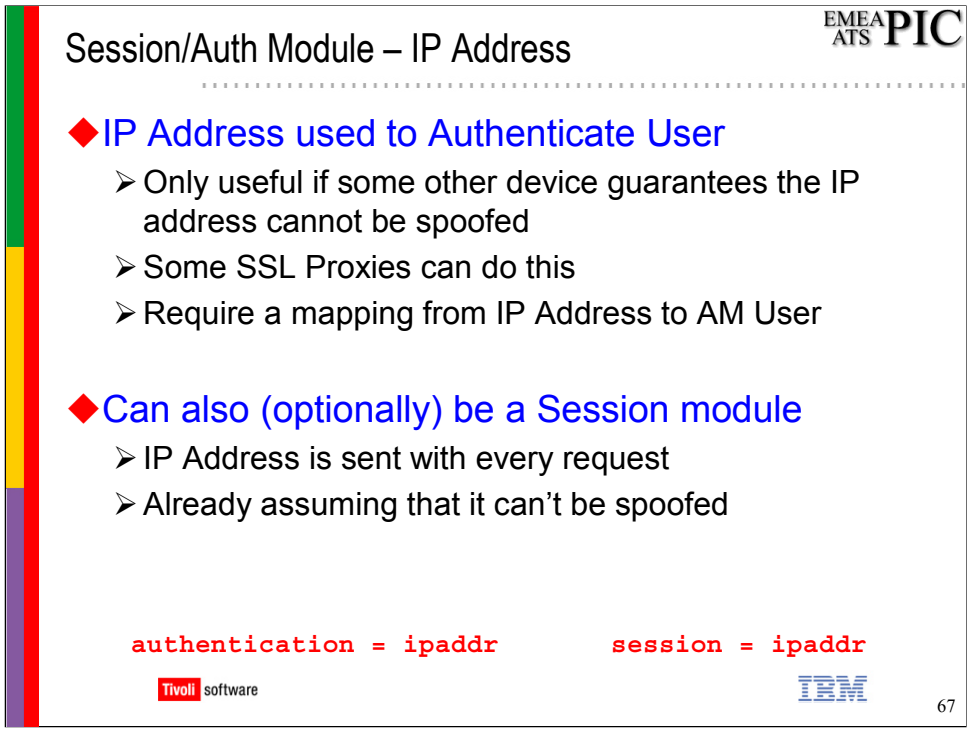
- Configuration allows Realm name to be changed
  - This is what is displayed in BA Pop-up in browser
- Basic Authentication in pdwebpi.conf:
  - Default config can be overridden per virtual host

```
[BA] or [BA:<VH-Label>]
# Realm name must be surrounded by double-quotes
basic-auth-realm = "Access Manager"
```

