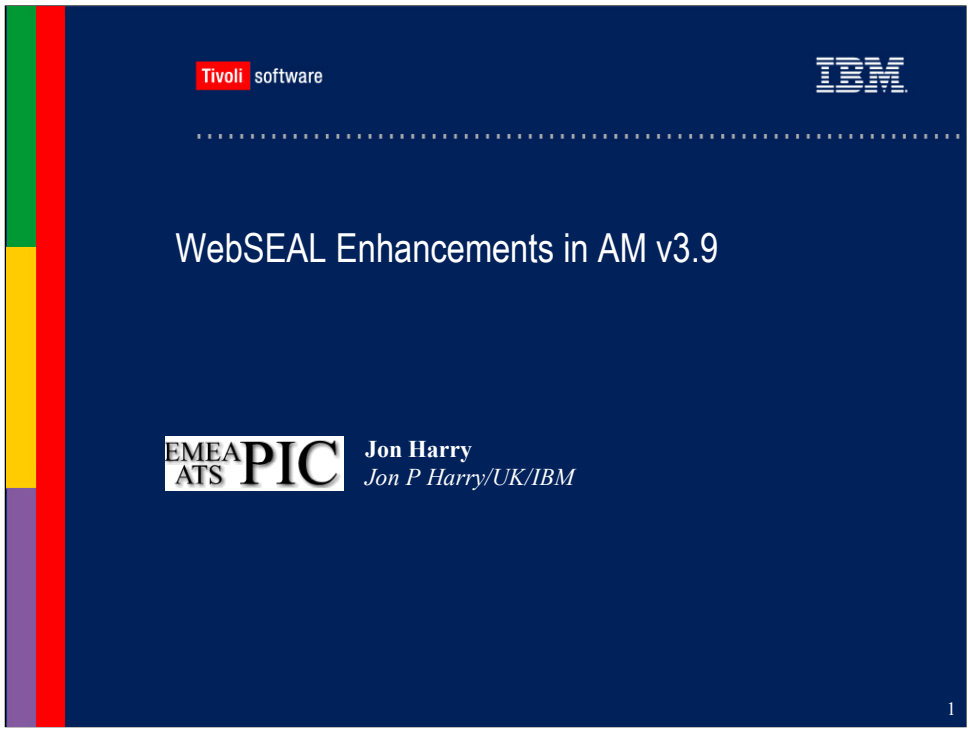
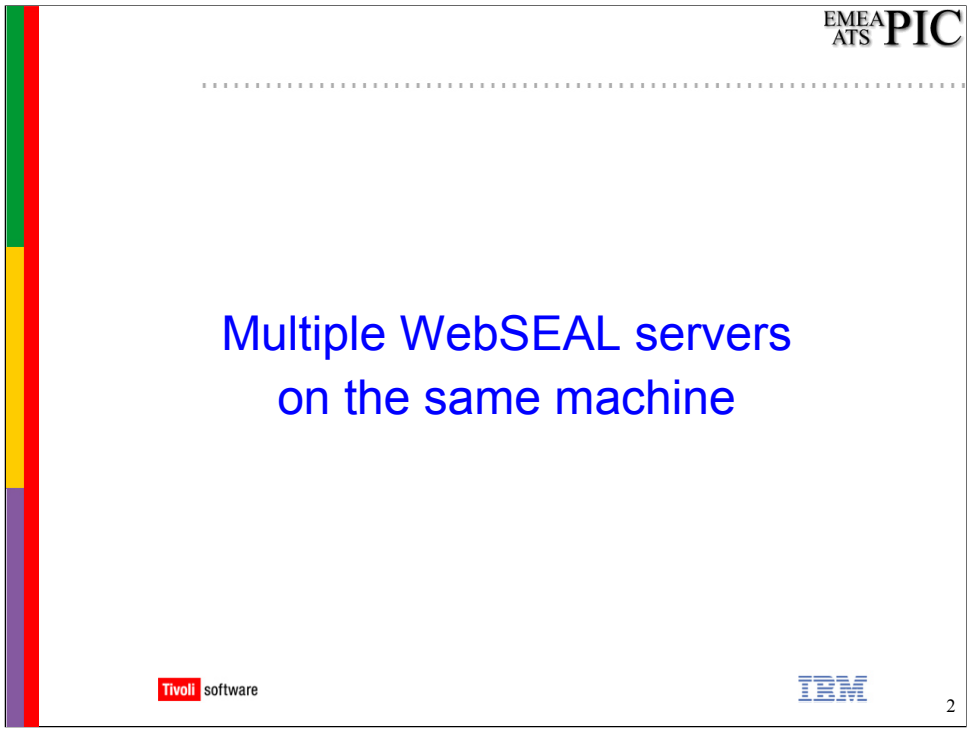


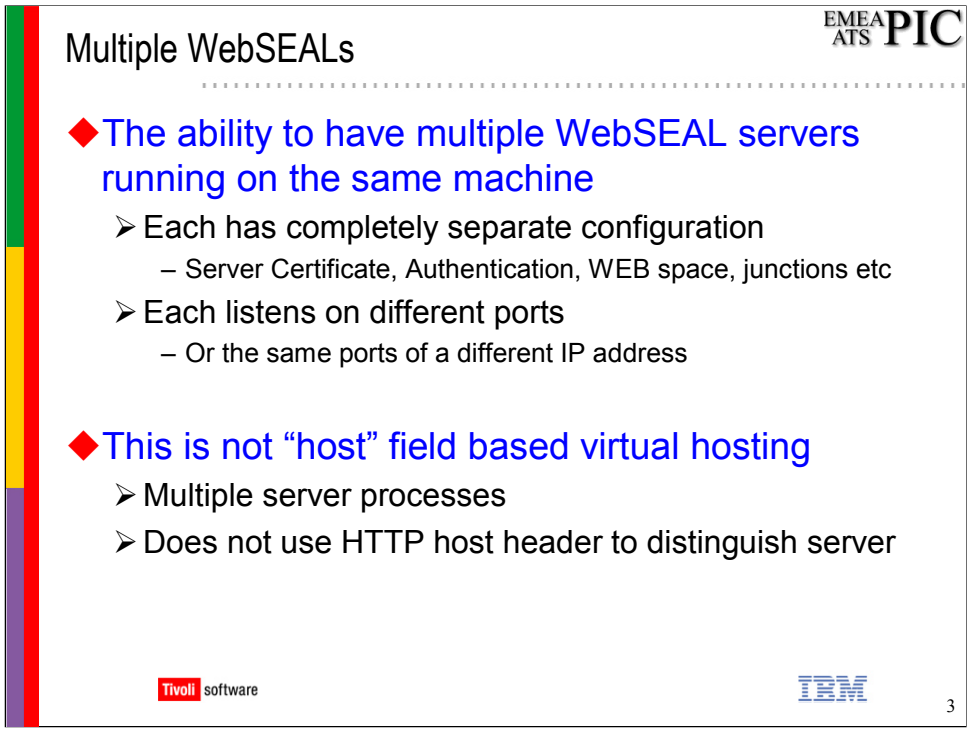
This Presentation covers the new features of WebSEAL in Access Manager v3.9.

It also covers a few subjects that have either been introduced since the introduction of PD v3.8 or were in PD v3.8 but not documented until now.



This feature was initially introduced in WebSEAL v3.8 fixpack 1.

Most Web servers allow several WEB sites to be hosted on a single machine. A different WEB site is characterised by different configuration and different URL space. Clients request different servers by typing different DNS names at the browser. In HTTP v1.1 the requested host is also included in the required Host HTTP Header. Note that the server never sees the DNS name that the user connects to – this is a mapping done at the client side.

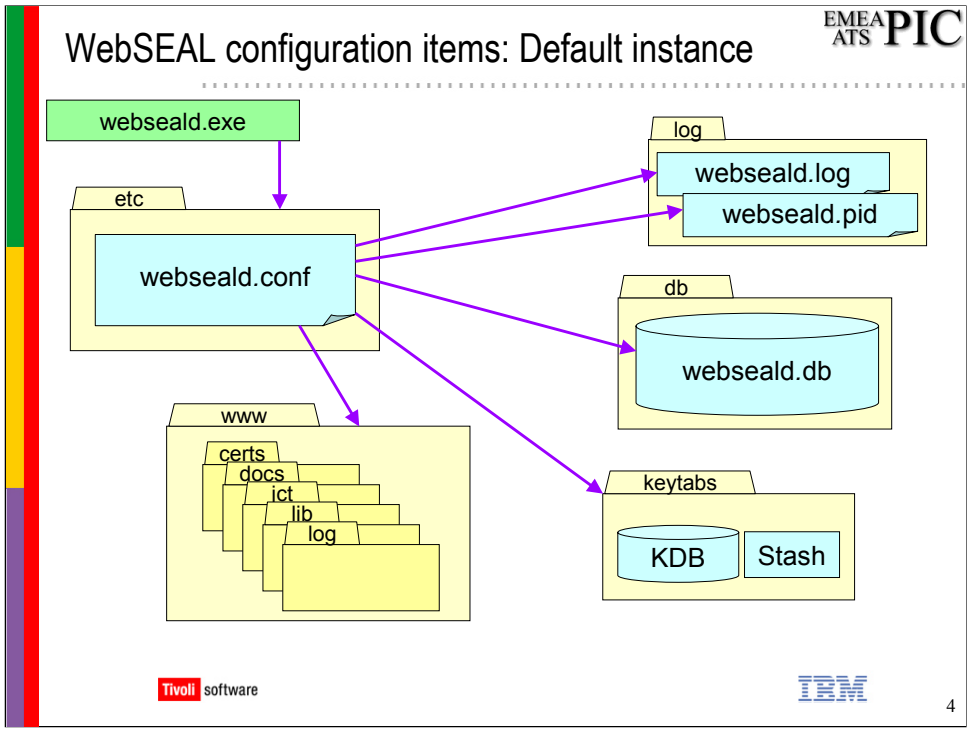


There are two main ways to support different WEB sites on the same machine:

1. Single Server instance
In this case only a single TCP/IP port is used by the server for each protocol (HTTP / HTTPS) but it will give the appearance of different WEB sites based on the contents of the HTTP Host header. This is usually considered "true" virtual hosting. The DNS names of each of the hosted sites each map to the same IP address. When a request is received at the server the Host header is examined to see which site the user is actually requesting.

2. Multiple Server instances
In this case a different server instance is used for each hosted site. Each protocol of each server instance requires a different IP Adress/port combination. Since WEB Services are usually expected to be on port 80 and 443 this usually requires that either multiple IP addresses are allocated or that there is a front-end device (load balancer for example) that can map incoming requests to different ports of the same IP address. In this case the Host HTTP header is not used.

WebSEAL supports Multiple Server Instances only.

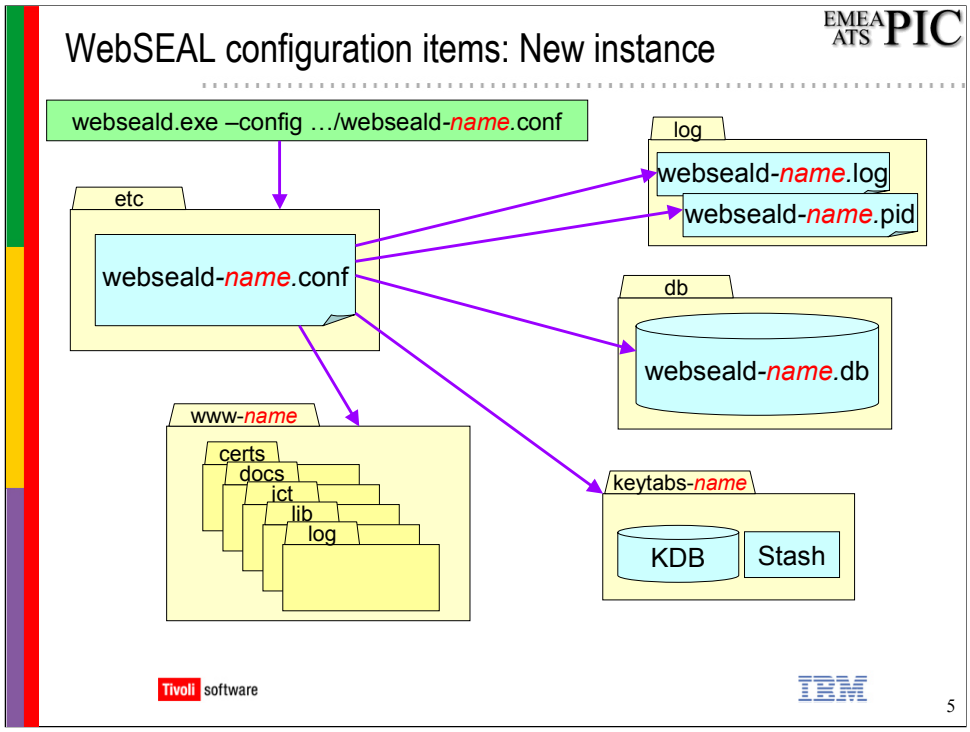


The diagram above shows where files are stored for the initial WebSEAL instance as configured using PDCONFIG.

If WebSEAL executable is not given any command-line options, as is the case for the default instance, it will load the webseald.conf configuration file from the <PDWeb>/etc directory.

The location of all other files to be used by the WebSEAL instance are specified in that configuration file.

If the webseald executable could be made to read an alternative webseald.conf configuration file then alternative versions of all of the other files could be used too.



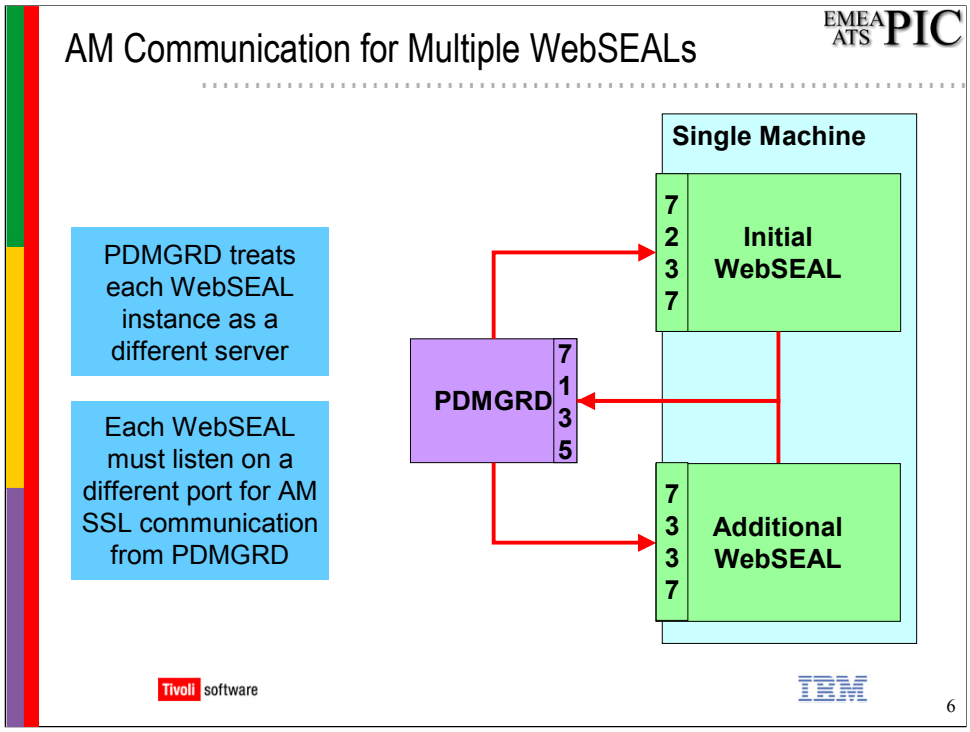
The diagram above shows where files are stored for additional WebSEAL instances created using the techniques described in this presentation.

The name of the instance – which is used throughout the multiple instance functionality to distinguish the instances from each other – is added to directories and files to make them unique.

The WebSEAL executable is given a command-line option that tells it to load an alternative configuration file (webseald-*name*.conf) from the <pdweb>/etc directory. This file then points to alternative versions of all of the other configuration files.

Since the webseald-*name*.conf file contains all configuration information for the WebSEAL server, each instance has a completely separate configuration and so can use different authentication, different certificates, different local pages, different junctions etc.

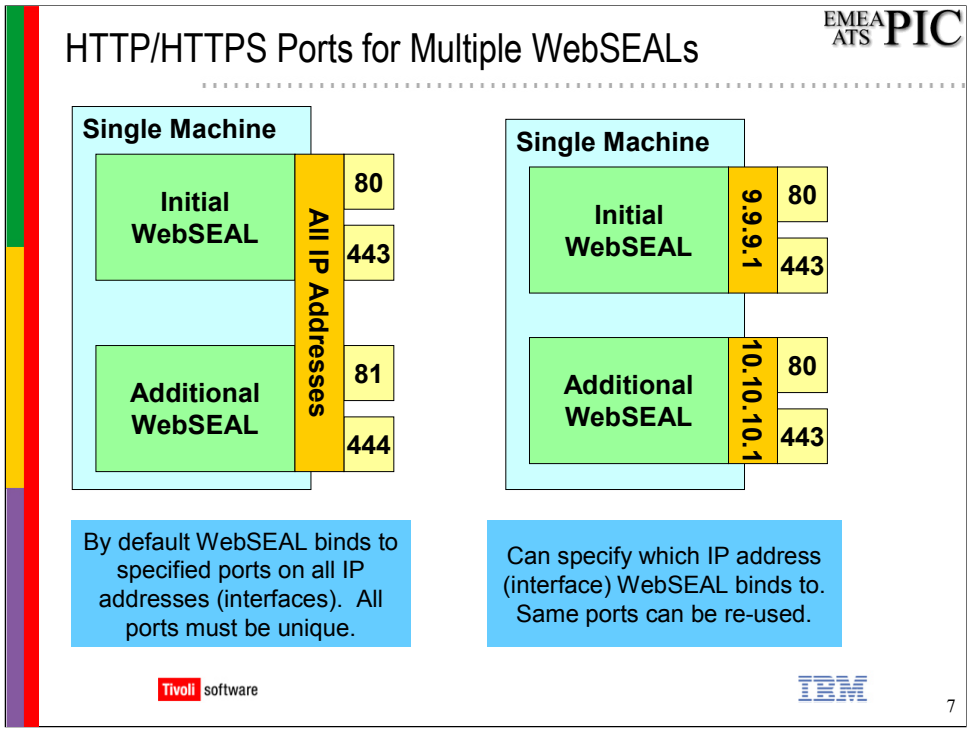
The only shared configuration is the registry configuration. All WebSEAL servers must still be in the same AM domain because they all use the same AMRTE (which can only be in a single AM domain).



Since each WebSEAL instance is a complete WebSEAL server they are treated as different servers by Access Manager. This means that each WebSEAL instance must listen on a different port for requests from the management server. The default port for the default WebSEAL instance is 7237. A different port must be used by each instance – this is specified at configuration time (shown later).

All WebSEAL instances (and all other AM servers) initiate connections with the management server on (by default) port 7135.

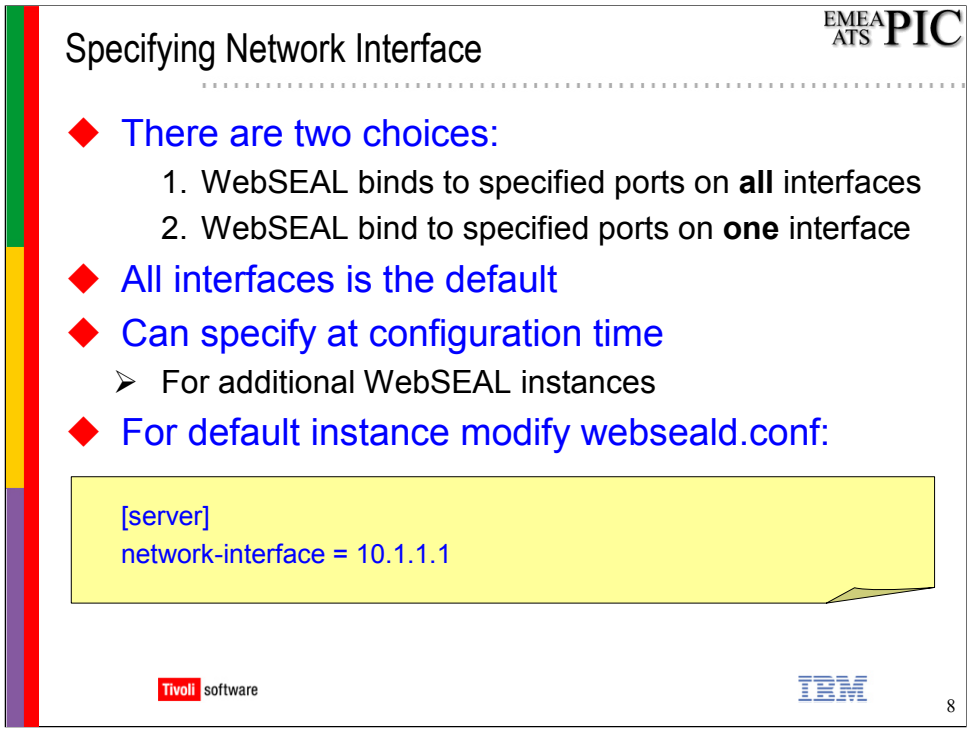
The default WebSEAL instance on a machine is known using the server name webseald-*hostname*. Additional instances are named *instancename*-webseald-*hostname*. This name is seen using a *server list* command and is used to specify the instance in other *server* commands.