Tivoli software

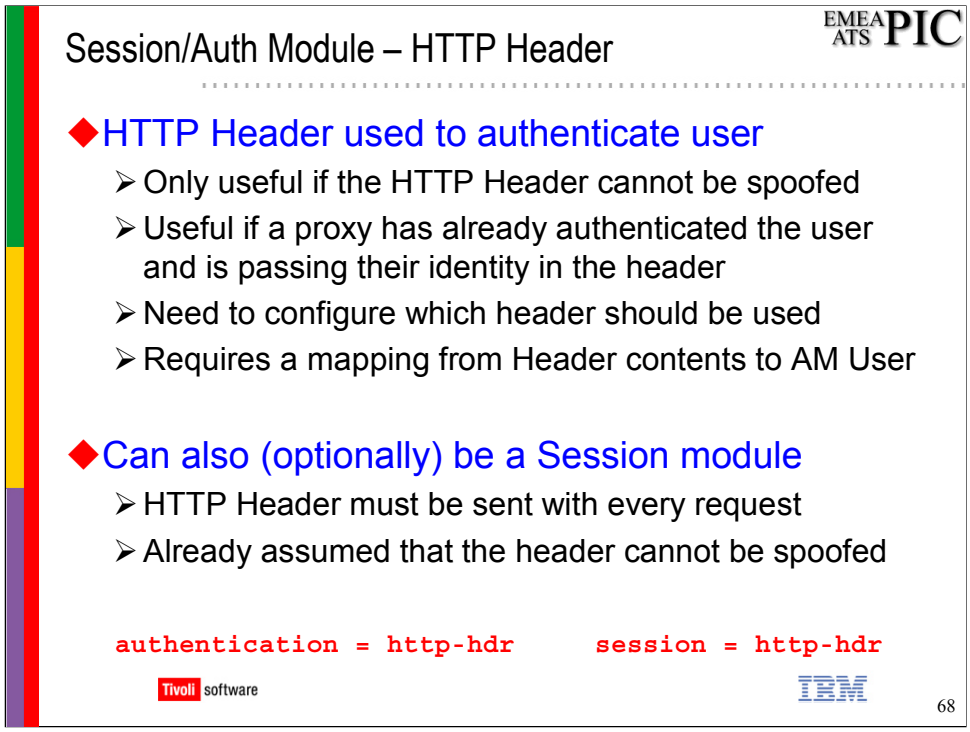
IBM

66



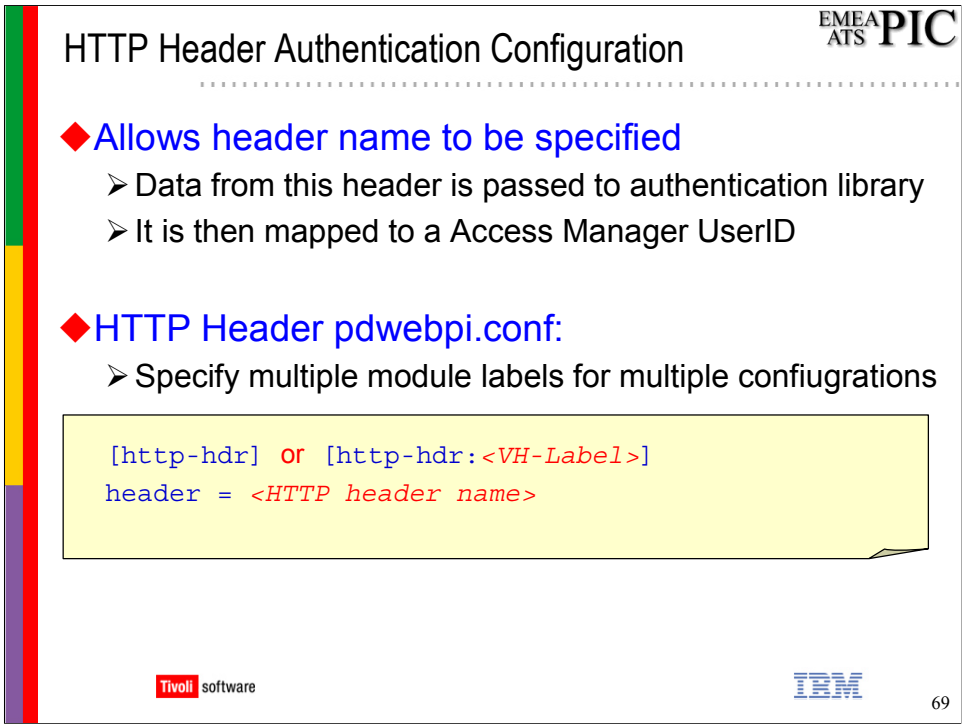
As an authentication module ipaddr module uses Client IP address for authentication.

As an session module ipaddr module uses Client IP address for indexing session.



As an authentication module http-hdr module uses specified HTTP Header for authentication.

As an session module http-hdr module uses specified HTTP Header for indexing session.



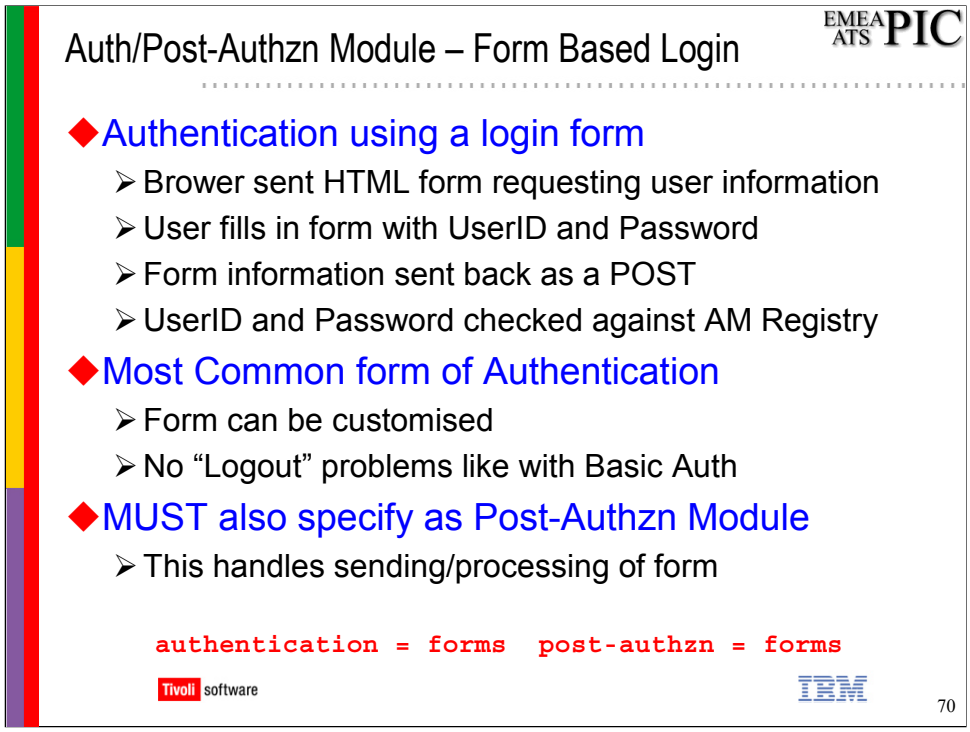
It is possible to configure multiple http header authentication modules for the same virtual-host as follows:

[modules]
http-hdr-1 = pdwebpi-http-hdr-module
http-hdr-2 = pdwebpi-http-hdr-module

[http-hdr-1]
header = xxx

[http-hdr-2]
header = yyy

[VirtualHost1]
authentication = http-hdr-1
authentication = http-hdr-2



If forms authentication is used then it must also be specified as a post-authzn module which will process login form when it is POSTed.

Form-based Login Configuration

EMEA ATS PIC

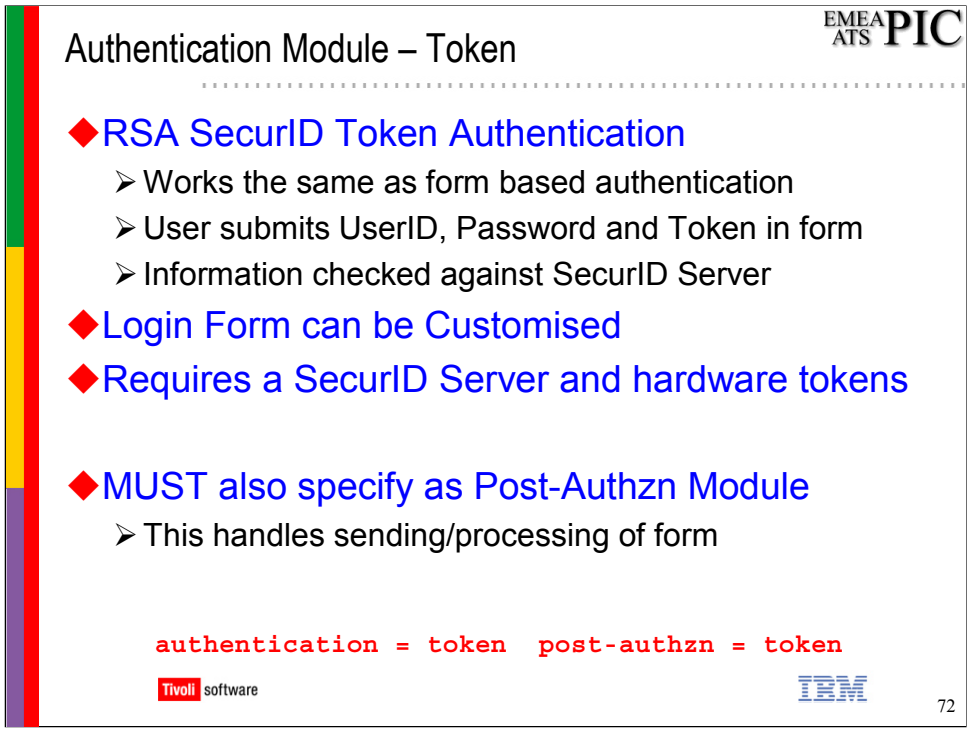
- Configuration allows Login Form to be specified
  - This is what is sent to the user when login required
  - File path relative to *<AMWebPI>*/nls/html/*<LANG>*/
- Form login in pdwebpi.conf:
  - Default config can be overridden per virtual host

```
[forms] or [forms:<VH-Label>]
login-form = login.html
```