Since WebSEAL does not support virtual hosting using the "Host" HTTP Header this means that a different IP Address/Port combination must be used for each protocol (HTTP/HTTPS) of each instance.

By default a WebSEAL server will bind to the specified ports on all IP addresses specified on the local machine. In this case each instance must use a different port number. Clients can access the WebSEAL instances via any of the local IP addresses but must specify the correct port number to connect to the desired instance.

If the same ports are to be used by multiple instances (port 80 and 443 for example) then each of these instances must be configured to bind to a different IP address. When a WebSEAL instance is bound to a single IP address clients must connect to that IP address in order to reach the server.

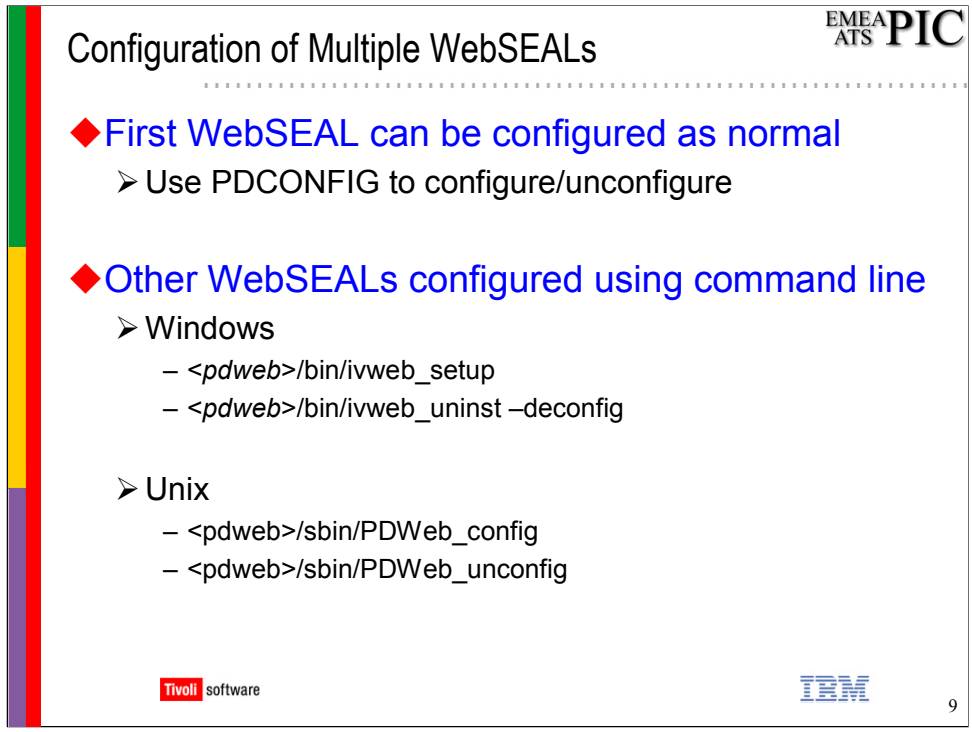


A WebSEAL instance can be configured either to listen on the specified ports of a single IP address or to listen to the specified ports on all IP addresses. It cannot be configured to listen (for example) on two out of four interfaces.

The default for a WebSEAL server (and the way in which the default instance is configured) is to listen on all IP addresses of the local machine.

A command-line parameter can be used to modify this behaviour when configuring additional instances (see later) but the configuration of the default instance can only be changed by modifying the webseald.conf file once it has been configured.

The *network-interface* parameter does not exist in the configuration file by default. It must be added to the *[server]* stanza. The WebSEAL server must be re-started once this change has been made.



Configuring the first WebSEAL on a machine is no different to configuring it when it is to be the only instance. Use the PDCONFIG utility to configure.

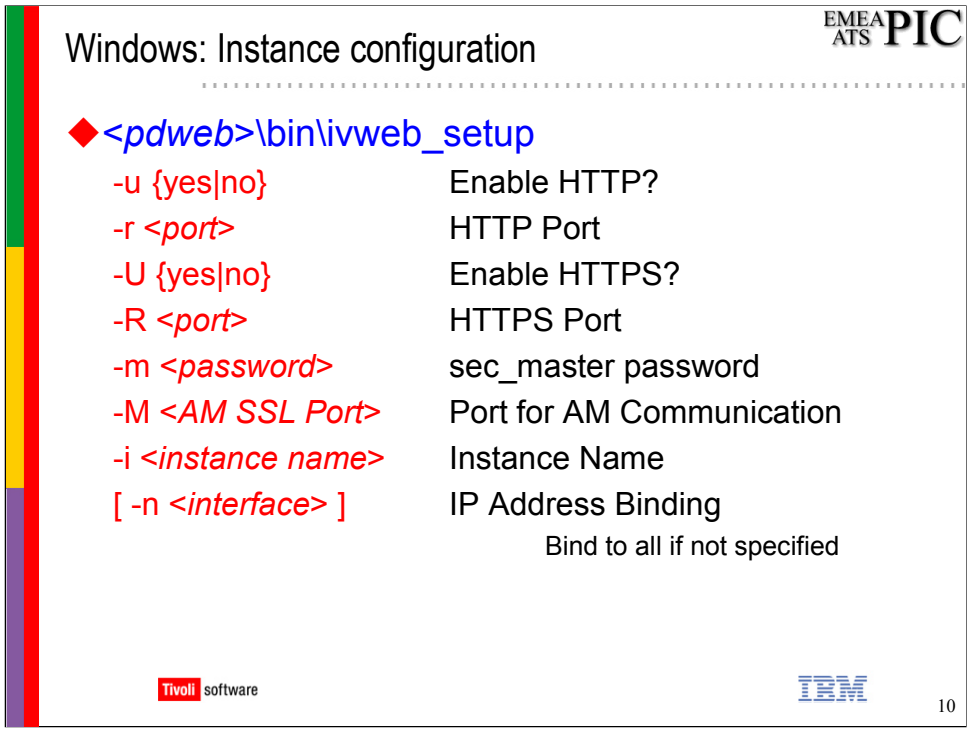
If you are going to use the same ports on different IP addresses you will need to edit webseald.conf and add a *network-interface* attribute to configure the WebSEAL instance to bind to a single IP address.

Additional WebSEAL instances are configured manually using the command-line. The command is different on Windows and UNIX systems – the names and path are shown above. The use of these commands differs between operating systems too – see the next pages for details. These commands are the same ones that are usually called from within PDCONFIG but the addition of an instance parameter indicates that this is an additional instance

To un-configure the default WebSEAL instance use the PDCONFIG utility as normal.

Un-configuring additional WebSEAL instances is done using the command-line as shown above.

The configuration/unconfiguration of the default instance and the configuration/unconfiguration of additional instances is independent. The order is not important.



The slide above shows the syntax for the *ivweb_setup* command which is used on Windows systems to configure additional WebSEAL instances.

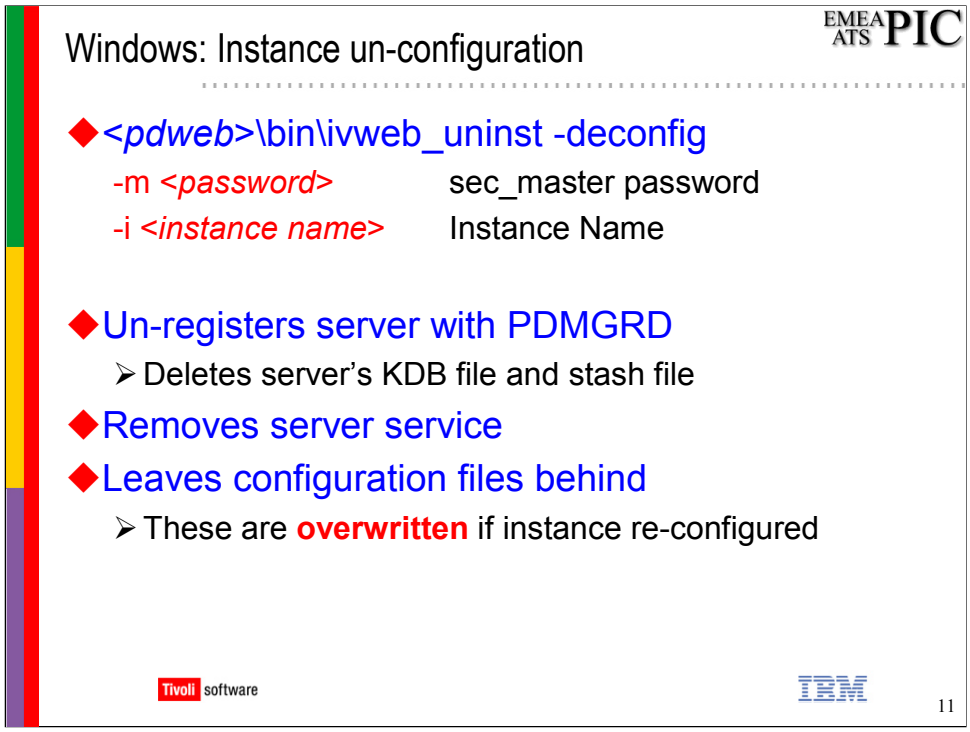
The ***–u*** and ***–U*** parameters specify whether HTTP and HTTP respectively should be enabled or not and the ***–r*** and ***–R*** parameters specify the HTTP and HTTPS port that should be used.

The ***–m*** parameter specifies the password for sec_master. This is required in order to register the new WebSEAL server with the AM Policy Server (PDMgrd). There is no option to use a different admin user.

The ***–M*** parameter specifies the port that the server will listen on for communication from the AM Policy Server. The port for the default WebSEAL instance is 7237 so you need to chose something different.

The ***–i*** parameter specifies the name for the instance. If this parameter is missed out then the default instance will be configured. It is recommended to use the PDCONFIG utility to configure the default instance.

The ***–n*** parameter (if present) specifies the IP address to which the instance should bind. If the parameter is not present the the instance will bind to all IP addresses on the local machine.



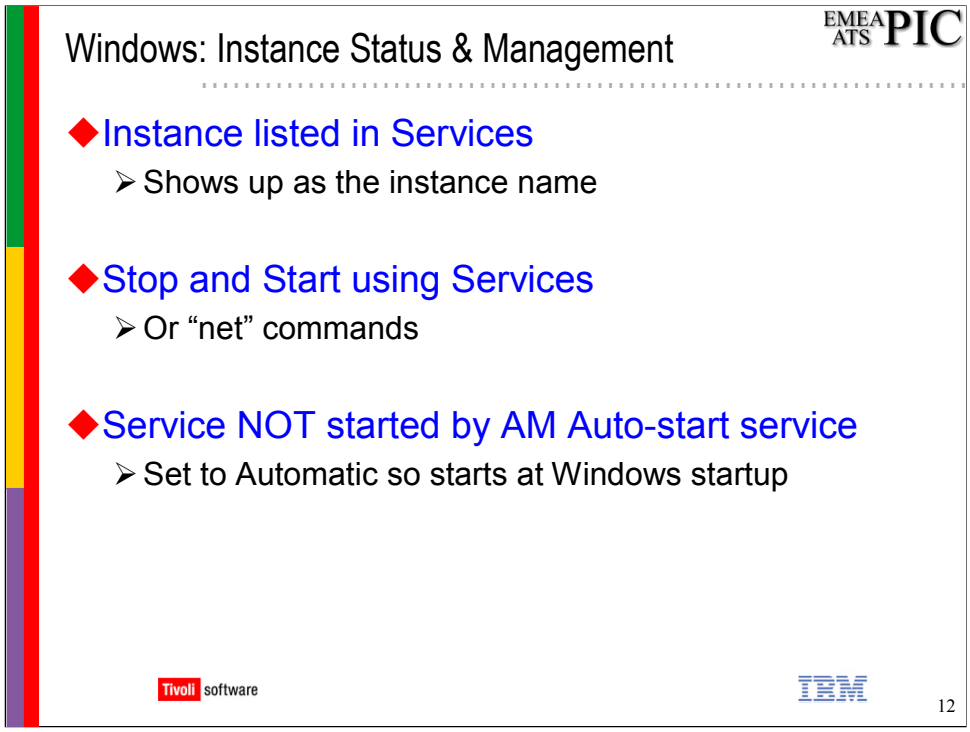
In order to un-configure an additional WebSEAL instance the *ivweb_uninst* command is used. This command must be run with the ***–deconfig*** option otherwise it will cause WebSEAL to be uninstalled (if all instances are unconfigured).

The unconfigure command takes two parameters (in addition to *–deconfig*).:

The ***–m*** parameter specifies the sec_master password. This is required to unregister the server from the AM Policy Server.

The ***–i*** parameter is used to specify the instance name. This is the same name that was used for configuration. If a *server list* shows a WebSEAL server called *webseald-myhost-instance1* then *instance1* is the instance name required for unconfigure.

When an additional instance is unconfigured using the ivweb_uninst command some configuration files are left behind. Note that these are **not** re-used if the instance is re-configured – they are overwritten from the template file.



On Windows systems all Access Manager services, including all WebSEAL instances, are run as services. However, additional WebSEAL instances behave differently from the other Access Manager services.

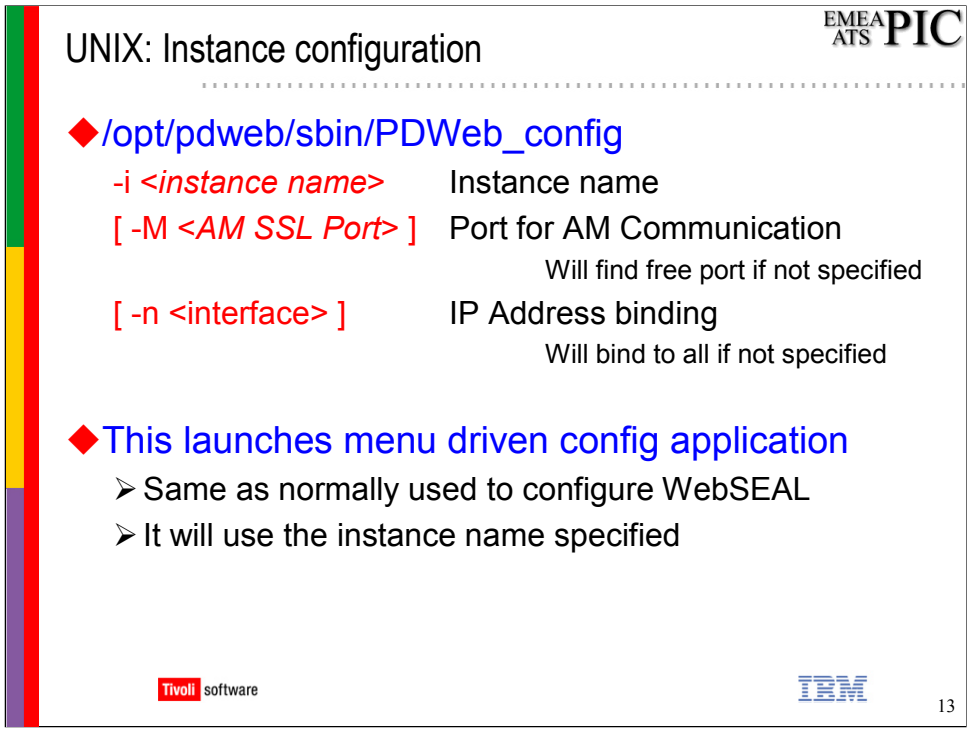
The Default WebSEAL instance is listed in Services as "Access Manager WebSEAL" and has a start-up mode of "Manual". It is started by the "Access Manager Auto-Start" service that is started when Windows starts.

Additional WebSEAL instances are listed with the instance name. They have their start-up mode set to "Automatic" and so they start when Windows starts.

If you want to run a WebSEAL instance in the foreground then this is done using the *–foreground* flag (as with the default instance) but the configuration file must be specifically given to load the correct instance configuration:

…/pdweb/bin>**webseald –foreground –config ../etc/webseald-*name*.conf**

Where *name* is the instance name.

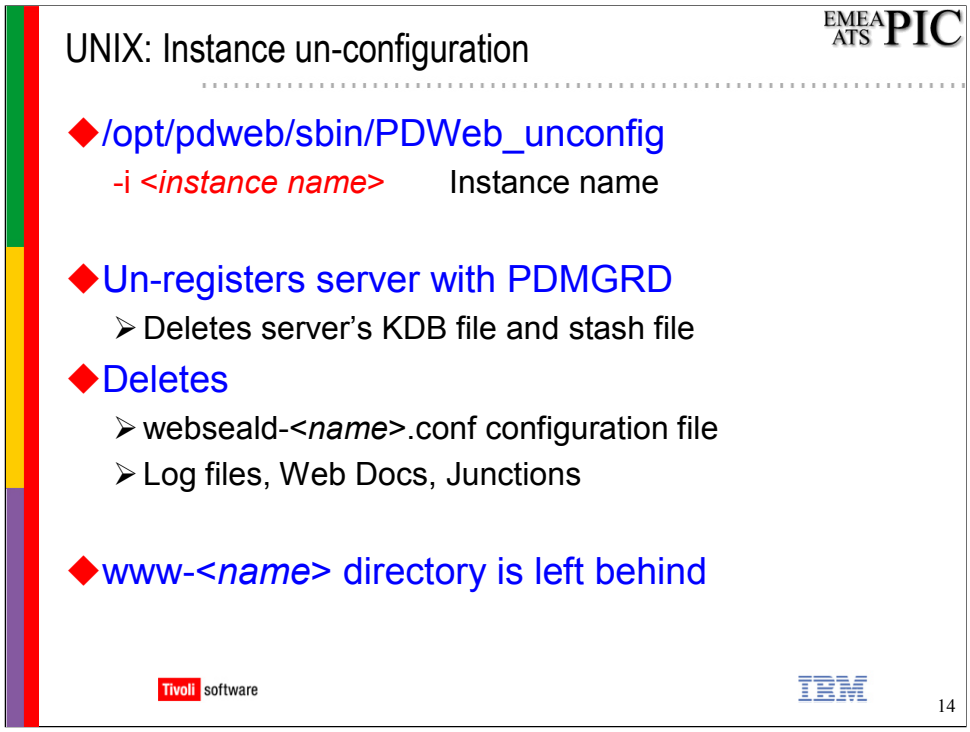


On UNIX systems the PDWeb_config utility is used to configure additional WebSEAL instances. This utility is not in the PATH by default but its location is shown above.

The user will be prompted for most of the configuration options (HTTP/HTTPS ports etc) during configuration rather than having to specify them on the command-line as in Windows. The only required parameter when configuring additional instances is the ***–i*** parameter which indicates the instance name.

If you wish to specify a particular port for the instance to listen on for communication from the Management Server then this can be given with the ***–M*** parameter. If this is not present then a free port will be found.

The ***–n*** parameter (if present) specifies the IP address to which the instance should bind. If the parameter is not present the the instance will bind to all IP addresses on the local machine.



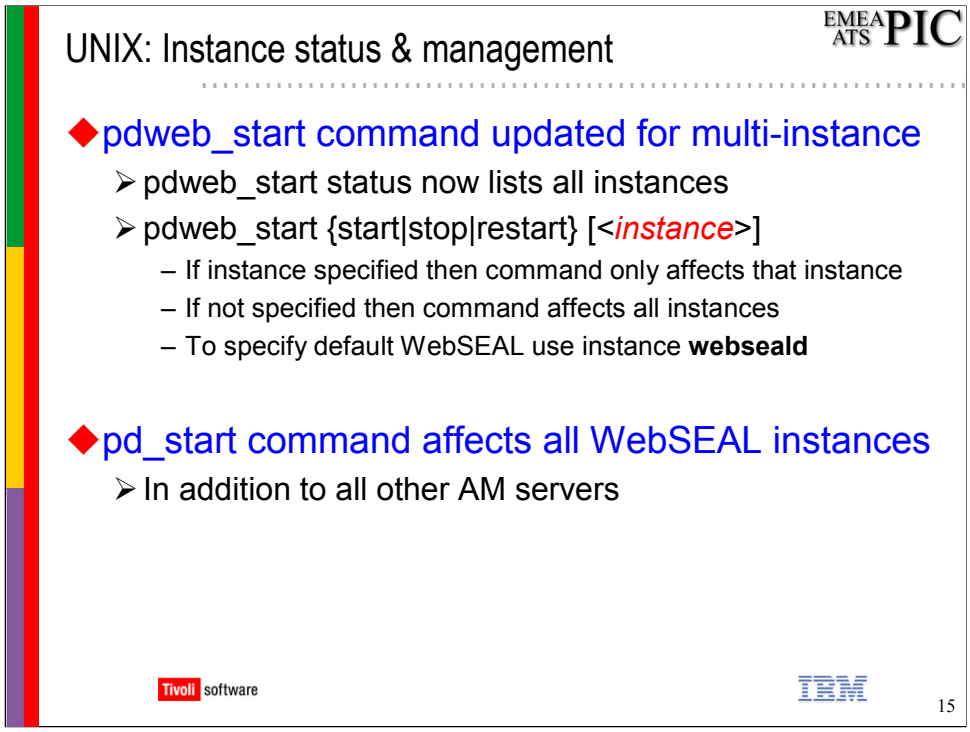
On UNIX systems the PDWeb_unconfig utility is used to un-configure additional WebSEAL instances. This utility is not in the PATH by default but its location is shown above.