Tivoli software

IBM

71



If token authentication is used then it must also be specified as a post-authzn module which will process login form when it is POSTed.

Token Login Configuration

EMEA ATS PIC

- Configuration allows Login Forms to be specified
  - Page for initial token login
  - Page for requesting next token if this is needed
  - File path relative to *<AMWebPI>*/nls/html/*<LANG>*/
- Token login in pdwebpi.conf:
  - Default config can be overridden per virtual host

```
[token-card] or [token-card:<VH-Label>]
token-login-form = tokenlogin.html
next-token-form = nexttoken.html
```

Tivoli software

IBM

73

As an authentication module failover module uses failover cookie for authentication.

As a post-authzn module failover module sets failover cookie for use by other WebSEAL servers and AMWeb Plug-ins.

Failover Configuration

EMEA ATS PIC

- Configuration Requires
  - Path and Filename for shared failover key file
    - Generated using cdsso_key_gen and distributed
  - Cookie lifetime
  - Whether cookies can be used for Domain SSO
- Failover in pdwebpi.conf:
  - Default config can be overridden per virtual host

```
[failover] or [failover:<VH-Label>]
failover-cookies-keyfile = <filename for shared key>
failover-cookies-lifetime = 30
enable-failover-cookie-for-domain = false
```

Tivoli software

IBM

75

Authentication Module - Certificate

EMEA ATS PIC

- Client Side x.509 Certificate
  - Web Server requests client certificate during SSL Session negotiation
  - Certificate signature/Expiry Dates checked
  - DN of Certificate used as AM User
- Only available over SSL Sessions
- No DN Mapping provided

```
authentication = cert
```

Tivoli software

IBM

76

As an authentication module ecsso module uses incoming eCSSO tokens on requests to authenticate user.

As a post-authzn module ecsso module responds to incoming /vouchfor requests and sets eCSSO domain cookie.

The configuration of the ecsso module cannot be made Virtual Host specific by qualifying the stanza name with the virtual host. If different configurations are needed then multiple module labels must be used – each with its own configuration stanzas.