The only parameter that is required for this command is the ***–i*** parameter which specifies the instance name. This is the same name that was used for configuration. If a *server list* shows a WebSEAL server called *webseald-myhost-instance1* then *instance1* is the instance name required for unconfigure.

The un-configuration utility will prompt for the password of sec_maser.

The un-configuration unregisters the server instance from the AM Policy Server and also deletes most of the configuration files.

The www-*name* directory is left behind but most of the files it contained are removed. It can be deleted once un-configuration is complete. If it is not removed and the instance is re-configured then any remaining files will be overwritten.



WebSEAL on UNIX has the pdweb_start command which is used to start, stop, re-start and check the status of WebSEAL.

When pdweb_start is used to check status, all WebSEAL instances will be shown. When using pdweb_start to start, stop or re-start WebSEAL instances a number of choices exist:

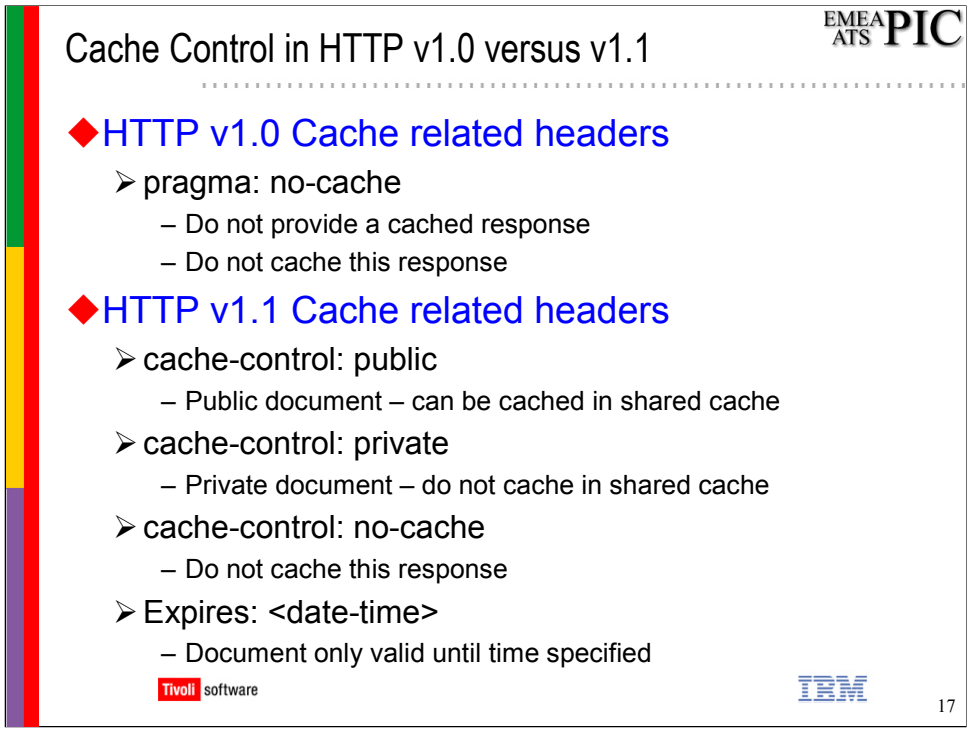
If no instance is specified then the command will affect all instances on the machine.

If a named instance is specified then the command will only affect that instance.

If the instance name given is webseald then this will affect the default instance.

The pd_start command, which is used to start, stop, re-start and check the status of all Access Manager servers on a machine will show and affect all configured WebSEAL instances.

This next section details how WebSEAL has been enhanced so that it is able to make HTTP v1.1 requests to junctioned back-end servers so that they are able to include HTTP v1.1 directives in their response. There is no detailed discussion of the HTTP v1.1 standard itself – if this information is needed then the reader is directed to the IETF RFC 2616.



The slide above illustrates that the cache control function available in HTTP v1.1 is far superior to what is available in HTTP v1.0.

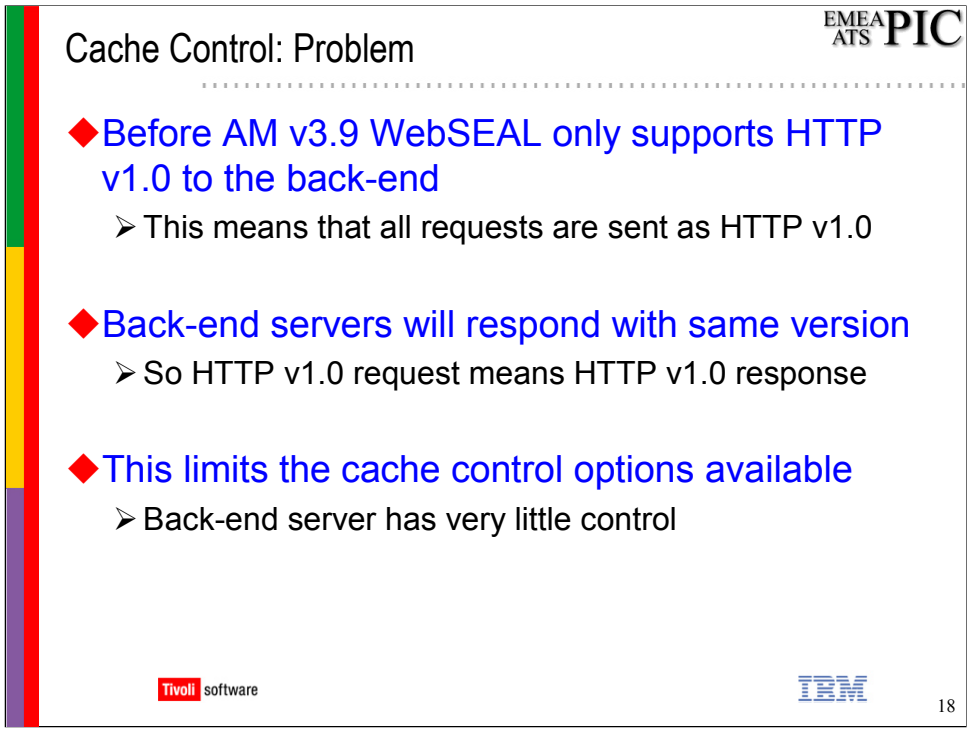
Cache control is important for two main reasons:

Better Caching = Better Performance

If a cache can be used more effectively then it increases the performance of the entire solution because retrieving information from a cache is far quicker than making a request all the way to a back-end server. The more fine-grained the control of the cache can be (expiration time etc) then the more information can be cached.

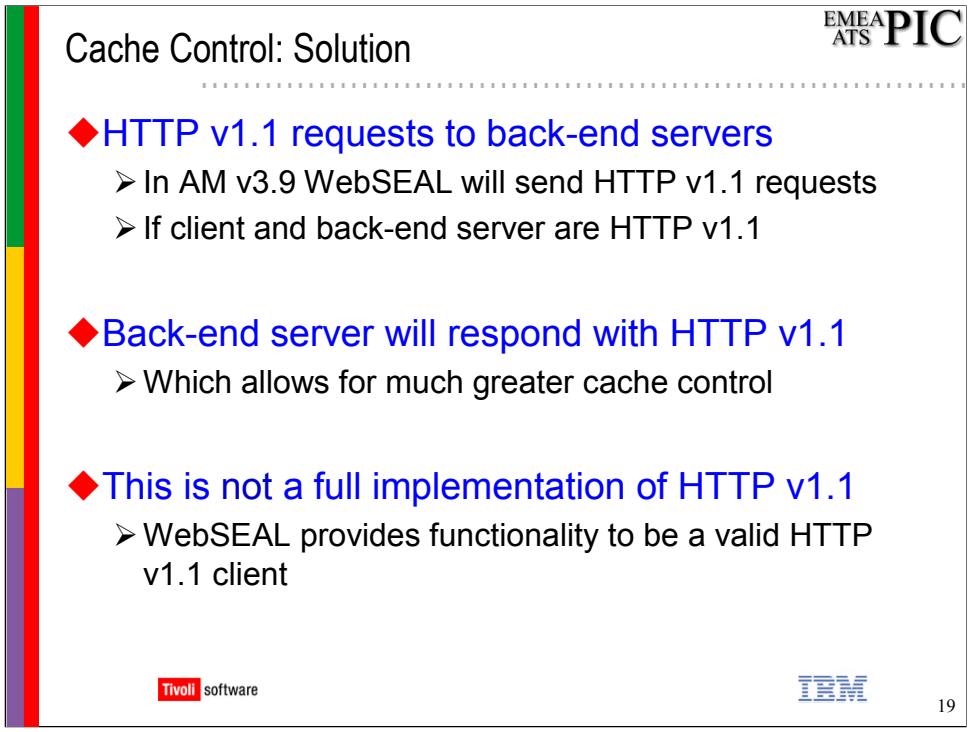
Security Issues

In an environment where HTTP is used for part of the path to the client (a separate VPN solution is used for encryption for example) it is important to be able to indicate which information should be considered private (and not cached for everyone) and which should be considered public (and can be cached for all users). This kind of cache control is not possible with HTTP v1.0.



The problem that exists in WebSEAL before AM v3.9 is that it only supports HTTP v1.0 to junctioned back-end servers. This means that all requests sent to back-end servers indicate that the client (WebSEAL) is HTTP v1.0.

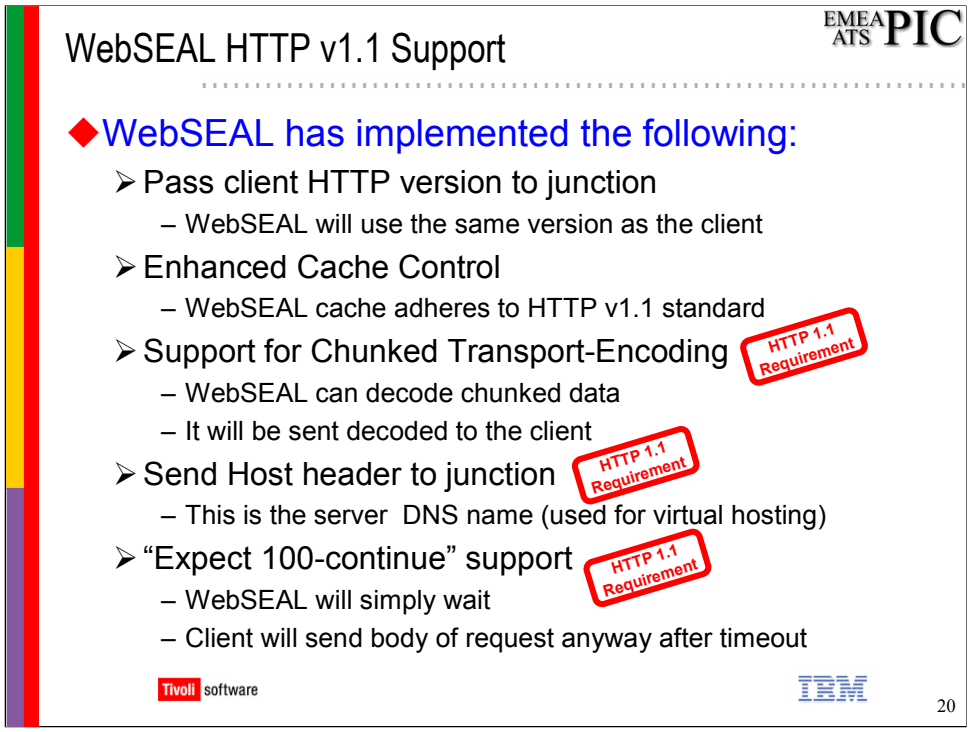
When a WEB server receives an HTTP v1.0 request it has to assume that the client can only understand an HTTP v1.0 response. This means that the WEB server is unable to use any advanced features of HTTP v1.1. In particular, none of the advanced cache control features of HTTP v1.1 are available.



The solution to this problem is for WebSEAL to send HTTP v1.1 requests to the back-end server. In AM v3.9 WebSEAL has been enhanced so that it is now capable of making HTTP v1.1 requests to junctioned back-end servers.

Since the back-end servers are now receiving HTTP v1.1 requests they are now free to respond using HTTP v1.1. They can use any of the cache control features that are part of the HTTP v1.1 standard.

It is important to note that although the HTTP v1.1 standard specifies a large number of features above those supported in HTTP v1.0 most of these are optional and are not required in order for a client to be capable of handling HTTP v1.1 responses. WebSEAL supports only those parts of HTTP v1.1 that are required so that it can communicate with servers using HTTP v1.1. These are discussed on the next page.



The slide above shows the features of HTTP v1.1 that WebSEAL has implemented in order to allow it to communicate with an HTTP v1.1 server.

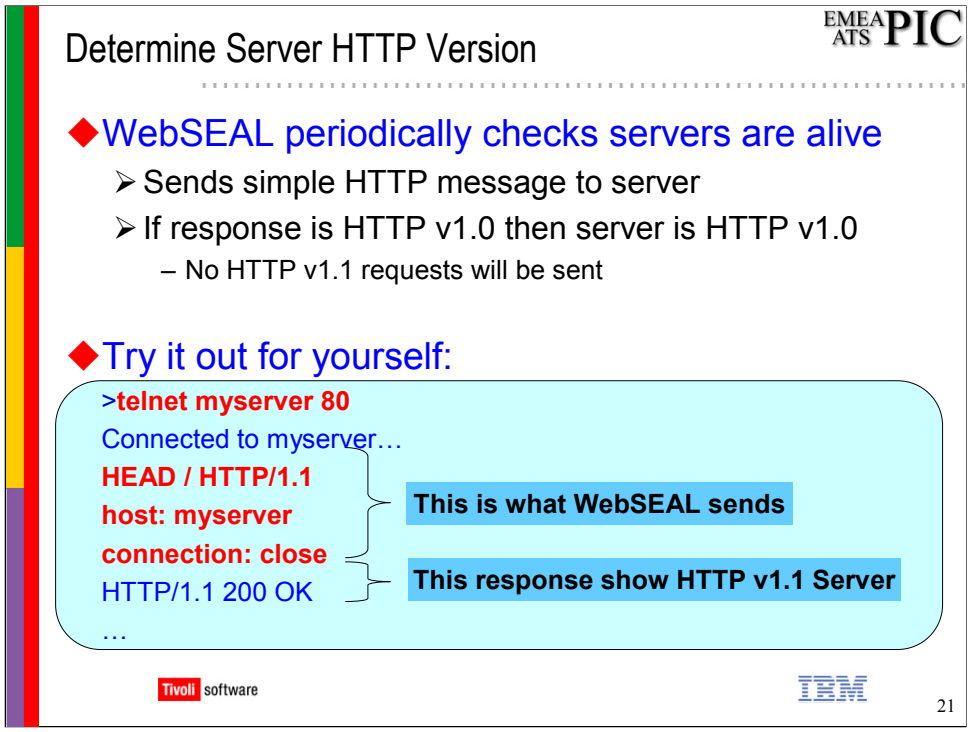
The first thing that WebSEAL must do is to tell the server in its requests that it is expecting an HTTP v1.1 response. WebSEAL will always do this if the end-user browser is capable of handling HTTP v1.1 (which is certainly the case for recent IE and Netscape browsers).

WebSEAL has a cache that it uses to store data from the back-end servers. This cache can interpret HTTP v1.1 cache control directives.

An HTTP v1.1 client MUST be able to decode "chunked" data. There are many other possible encodings but this is the only one that is required. When WebSEAL receives chunked data it will decode it as it is received so that it can be processed. It is sent onto the client without being re-encoded.

An HTTP v1.1 request must include the Host header that indicates the DNS name that the client is making the request to. This header is used in virtual hosting. WebSEAL has always sent this request to back-end servers.

In HTTP v1.1 a client may send just the request headers to the server and specify that it will wait for a 100-continue message before sending the body of the request. WebSEAL does not send the 100-continue message – it just waits for the client to send the body anyway (which it will do eventually after a timeout).

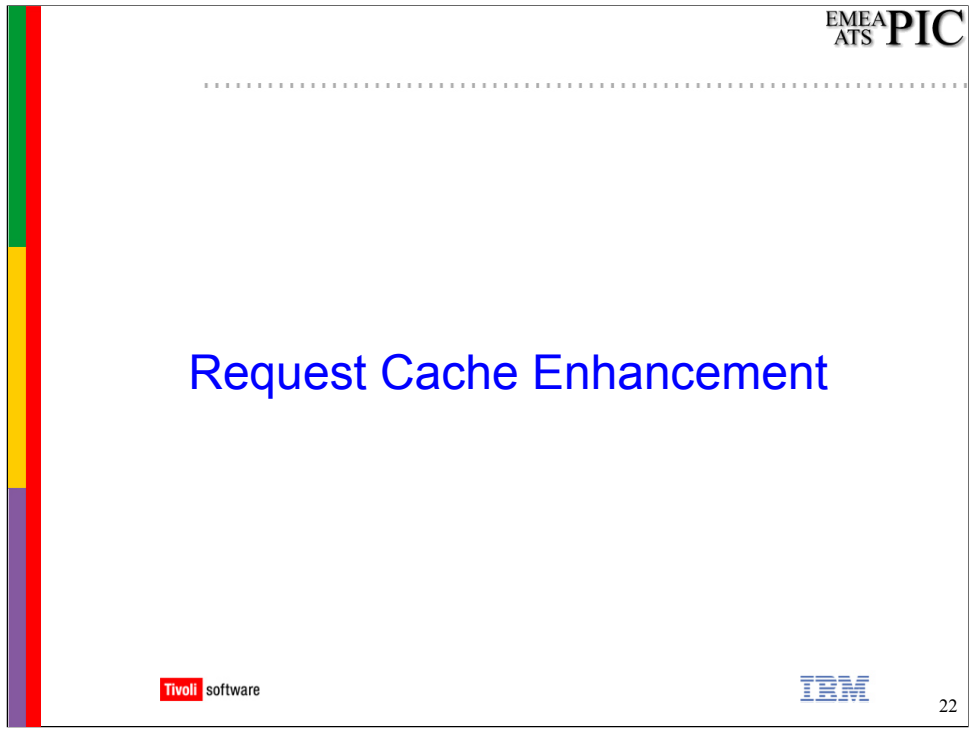


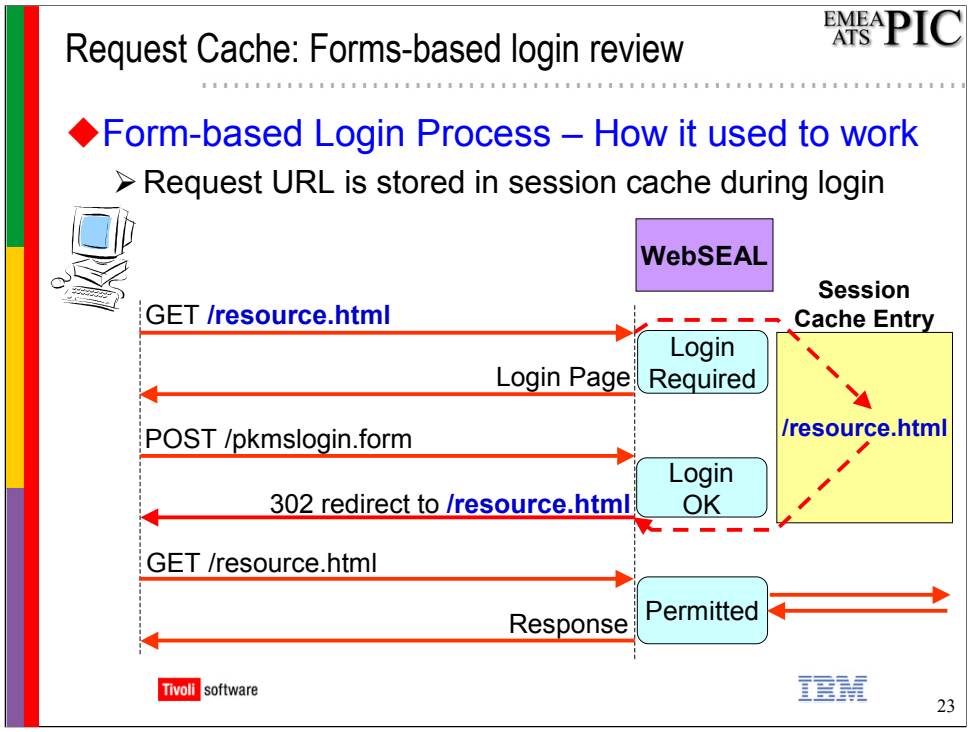
Since it is now possible for WebSEAL to send either HTTP v1.0 or HTTP v1.1 requests to junctioned back-end servers it must have a way to determine which the server can handle.