eCSSO Configuration – 1

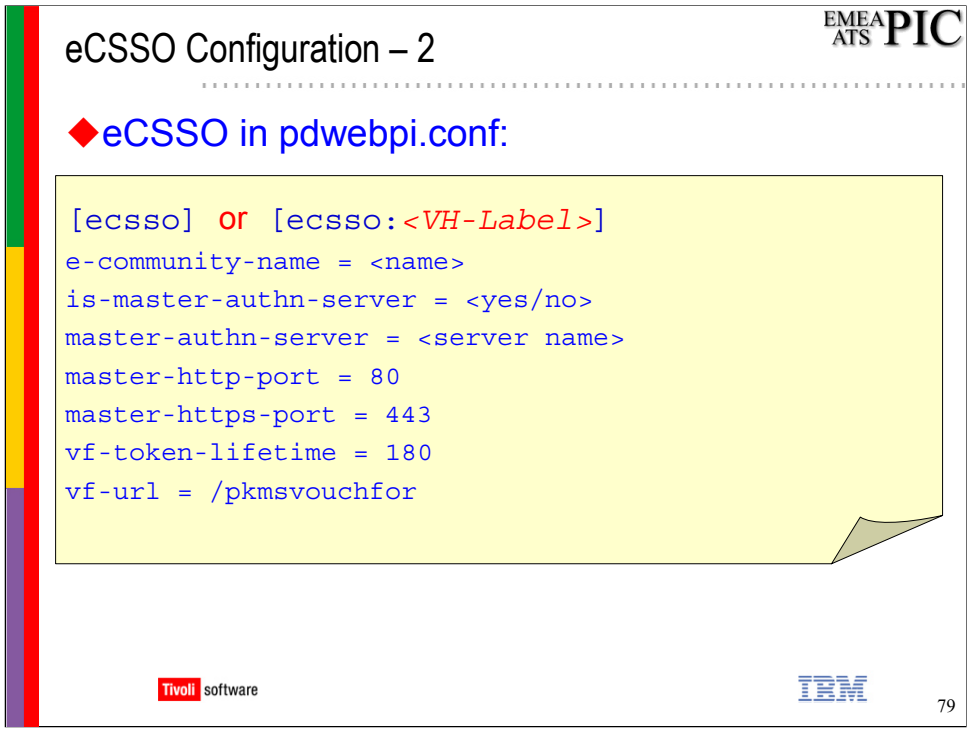
EMEA ATS PIC

- Need to configure:
  - eCommunity name
  - Whether local machine is Master Authentication Server
    - If not then need to give hostname/ports for MAS
  - "Vouchfor" URI to trigger eCSSO
- There is a change to eCSSO in AMWebPI
  - There is no "local-domain-key" parameter
- Local Key is configured with remote keys
  - It simply specifies the local DNS domain

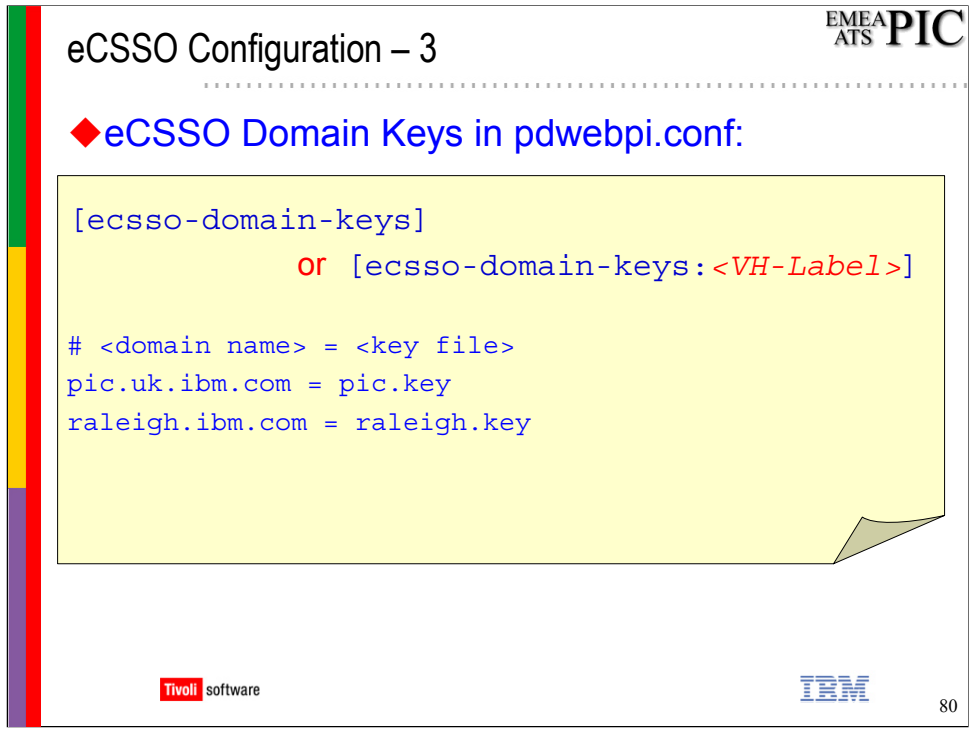
Tivoli software

IBM

78



Note: If the e-community name is not specified then it defaults to the module label name (in the example above this is *ecsso*).