WebSEAL already polls the junctioned back-end servers to determine their availability (every 300 seconds as per the *ping-time* option in webseald.conf) and the response to this ping is used to check the HTTP version that the server supports.

The screenshot above shows the HTTP message that is sent to server by WebSEAL and the response that it will get back if the server suppots HTTP v1.1. In this case WebSEAL will send requests to the back-end using the same HTTP version that is used by the end user (which will normally be HTTP v1.1).

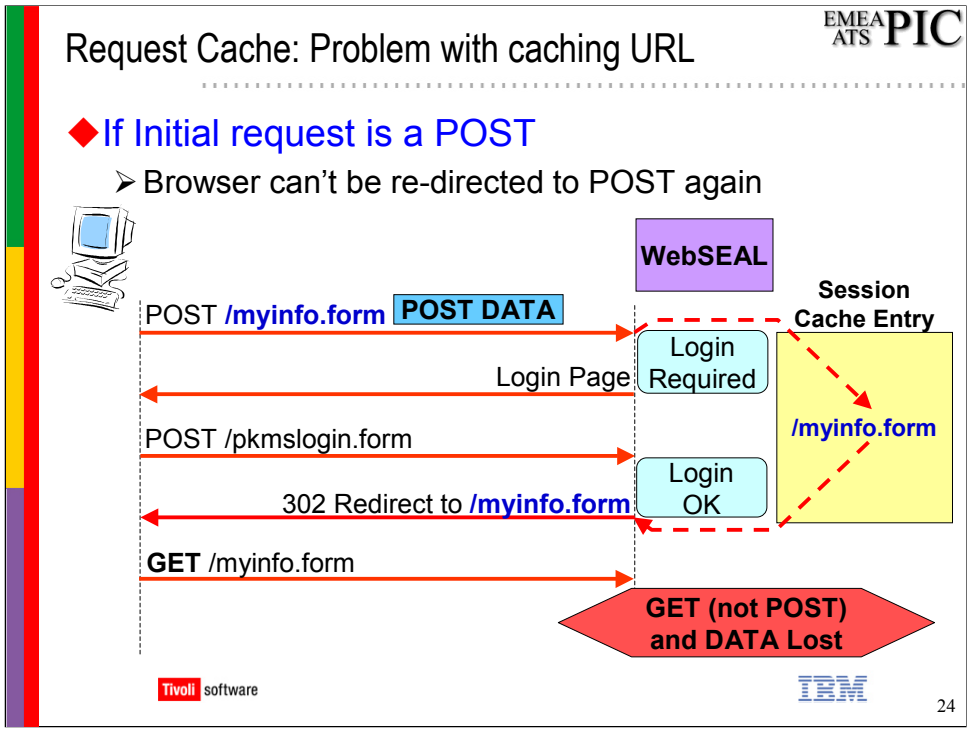
If the server responds with HTTP v1.0 to this HTTP v1.1 message then WebSEAL will forward all requests using HTTP v1.0 regardless of the version used by the end user.


EMEA ATS PIC
Request Cache Enhancement
Tivoli software
IBM
22



The slide above shows how WebSEAL (before v3.9) is able to re-direct a user back to the original resource they requested after performing a form-based login. This functionality is required for form-based login (as opposed to BA login) because form-based authentication is not a standard part of HTTP communication. The browser is unaware that the pages are part of a login mechanism - as opposed to just being normal pages – and so it will not automatically re-try the request for the requested resource once the login is complete (as it does after BA authentication).

When WebSEAL realises that it needs to authenticate the user the URL they requested is stored in the session cache. Once the login is successfully completed this URL is sent as a re-direct and so the browser makes a new request for the resource which is now permitted – assuming the user has appropriate permissions.



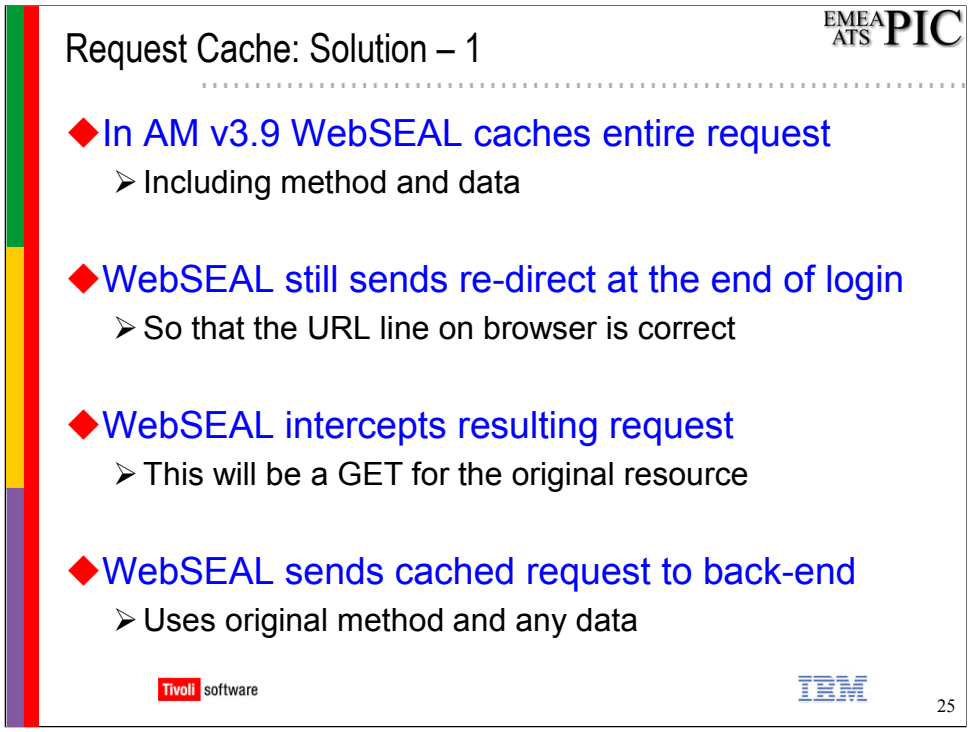
The problem with the process described on the previous page is that only the URL that the user requested is stored in the credential cache. Any data that was included in the body of the request is not stored. Data may be sent with either a POST or a GET but the most common time data is sent to a server is when the user has filled out a form and is performing a POST to send that information to the server.

A user will be authenticated when making a POST if the POST target is the first protected page they have accessed or if their session has timed out while they were completing the form.

Following successful authentication WebSEAL sends the user a re-direct back to the URL that prompted the authentication. There are two problems here:

1. WebSEAL sends a 302-Redirect to the original resource. In most cases this will cause the client to sent a GET request for that resource. This may not be correct if the original request was a POST. This problem is not easy to solve because the HTTP specification has often been misinterpreted in this area.

2. The browser as no way of knowing that it needs to resend the request body that was sent in the original request. The browser does not know that this re-direct is in any way linked to the original request it made before. The result is that the data is lost – which will not please the end user ! They will most likely receive an error from the back-end server.



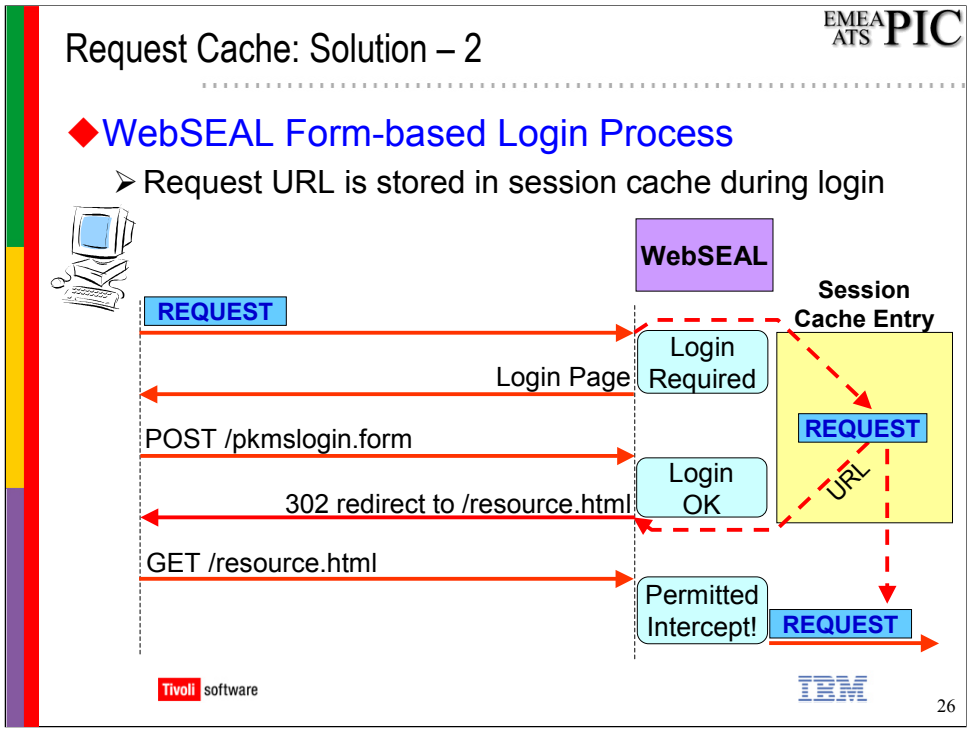
A solution to the problem is for WebSEAL to cache the entire user request rather than just the URL. This includes the HTTP Method (POST/GET etc) that was used and also includes and message body – where any data is found.

Now that the user request is cached at WebSEAL, the next question is when to use this to re-issue the users request.

This could be done after successful login and the response from the back-end returned to the user as the response to the login POST. This avoids the need for a re-direct but it means that the URL in the browser will be incorrect (it will show the post action).

What actually happens is that WebSEAL still sends the browser a re-direct to original URL but it then intercepts the potentially bad (wrong method/no data) request that is returned and forwards the cached request to the back-end instead. The response to this new request is then returned to the browser.

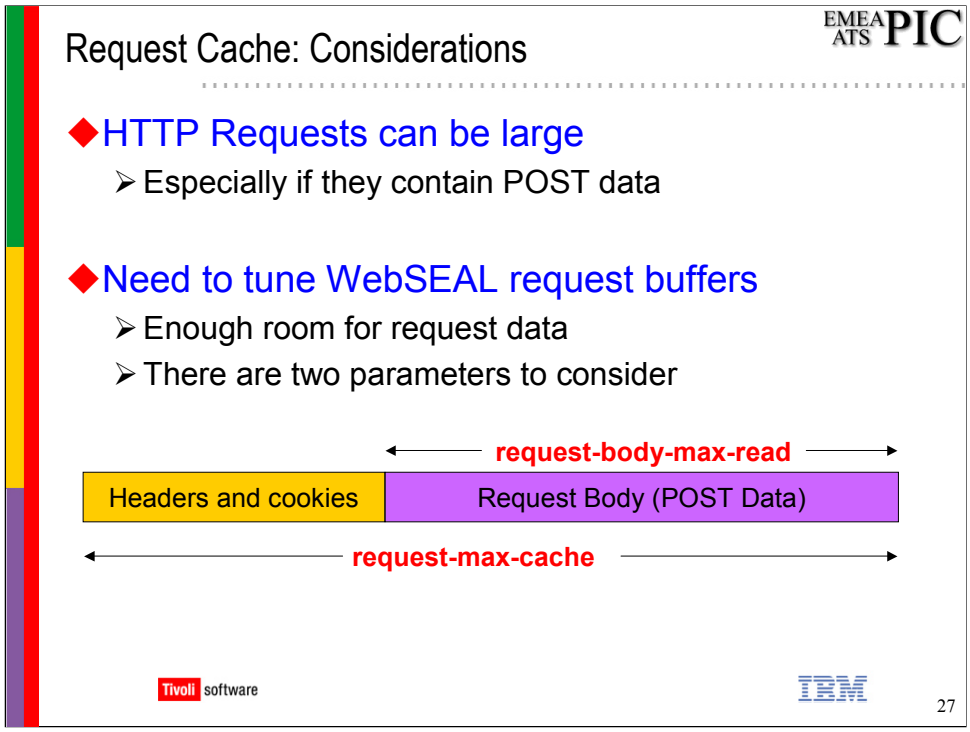
A diagram demonstrating this process is shown on the next page.



The diagram above shows how WebSEAL in AM v3.9 handles request caching during form-based authentication.

The process is similar to what was done before v3.9 except that now the entire user request is stored in the session cache instead of just the URL.

When the browser makes the request for the original resource following successful authentication and re-direct WebSEAL intercepts this and sends the cached request from before the authentication instead. This cached request contains all of the original request data that the browser sent in the original request which may be missing in the second request.

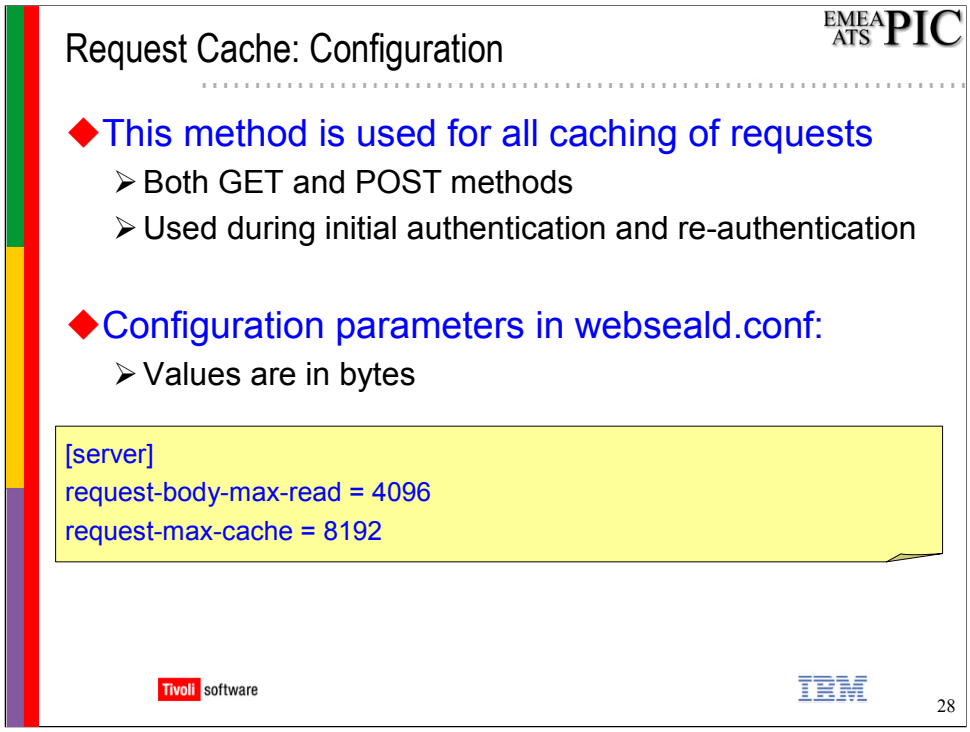


HTTP requests can be large – especially if they contain a large amount of POST data (perhaps from a long form).

If WebSEAL is to store all of the information in the request during authentication then sufficient room must be allocated in the cache entries to allow this.

The diagram above shows the two parameters that affect the amount of data that can be stored in a session cache entry. *Request-body-max-read* specifies the maximum size of the request body and *request-max-cache* specifies the maximum size of the whole request. The difference in these two values is the size of the request headers (and any cookies).

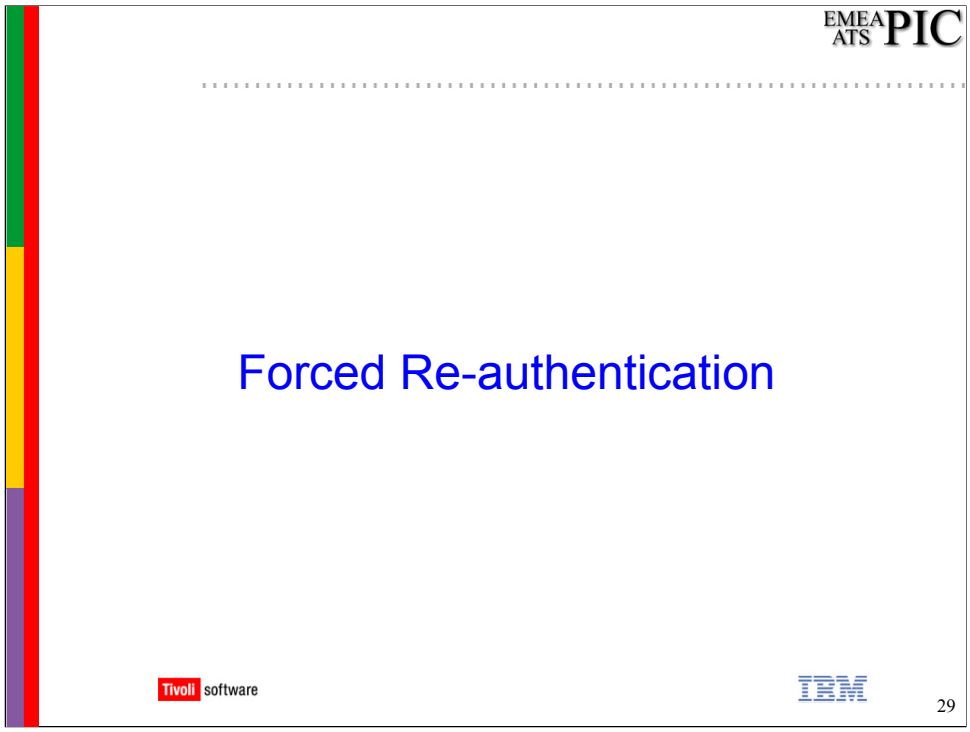
Note: WebSEAL does not have to cache **all** requests. Only requests that result in an authentication need to be cached.

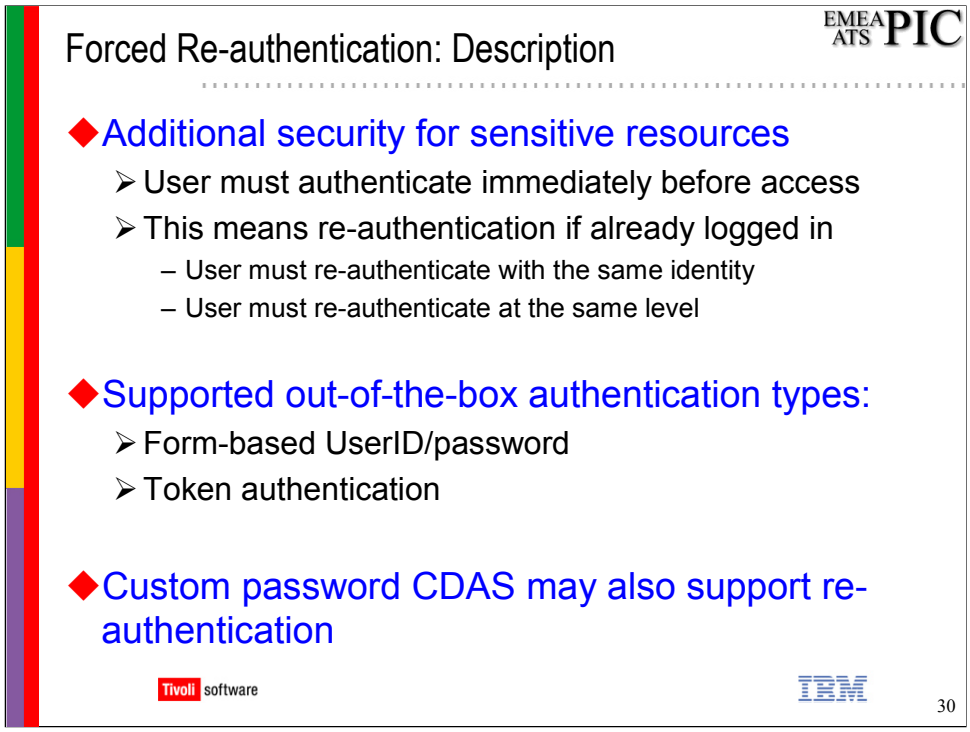


In WebSEAL v3.9 the method of caching the entire request (as opposed to just the URL) during authentication is used in all cases. This is not configurable.