If the module label (default ecsso) is changed then the name of the domain keys module also changes:

e.g.

[modules]

ecsso = pdwebpi-ecsso-module

My-ecsso = pdwebpi-ecsso-module

…

[ecsso]

…

[ecsso-domain-keys]

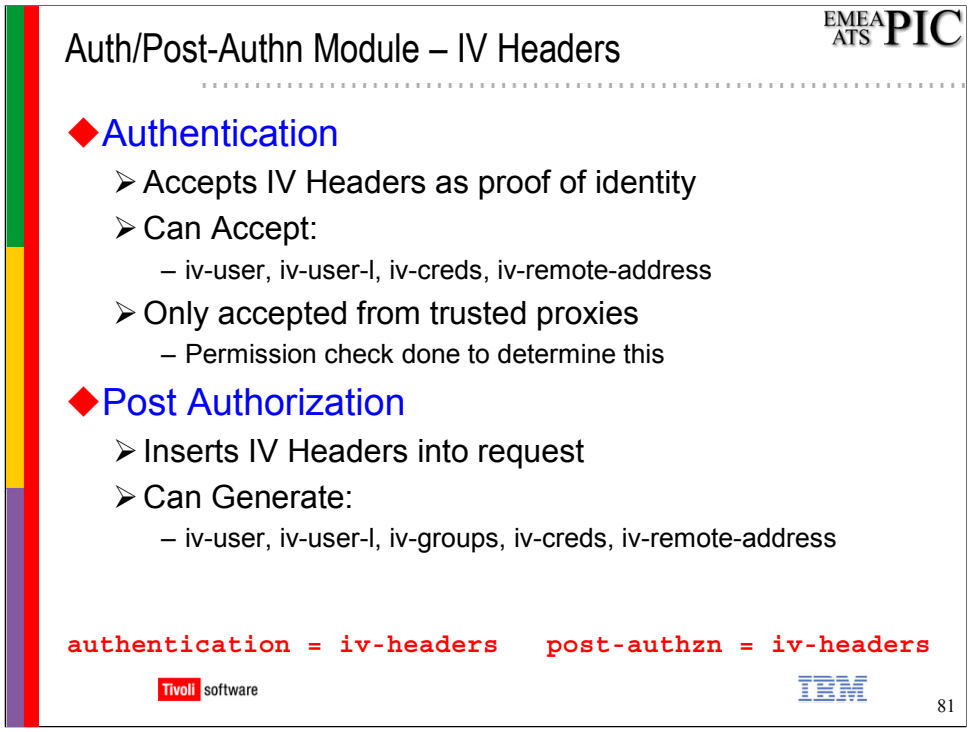
…

[my-ecsso]

…

[my-ecsso-domain-keys]

…



As an authentication module iv-headers module uses IV-HEADERS in requests to authenticate user

As a post-authzn module iv-headers module inserts IV HTTP Headers into requests before they are passed to the Web Server for processing.

IV Headers Configuration

EMEA ATS PIC

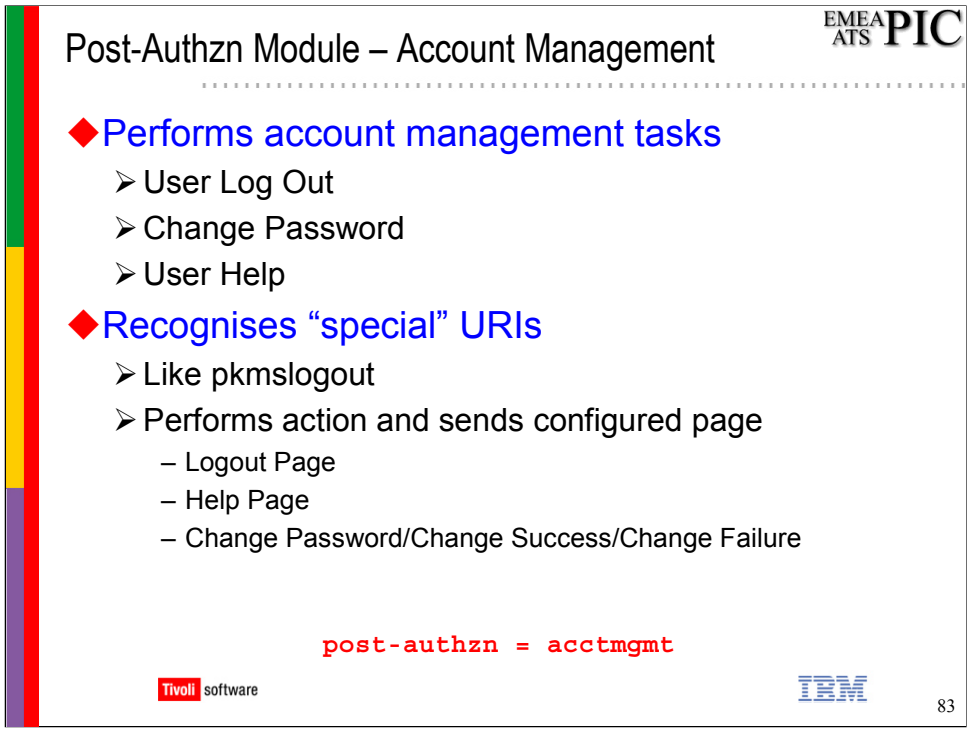
- Specify which headers are accepted/generated
  - List of headers
    - iv-creds, iv-user, iv-user-l, iv-groups, iv-remote-address
  - Or all to accept/generate all
- IV Headers in pdwebpi.conf:
  - Default config can be overridden per virtual host

```
[iv-headers] or [iv-headers:<VH-Label>]
accept = iv-creds, iv-user-l
generate = all
```

Tivoli software

IBM

82



One important Post-Authzn module which should always be specified is the Account Management module (label **acctmgmt**). This module performs account management tasks by recognising the URLs that trigger the management actions (such as /pkmslogout) and performing the appropriate action.

Post-Authzn Module – LTPA Cookie

EMEA ATS PIC

- Allows LTPA Cookie to be inserted into request
  - Before it is handled by the Web Server
- Allows Single Sign-On to WebSphere
  - Usually WebSphere sharing User Registry
- LTPA Cookie is not sent back to client
  - It is cached inside the Plug-in

```
post-authzn = ltpa
```

Tivoli software

IBM

84

LTPA Configuration

EMEA ATS PIC

- LTPA Configuration Requires
  - Path and filename for LTPA keyfile (from WebSphere)
  - Stash file or Password to decode keyfile
  - Lifetime of LTPA cookie
- LTPA in pdwebpi.conf:
  - Default config can be overridden per virtual host

```
[ltpa] or [ltpa:<VH-Label>]
ltpa-keyfile = <full path of keyfile>
ltpa-stash-file = <password stash file location>
ltpa-password = <password in lieu of stash file>
ltpa-lifetime = <lifetime of LTPA cookie in seconds>
```

Tivoli software IBM

85

Post-Authzn Module – Tag/Value Support

EMEA ATS PIC

- Allows HTTP Headers to be added to request
  - Before it is processed by the Web Server
- Information is taken from users credential
  - Authentication Modules add data to credential at login time
- Useful for passing information to Web Apps
  - Easy for them to extract data from HTTP Headers

```
post-authzn = tag-value
```

Tivoli software

IBM

86

Tag-Value Configuration

EMEA ATS PIC

- Tag-Value
  - Specify whether configuration should be cached
    - Rather than checking objectspace each time
  - If cache on then refresh time also set (in seconds)
- Tag-Value in pdwebpi.conf:
  - Default config can be overridden per virtual host

```
[tag-value] or [tag-value:<VH-Label>]
cache-definitions = yes
cache-refresh-interval = 60
```

Tivoli software

IBM

87