The *request-body-max-read* and *request-max-cache* parameters are set in the *[server]* stanza of webseald.conf. The values shown above are the default values.

The defaults allow for a total request size of 8192 byes. A maximum of 4096 bytes is available for the request body.


EMEA ATS PIC
Forced Re-authentication
Tivoli software
IBM
29

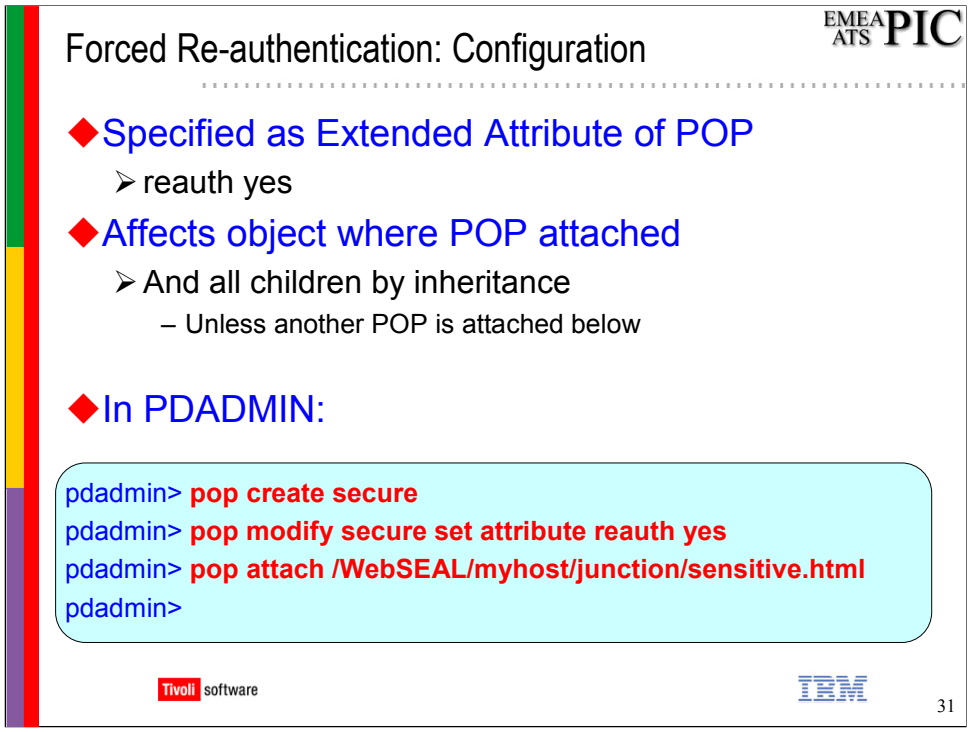


In order to ensure that the user accessing a sensitive resource is the same person that initially authenticated at the start of a session it is often desirable to ask them to re-authenticate before allowing access. This can also be used for confirmation that they REALLY want to perform some invasive action (rather than quickly clicking "YES" to "Are you sure?").

In WebSEAL v3.7.1 the concept of Step-Up authentication was introduced (which allows access decisions to be made based on the authentication method used) but this did not provide the ability to force a re-authentication at the **same** level.

Not all authentication types are supported for re-authentication. Basic Authentication, for example, cannot support re-authentication because the UserID and password for the user are submitted with every request – so the end user is not re-authenticated.

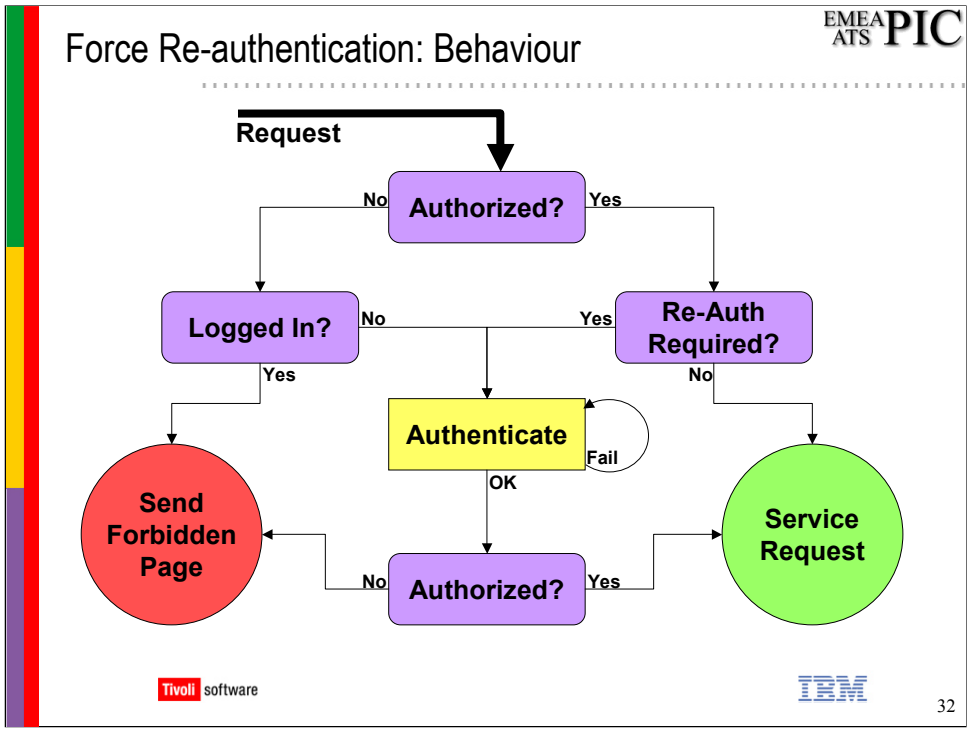
Of the out-of-the-box authentication mechanisms only form-based authentication methods (UserID/Password and Token Authentication) are supported for re-authentication. Custom UserID/Password CDAS modules can also work.



Forced re-authentication is configured by attaching a POP to the object in question that contains an extended attribute with the name *reauth*. The value of the attribute is not important (yes is used in the example above). This can be done either using the WPM or (as shown above) using PDADMIN.

It is worth noting that all children are affected by the POP (by inheritance) but this does not mean that a single re-authentication will allow access to all the children. The user must re-authenticate to access each one.

If non-html objects (such as pictures) are protected by forced re-authentication then they will not be displayed. This is because there is no way for a request for a picture that is part of a page to return a login form. The administrator must ensure that re-authentication is not required for these objects.



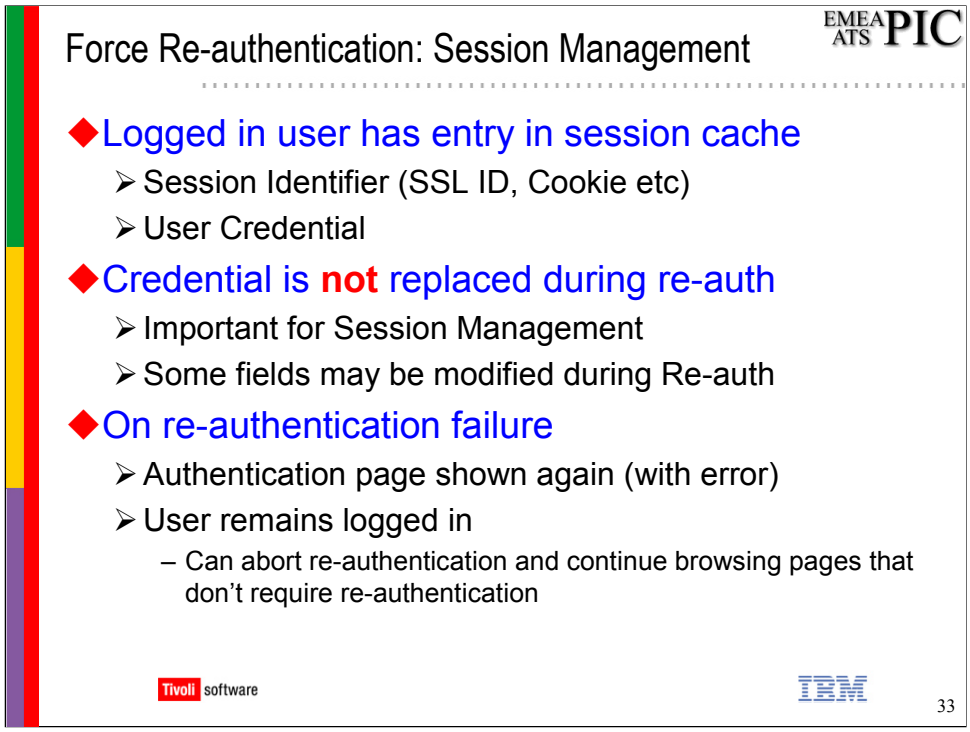
The flowchart above shows the behaviour of WebSEAL with Forced Re-authentication enabled.

When a request is received it is authorized. Before AM v3.9, if this were successful, the request would be serviced. However, in AM v3.9, an additional check is done to see if the object being requested requires re-authentication. If it is required (*reauth* extended attribute in the POP) then a login form is returned to the user instead of the requested resource. Unless the user can re-authenticate the request will not be serviced.

During the re-authentication step the users original request is stored in the request cache and is re-submitted after authentication (as described elsewhere in this presentation).

If an unauthenticated user tries to access a protected resource they will have to authenticate. Once this is done there is no need to re-authenticate them because they have only just authenticated anyway.

If the user is not authorized to access the resource they will receive a forbidden page.



Re-authentication differs from initial authentication because the user is already logged in to WebSEAL and so already has a session cache entry containing a credential.

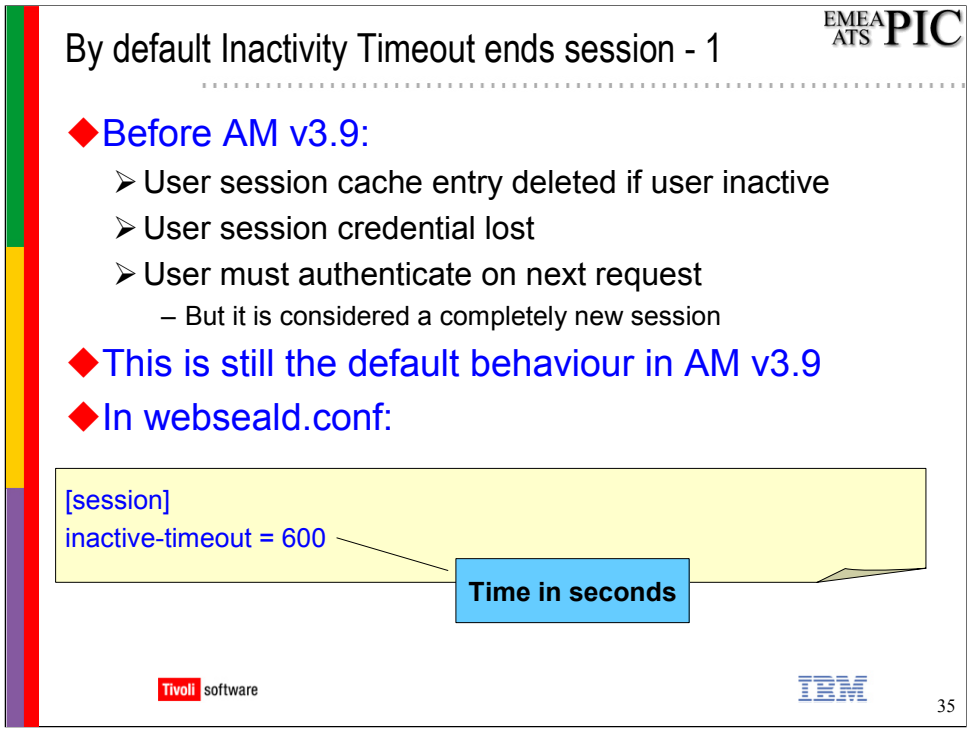
It is important to know that this credential is not replaced during re-authentication. This means that any session information stored in the credential (discussed later) will not be affected by the re-authentication.

The existing credential is available to the authentication mechanism during re-authentication and so it may be used during the re-authentication and may have certain fields (custom attributes) modified by the authentication mechanism during the re-authentication.

If forced re-authentication fails then the user will be denied access to the resource they were trying to reach but they will **not** be logged off. Their credential is still valid and they can abort the re-authentication process (by requesting some other URL) and still access other pages that do not require forced re-authentication.

If a user session is inactive for a configured period of time it has always been the case that WebSEAL would require the user to re-authenticate before allowing access to protected resources.

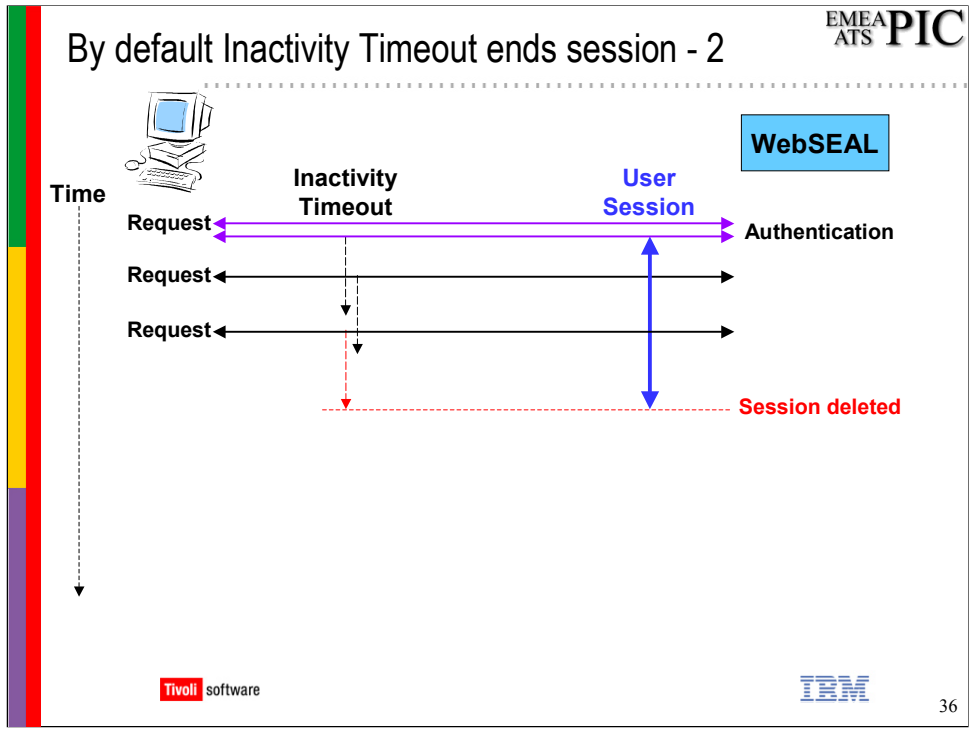
The changes to this functionality in AM v3.9 do not change the behaviour that a user sees but they do alter the way that WebSEAL handles a user session if it is inactive for more than a configured period of time.



Before AM v3.9 a user session would simply be removed from the session cache if the inactivity timeout was exceeded. This means that the user is effectively logged out from WebSEAL and so must authenticate again in order to access any protected resources. When they perform this second login WebSEAL considers this a completely new user session. A new session cache entry is generated and a new credential is built for the user.

This is still the default configuration for WebSEAL in AM v3.9

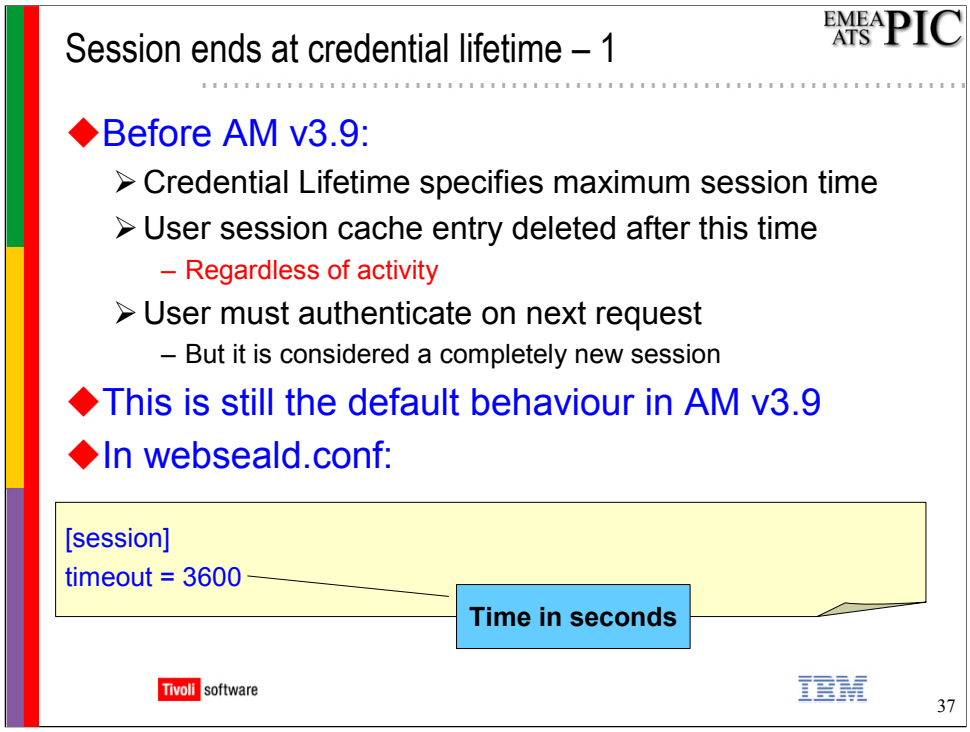
The inactivity timer for all WebSEAL sessions is set in webseald.conf as shown above. The default is 10 minutes (600 seconds).



The diagram above shows user communication with WebSEAL, how this affects the inactivity timeout and how the inactivity timeout causes the user's session to be deleted if it expires. This is the default behaviour in AM v3.9.

When a user initially authenticates the inactivity timeout is reset to the value configured in webseald.conf. Each time the user makes a request the inactivity timeout is reset again.

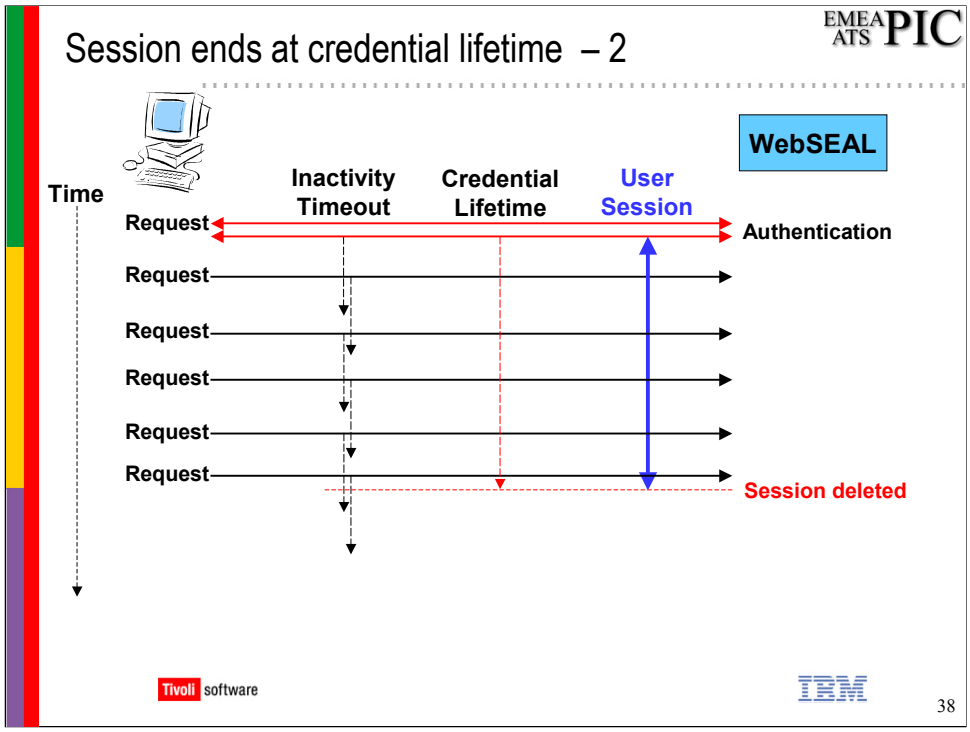
If the timeout expires then the users session is deleted from the session cache. Their session is over. If they later authenticate again to access more protected resources WebSEAL will consider this a new session.



In addition to the inactivity timeout WebSEAL also specifies a credential lifetime. This effectively specifies the maximum session time – regardless of activity. When the credential lifetime expires the user session is deleted from the session cache. This means that the user is effectively logged out from WebSEAL and so must authenticate again in order to access any protected resources. When they perform this login WebSEAL considers this a completely new user session. A new session cache entry is generated and a new credential is built for the user.

This is still the default configuration for WebSEAL in AM v3.9

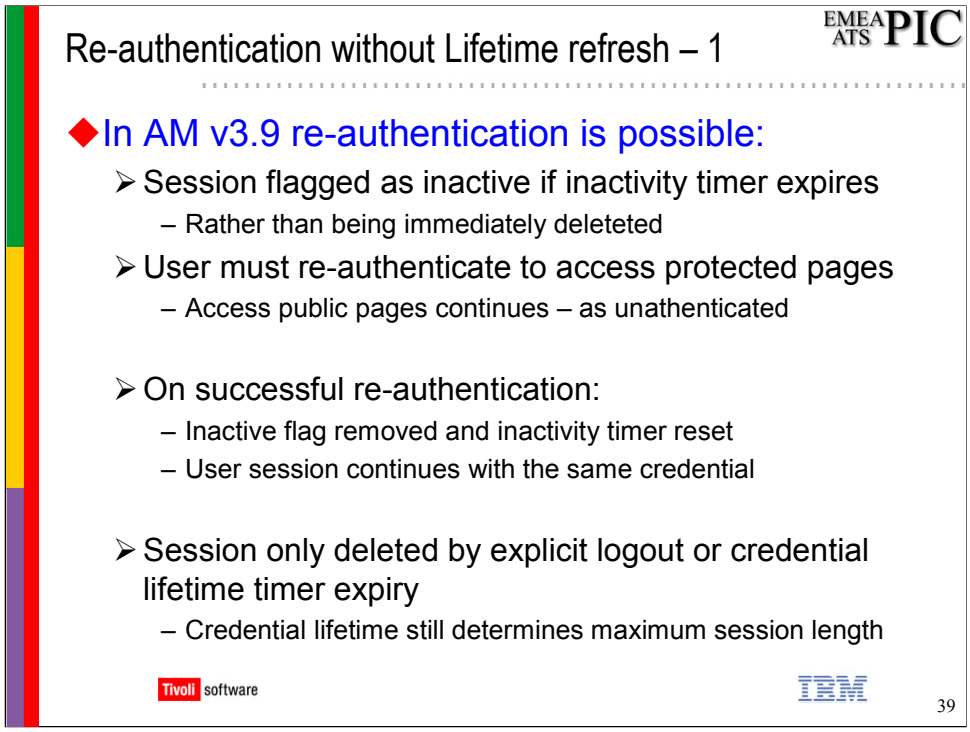
The credential lifetime is configured using the *lifetime* parameter in webseald.conf. The default lifetime is 1 hour (3600 seconds).



The diagram above shows how the credential lifetime affects a users session when using default WebSEAL configuration.

When the user initially authenticates the credential lifetime is reset.

Even though the inactivity timer never expires the session is deleted anyway when the credential lifetime is reached. At this time the user's session is over. In order to access any protected resources they must authenticate again but WebSEAL will consider this a completely new session.

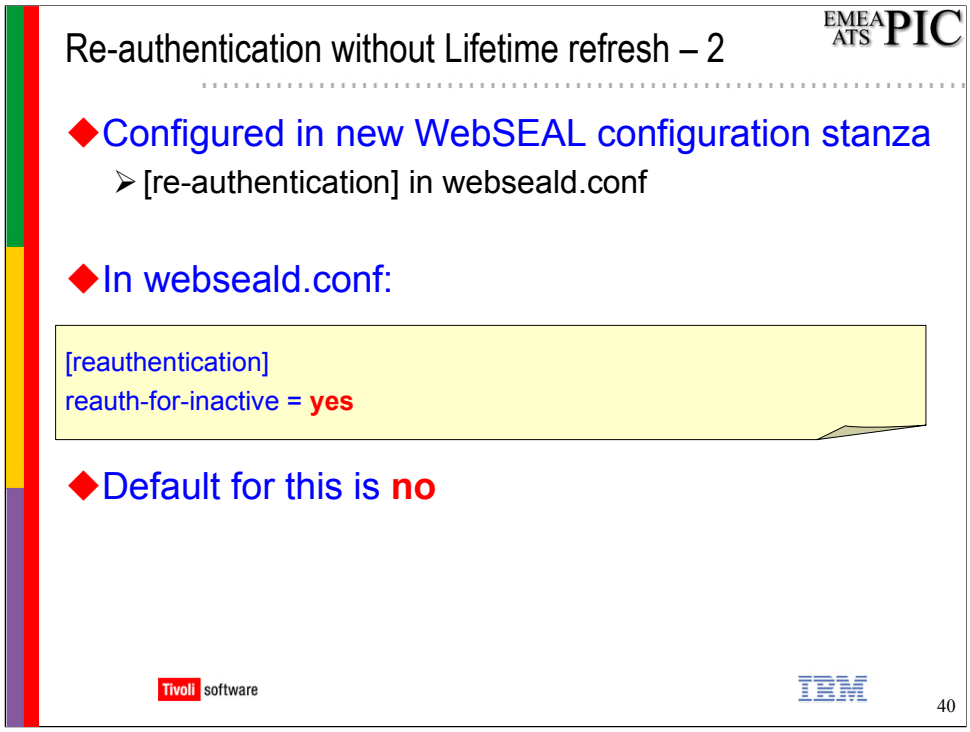


In AM v3.9 there is now the option for WebSEAL to flag a session where the inactive timeout has expired as inactive rather than deleting from the session cache. A session marked as inactive cannot be used for accessing protected resources – and authorizations are done as unauthenticated.

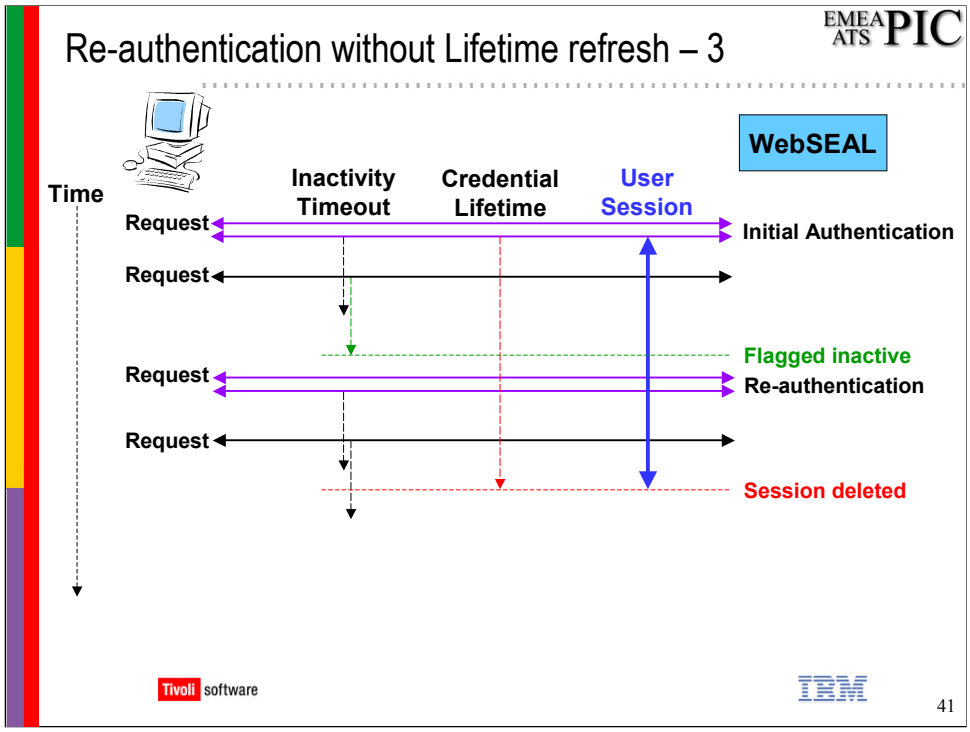
The effect for the user is the same as before – they must authenticate again before they can access any protected resources.

The difference internally to WebSEAL is that the credential has not been removed from the cache and so when the user re-authenticates the inactive flag can simply be removed and the user can continue accessing WebSEAL **using the same session entry and credential**. This is important because it means that any User session data (see later) is maintained.

The only way that a user session will expire with WebSEAL configured in this way is if the user explicitly logs out, if they are forced out (see later) or if the credential lifetime of the session expires. The credential lifetime still determines the maximum session length – when it is reached the session will end regardless of activity.



To configure WebSEAL to mark inactive sessions rather than deleting them from the credential cache change the *reauth-for-inactive* parameter to *yes* in the *[reauthentication]* stanza.



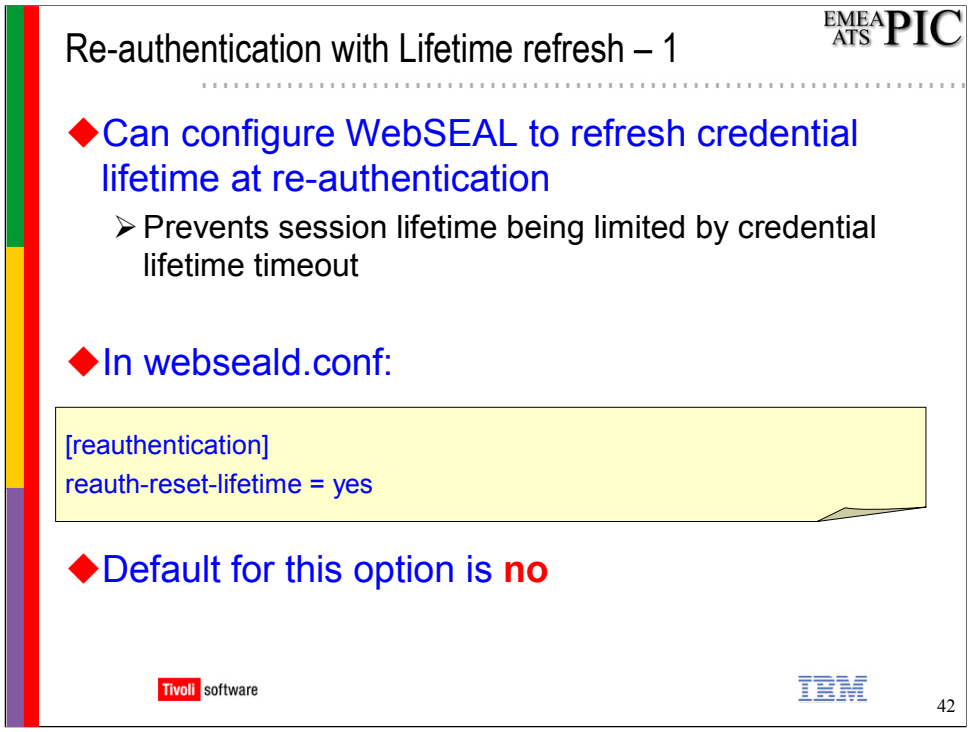
The diagram above shows how WebSEAL handles a session when re-authentication of inactive sessions is enabled as shown on the previous page.

When the user initially authenticates the inactivity timeout and credential lifetime are both reset. As long as the session remains active (inactivity timeout does not expire) nothing is different from before.

When the inactivity timer expires the session is marked inactive. If the user tries to access a protected resource before the credential lifetime expires they will have to re-authenticate but rather than having to build a new credential WebSEAL simply removes the inactive flag from the existing session which re-activates it.

Although the user can re-authenticate after inactivity their session still has a fixed maximum length determined by the credential lifetime. When this expires the session will be deleted from the session cache.

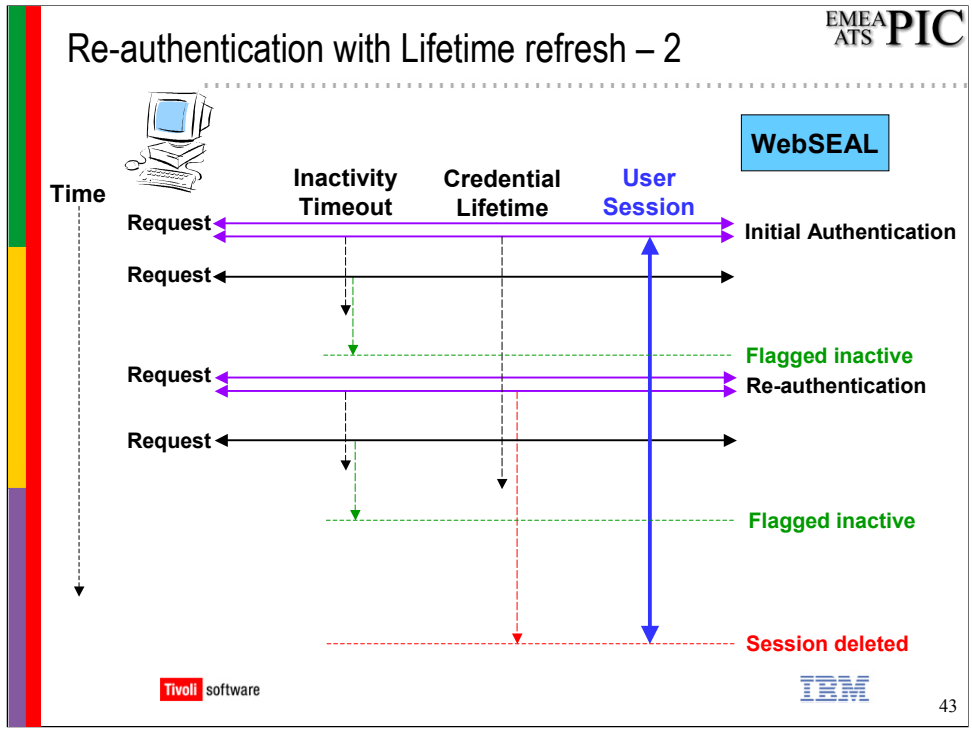
The user can authenticate again after this but this will be a new session with a new credential.



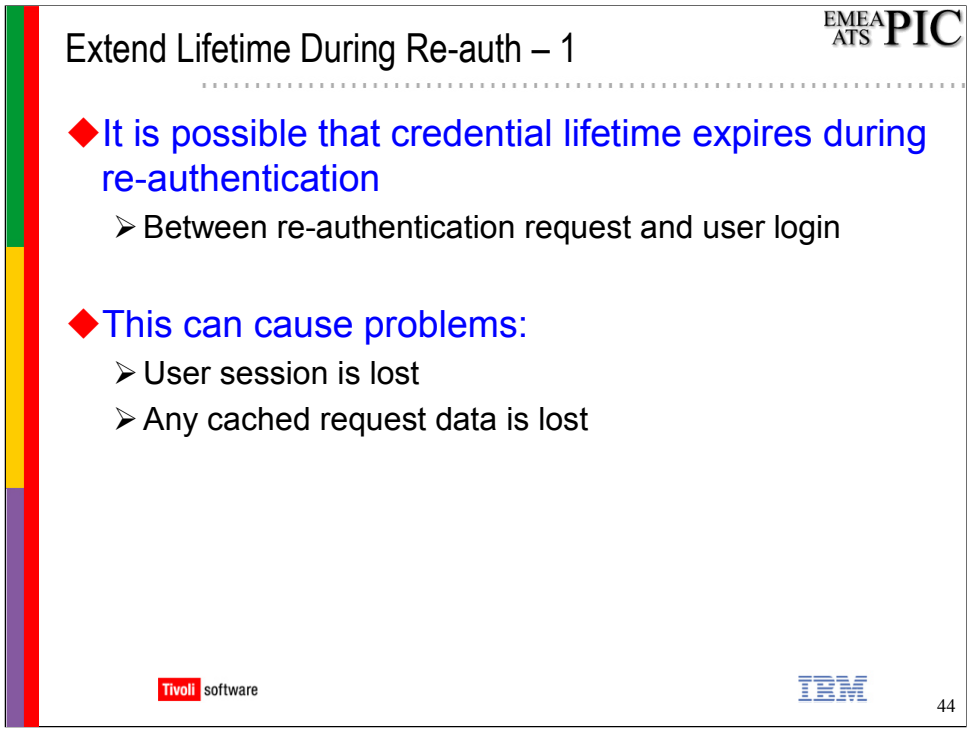
In order to prevent the session being limited by the credential lifetime it is also possible to configure WebSEAL to reset the credential lifetime each time the user re-authenticates.

This is configured in webseald.conf by setting the *reauth-reset-lifetime* parameter to *yes* in the *[reauthentication]* stanza.

Note: The credential lifetime is only reset during a re-authentication. This means that if the user is constantly active and the inactivity timer does not expire (and therefore no re-authentication is required) the lifetime will not be refreshed, the credential lifetime will be reached and the session will end.



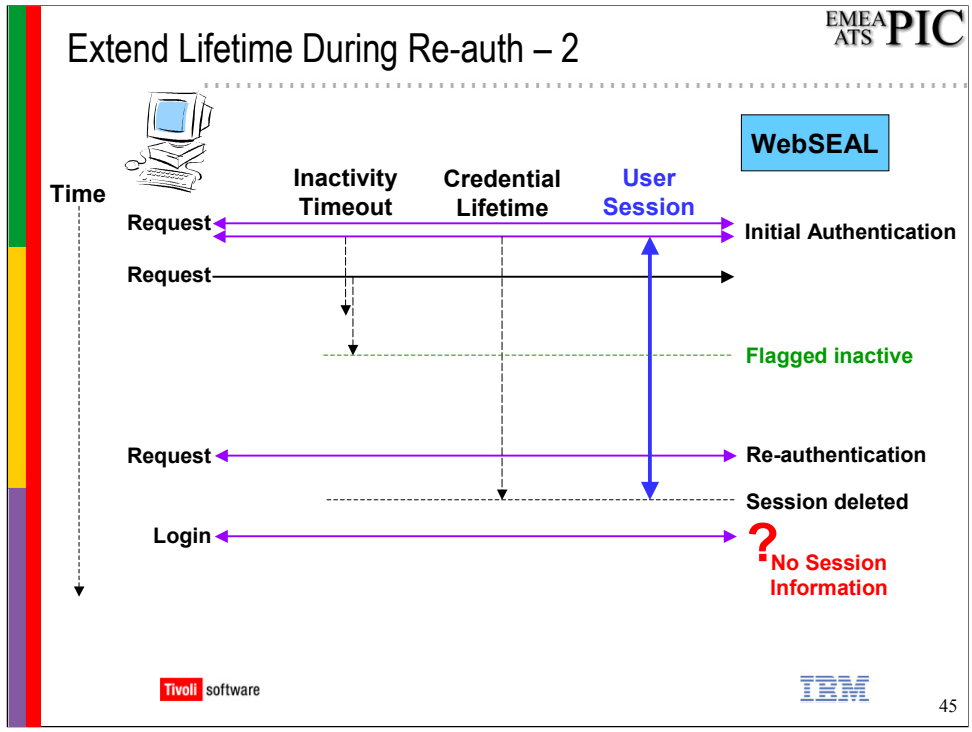
The diagram above shows how the credential lifetime is refreshed at each re-authentication when the *reauth-reset-lifetime* parameter is set. This means that the session no longer has a maximum lifetime. The session will only end if the credential lifetime is reached without the user re-authenticating or if the user specifically logs off or is logged off (see later).



It is potentially possible for the credential lifetime to expire while the user is performing a re-authentication. This is most likely to happen if the inactivity timer expires when the credential lifetime has almost been reached. The user is sent a login page but the credential lifetime expires while it is being completed.

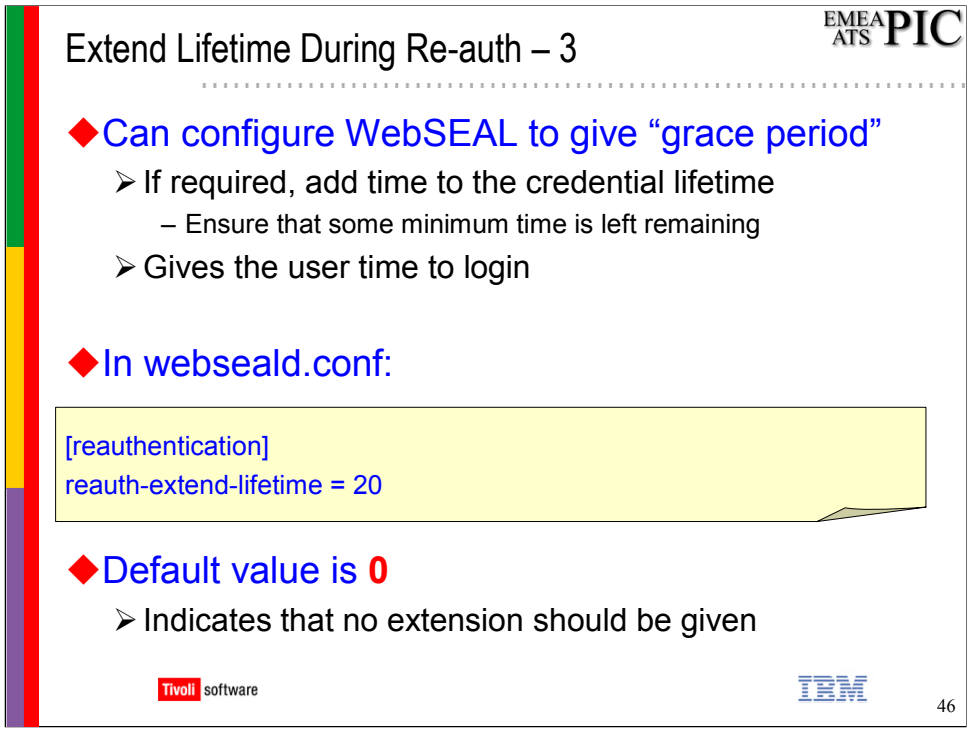
This could cause a problem because the users session cache entry is deleted when the lifetime expires and so when the login form is returned to WebSEAL there is no longer a session.

Not only is the user session lost but any user request data (which was stored in the session cache during authentication) is lost too.



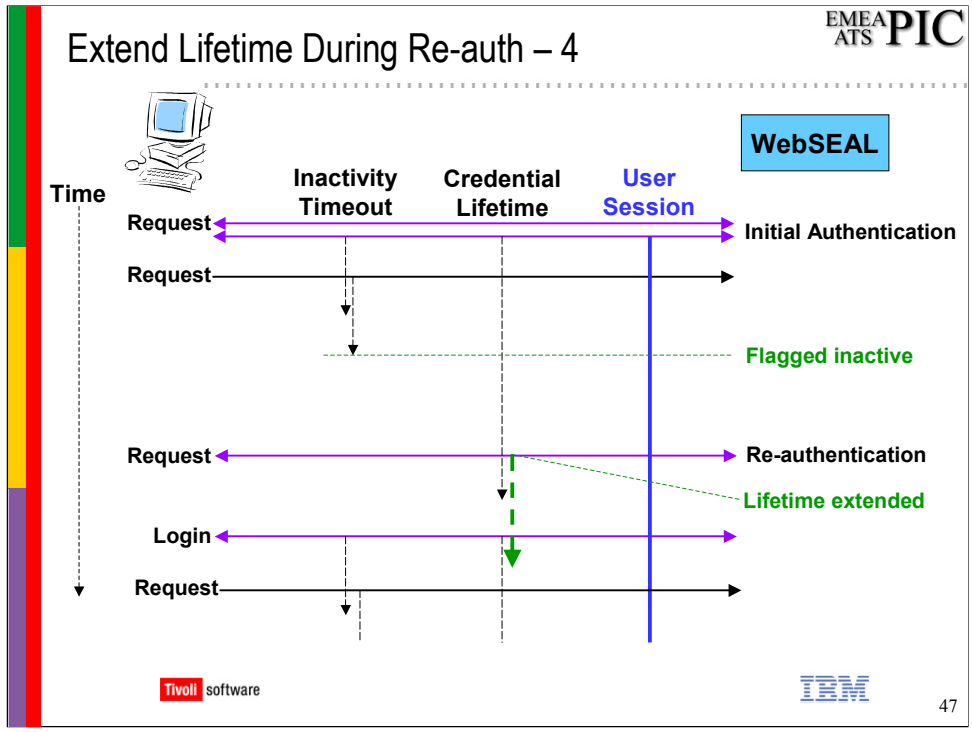
The diagram above demonstrates the problem described on the previous page.

The users session is deleted between the re-authentication being requested and the user responding.



The problem of the credential lifetime being reached during re-authentication can be countered by configuring a minimum remaining time that is acceptable when a re-authentication request is sent by WebSEAL. If the credential lifetime will be reached within this time then the credential lifetime is extended to give the minimum time specified.

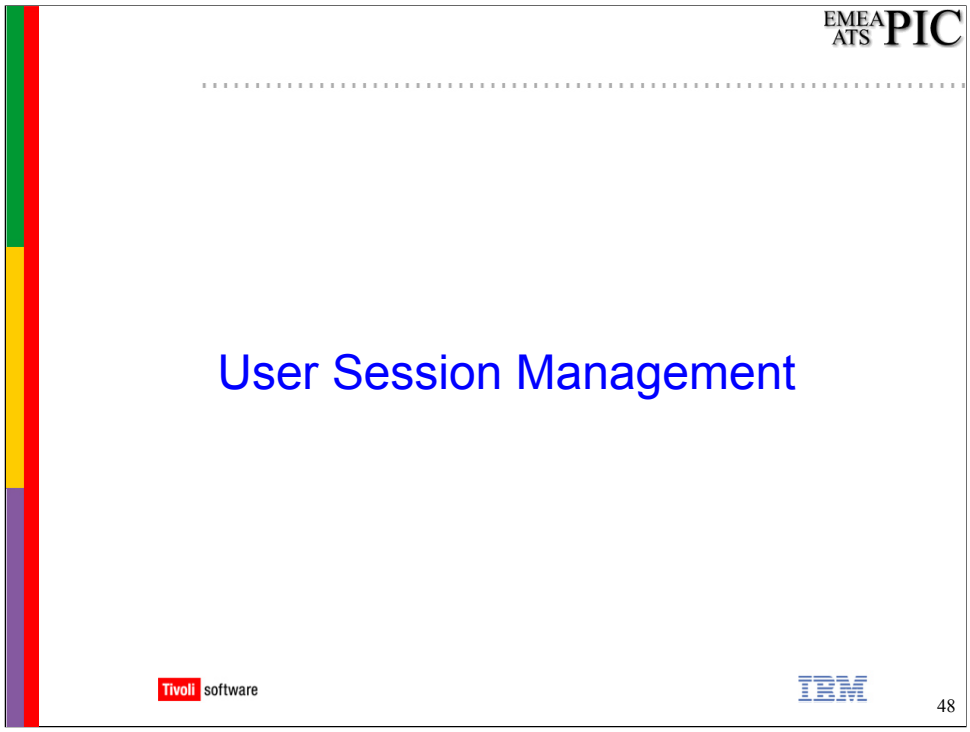
By default this grace period is set to 0 which means that no extension will be given during re-authentication. If a grace period is wanted then the *reauth-extend-lifetime* parameter in the *[reauthentication]* stanza should be modified with the time required.

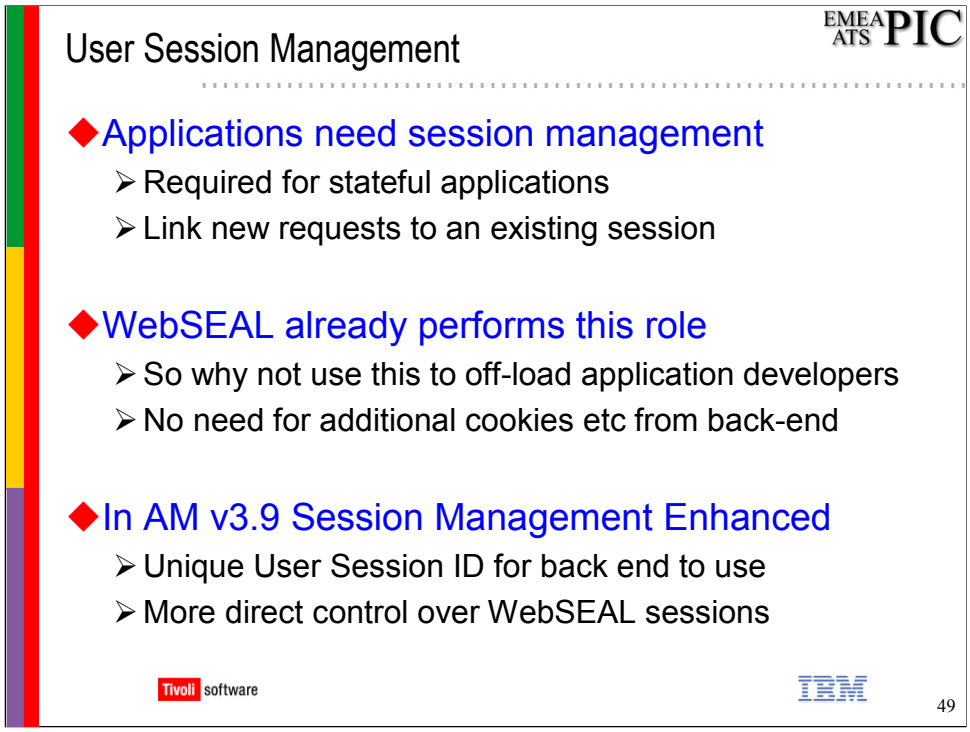


The diagram above shows how an extension to the credential lifetime can prevent the user session being deleted during re-authentication.

The extension should be configured to give a user that receives a re-authentication request adequate time to complete and submit it.

The *reauth-extend*-lifetime option is intended for use in conjunction with the *reauth-reset-lifetime=yes* option. This means that when a successful re-authentication is completed the credential lifetime will be extended again to give another full amount of session time.


EMEA ATS PIC
User Session Management
Tivoli software
IBM
48



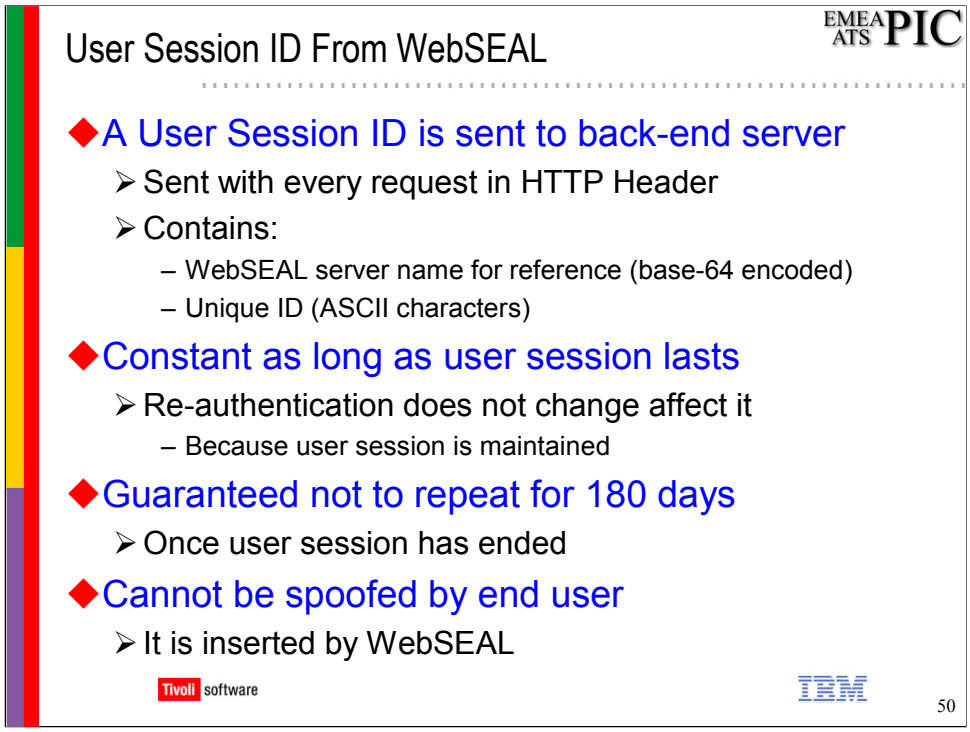
Any stateful WEB application needs to maintain a table of some kind that links HTTP requests from the same user to information about their session state.

WebSEAL is stateful and it can use a variety different information to link each new request with previous ones from the same user (e.g. SSL Session ID, BA Header, Session Cookies).

When WebSEAL is used to protect existing WEB applications there is usually a duplication of effort. WebSEAL and the WEB application both have to maintain the user session. Since WebSEAL is already doing this work it would be useful if the backend server could take advantage of this.

In AM v3.9 WebSEAL can be configured to create a “User Session ID” when a user session is created and then pass this to the back-end server with each request that the user makes. Rather than send a cookie to the user (or use BA Headers or URL re-writing) the back-end application can use the User Session ID to identify a user session.

In addition to providing this User Session ID for the application WebSEAL now also has the ability to drop a user session on the request of the back-end application (through the Admin API)



The User Session ID contains two parts:

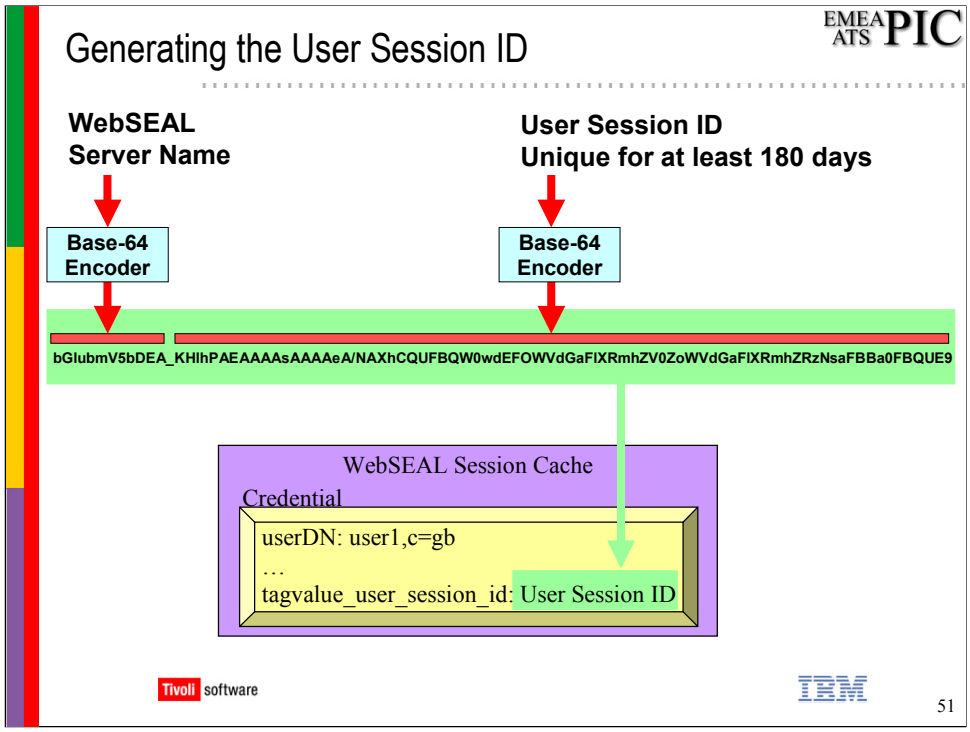
Server Name: The name of the WebSEAL server (base-64 encoded to avoid problems with invalid characters). This is used by the backend server when communicating via the Admin API to identify the WebSEAL server that owns the session.

Unique ID: A unique ID of ASCII characters that identifies the session. This ID is designed so that it will not be re-used for another session for at least 180 days.

The User Session ID is generated when WebSEAL creates a user credential and it is stored in the credential until the user session ends (and the credential is deleted). Using the new re-authentication features of WebSEAL in AM v3.9 the users session (and their User Session ID) can be maintained across re-authentications due to inactivity and forced re-authentications.

The Session ID is inserted into HTTP requests (as an HTTP Header) at WebSEAL – it is never sent to the end-user. This means that the Session ID cannot be spoofed by an end user as long as they are accessing the backend through WebSEAL.

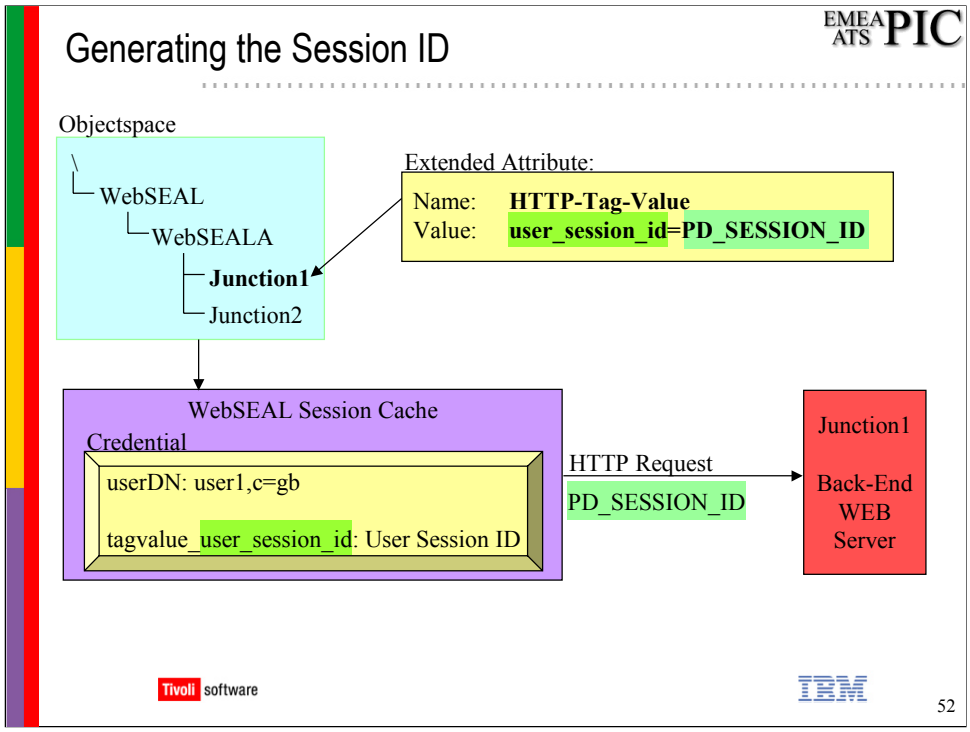
If the user attempts to insert an HTTP Header with the same name used for the User Session ID then WebSEAL will overwrite this as it processes the request.



The diagram above shows how a User Session ID looks. The first part is the base-64 encoded WebSEAL server name and the second part is a base-64 encoded unique session identifier. The two parts are separated by an _ (underscore) character.

This value is stored in the users credential using the attribute name **tagvalue_user_session_id**. This name is used so that the existing Tag-Value support can be used to insert the Session ID into HTTP requests to the back-end (see next page).

Note: If the backend application is a user of the AZN API the WebSEAL can send the entire user credential to the application and it can extract the User Session ID from there without the need to use an additional HTTP Header.

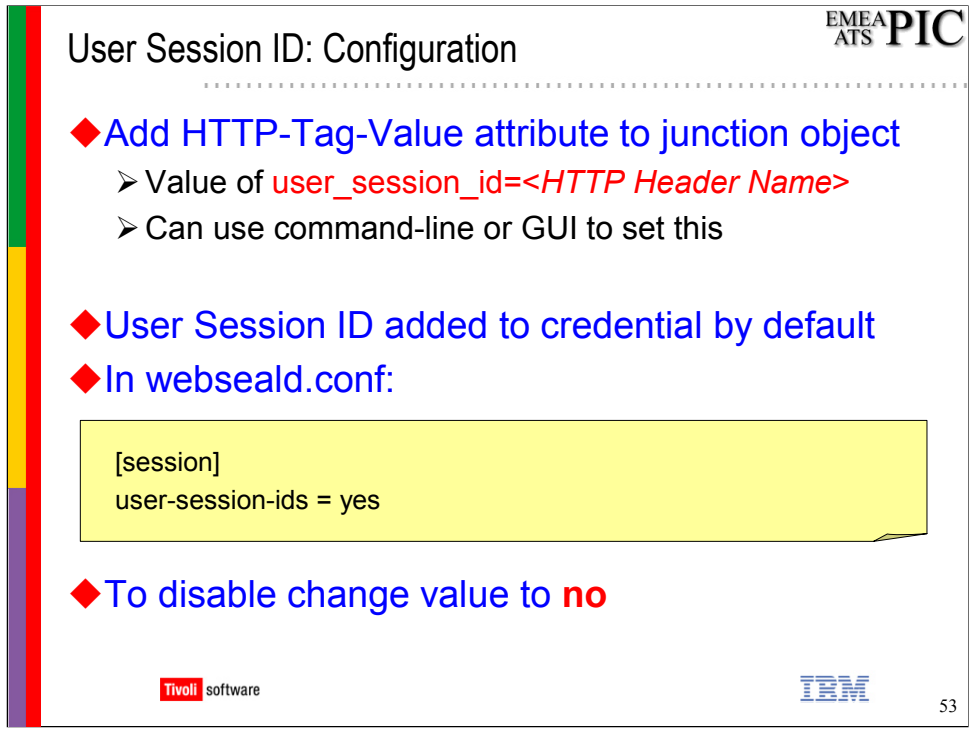


The existing WebSEAL Tag-Value support is used to extract the User Session ID from the credential and place it into the HTTP Headers that are sent to the back-end server.

The Tag-Value support is triggered by an extended attribute of the junction object in the AM objectspace. This attribute has a name of **HTTP-Tag-Value** and a value of ***<credential attribute>=<HTTP Header name>***.

When a request is made to a junctioned back-end server the junction object is checked for the extended attribute. If it is found then the user credential is searched for a custom attribute call **tagvalue_*<credential attribute>***. In the case of User Session ID the value of in the Credential attribute is **tagvalue_user_session_id**. The value of this Credential attribute (this is the User Session ID) is inserted into the HTTP Request using the ***<HTTP Header Name>*** given above. In the example above the HTTP header will be called **PD_SESSION_ID**.

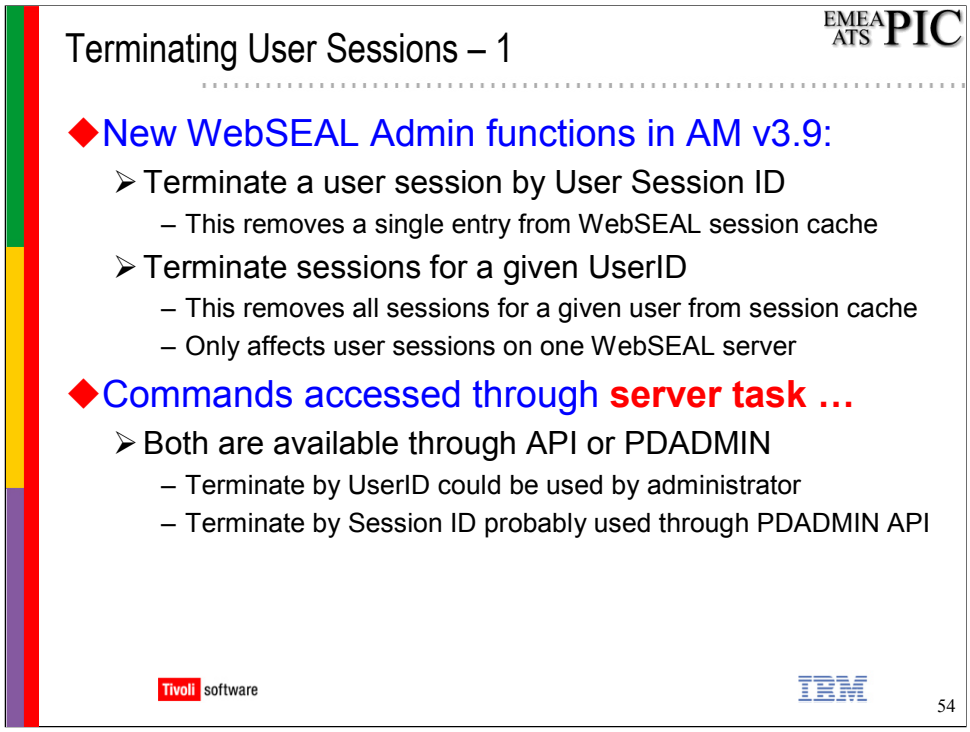
The back-end server can then acquire the User Session ID by requesting the value of the **PD_SESSION_ID** Header.



The default configuration of WebSEAL in AM v3.9 is to create a User Session ID for each session and put it into the credential. This means that the only configuration that is required is that described on the previous page to trigger the Tag-Value support. This can be done with PDADMIN or the WPM. In PDADMIN the command would be like:

pdadmin> **object mod /WebSEAL/myhost/junction1 set attr HTTP-Tag-Value user_session_id=PD_SESSION_ID**

If you want to prevent WebSEAL from generating the User Session ID then this can be done in webseald.conf. Changing the *user-session-ids* parameter to *no* in the *[session]* stanza will turn off the feature.

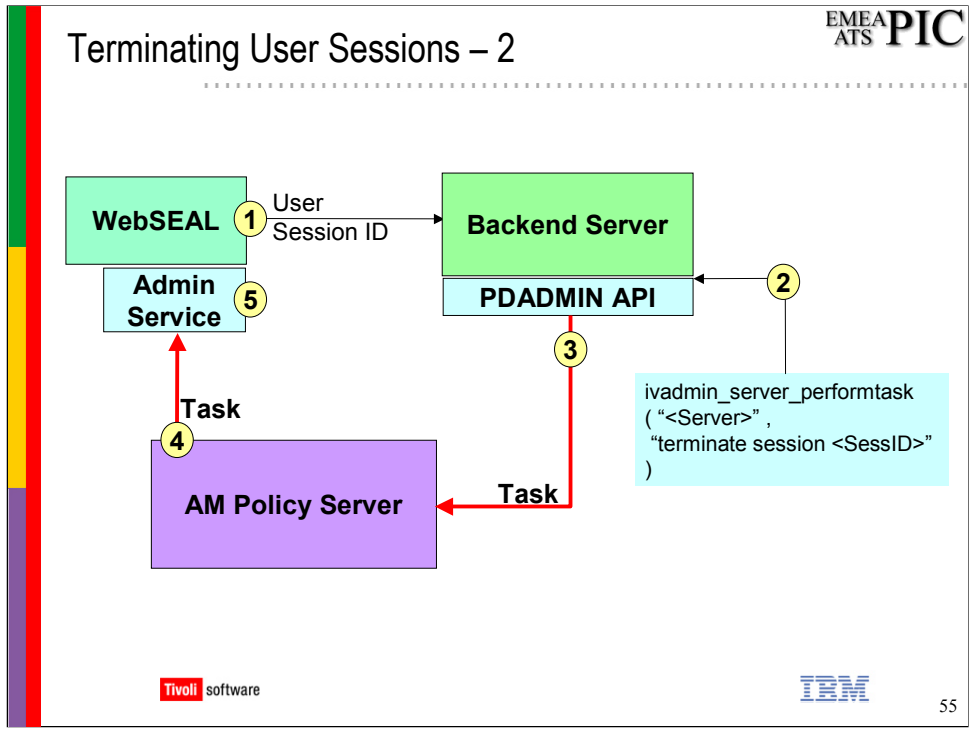


So far we have seen how the back-end server can acquire a User Session ID from WebSEAL which it can use for its own session maintenance. What is required now is for the back-end server to have some control over the session that is being managed by WebSEAL.

Two new commands have been implemented in the WebSEAL administration service that allow a user session to be terminated by giving the Unique ID portion of the User Session ID or the AM Username. These can be run from the PDADMIN command line (using server task) but they are intended for use by the back-end server through the PDADMIN API.

Terminating a session using the User Session ID causes WebSEAL to discard the single session that the User Session ID identifies. Other sessions from the same user can continue.

Terminating using the AM Username causes WebSEAL to discard ALL sessions that are owned by the Username given. This command may end many sessions if the user is logged in multiple times from different locations or different browsers.



The diagram above shows how the User Session ID is used to allow the Backend server to terminate sessions at WebSEAL.

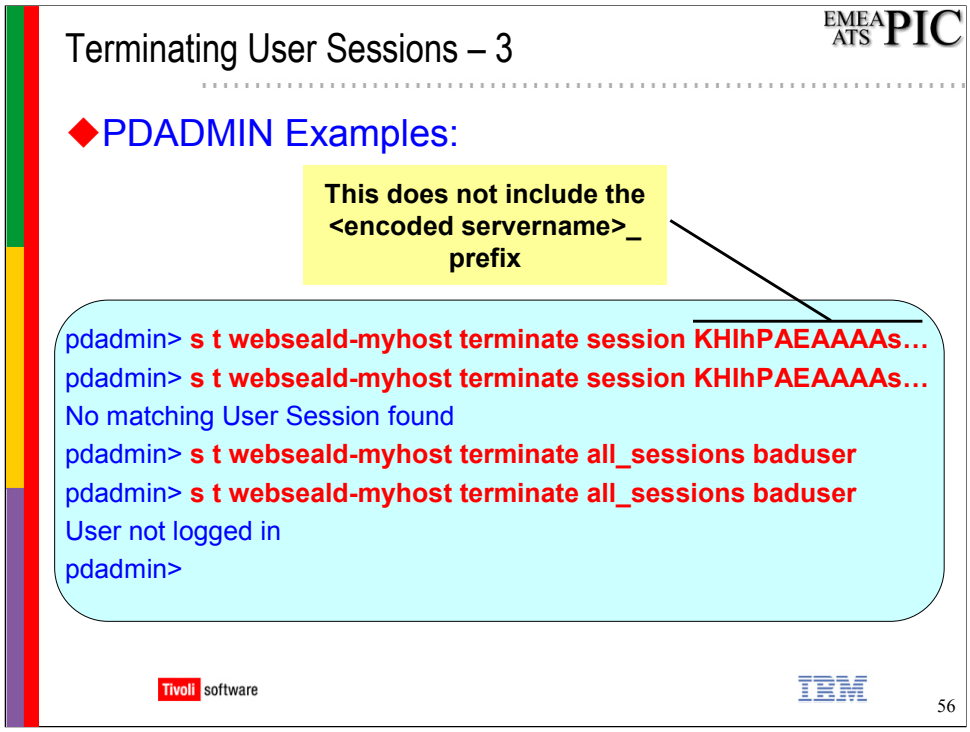
*1 – WebSEAL sends the session ID with every request (once Tag-Value is configured). Backend server could also extract from credential if using AZNAPI.

*2 – Backend server makes a call via the PDADMIN API specifying the WeBSEAL server name (either fixed or could be decoded from base-64 in Session ID) and the second, unique part, or the Session ID (left base-64 encoded).

*3 This request is passed to the AM Policy Server

*4 Management Server passes the request to the administrative service running on the WebSEAL server

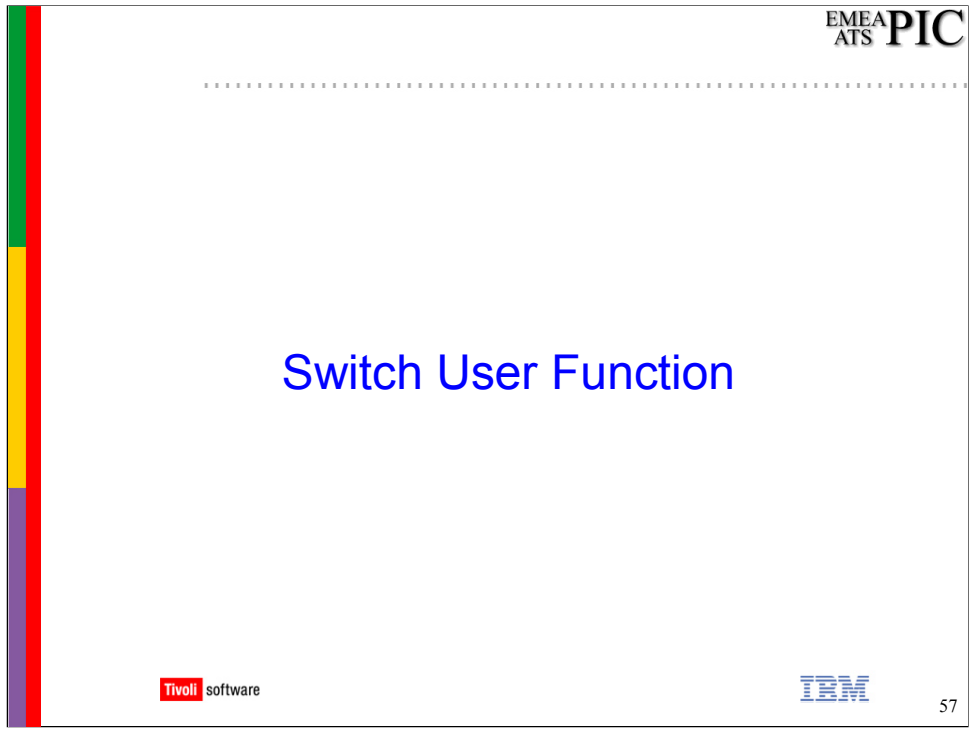
*5 The WebSEAL Admin Service kills the session on WebSEAL.

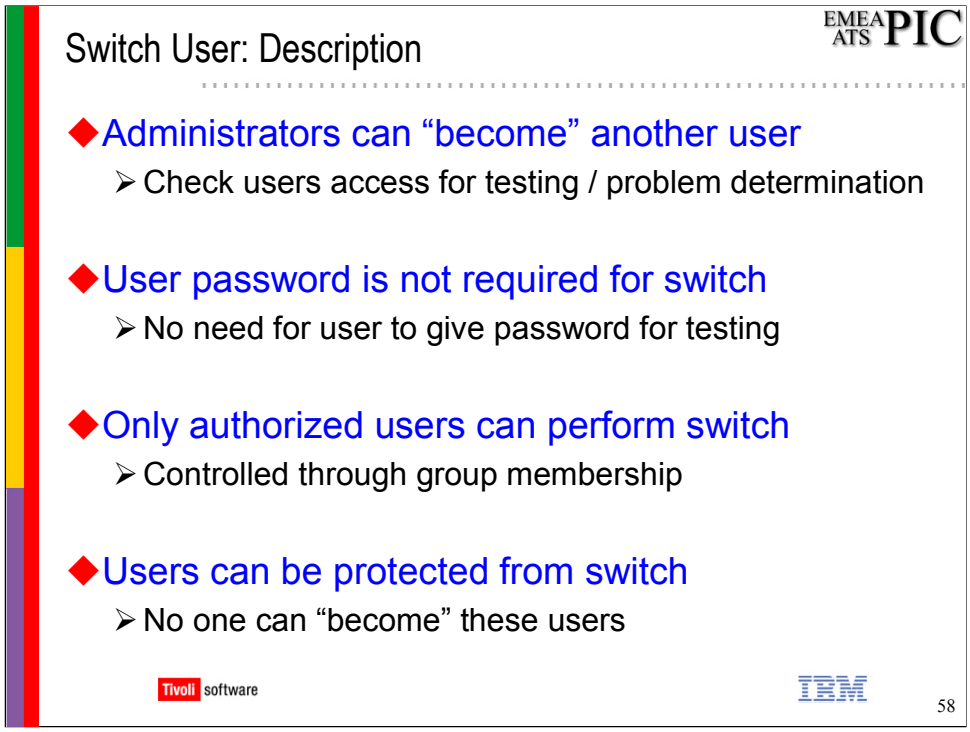


The examples above show how the server tasks can be used from the PDADMIN command line. **This is not the intended way for the terminate session command to be used** (since there is normally no way for the administrator to get the Session ID of an individual session) but it might be useful to be able to terminate all of the sessions for a given user if there is a situation where they need to be immediately logged off from WebSEAL.

The commands only affect the single WebSEAL server specified in the command. There is currently no single command that will log a user off from all WebSEAL servers. If this functionality is required then a simple script that reads the servers from a "server list" command and then calls "terminate all_sessions" on each one would do this.

There is nothing to stop a user who is logged off in this way from starting a new session. If this is to be prevented then their account-valid flag should be disabled before terminating their sessions. With this done they will not be able to log in again.


EMEA ATS PIC
Switch User Function
Tivoli software
IBM
57

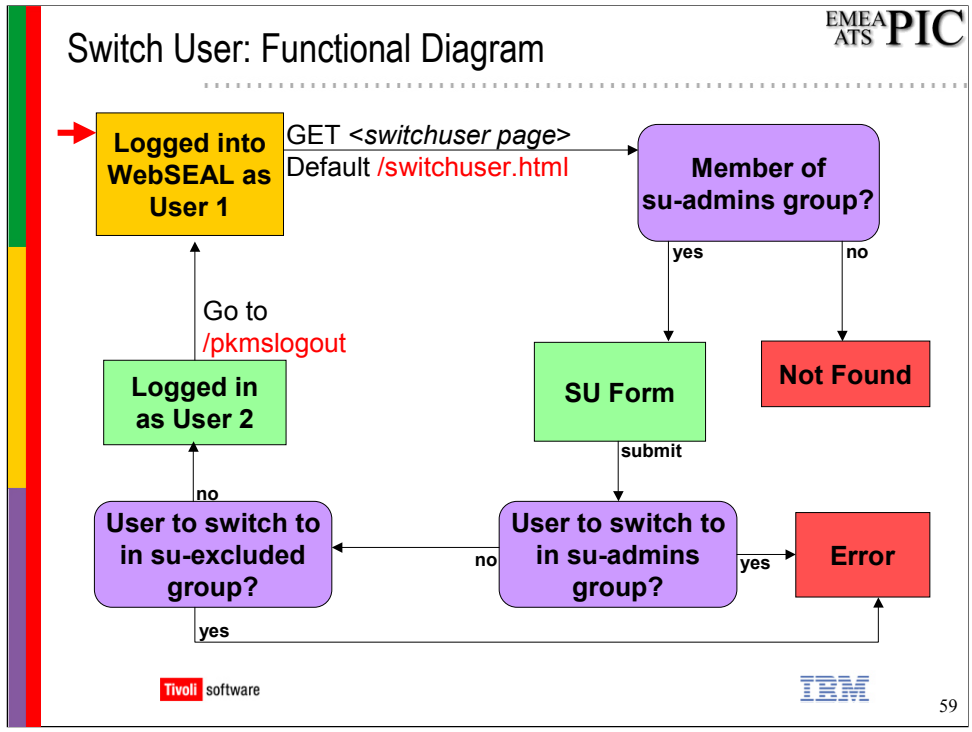


The “Switch User” function is rather like the SU command that is implemented in UNIX. It allows an administrative user to take on the identity of another user (“Become” another user) **without needing their authentication information**.

In a WebSEAL environment this could be used to see what pages a user will see when using a particular application. This could be used for application integration testing, permission testing or problem recreation and debugging.

This is a very powerful function which could easily be misused so users must be specifically authorized to use this feature. In addition it is possible to protect users from this feature by specifying that they cannot be the subject of switch user – i.e. no other user can “become” that user even if they are authorized to use the switch user function.

Note: It is not permitted to “become” another user that is authorized to use the switch user function. This removes the problem of chained switches – where a user switched from one UserID to another multiple times.



The flowchart above shows the logic of the switch user function. Initially we start with an authenticated user, **User 1**. **User 1** attempts to initiate the switch user function by connecting to the configured switch user page.

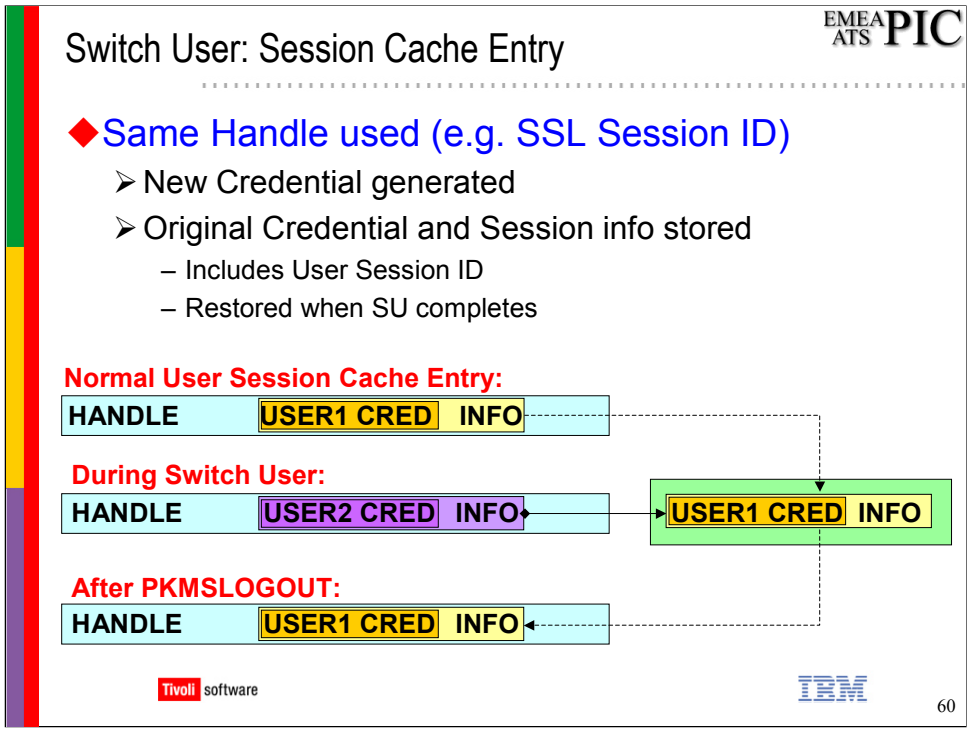
WebSEAL recognises this request and checks if the user is a member of the special group *su-admins*. Only members of this group are allowed to use the switch user function. If the user is not a member of the group they will get a standard "Not Found" page returned.

If **User 1** is a member of the *su-admins* group then the Switch User page is returned. This page contains a form where the user can fill in the username they wish to switch to and some other information (see later). **User 1** enters **User 2** into the form and submits it.

Two checks are done before the switch is authorized. First, WebSEAL checks if **User 2** is a member of the *su-admins* group. A user cannot "become" another user that is allowed to use switch user. Second, WebSEAL checks if **User 2** is a member of the *su-excluded* group. No users are allowed to "become" a member of the *su-excluded* group. A error is returned if either of these checks fails.

If the checks are OK then **User1** becomes **User2**. All subsequent requests are made as though they were made by **User2**.

This continues until /pkmslogout is called. At this time **User 2** is logged out and the user becomes **User 1** again.

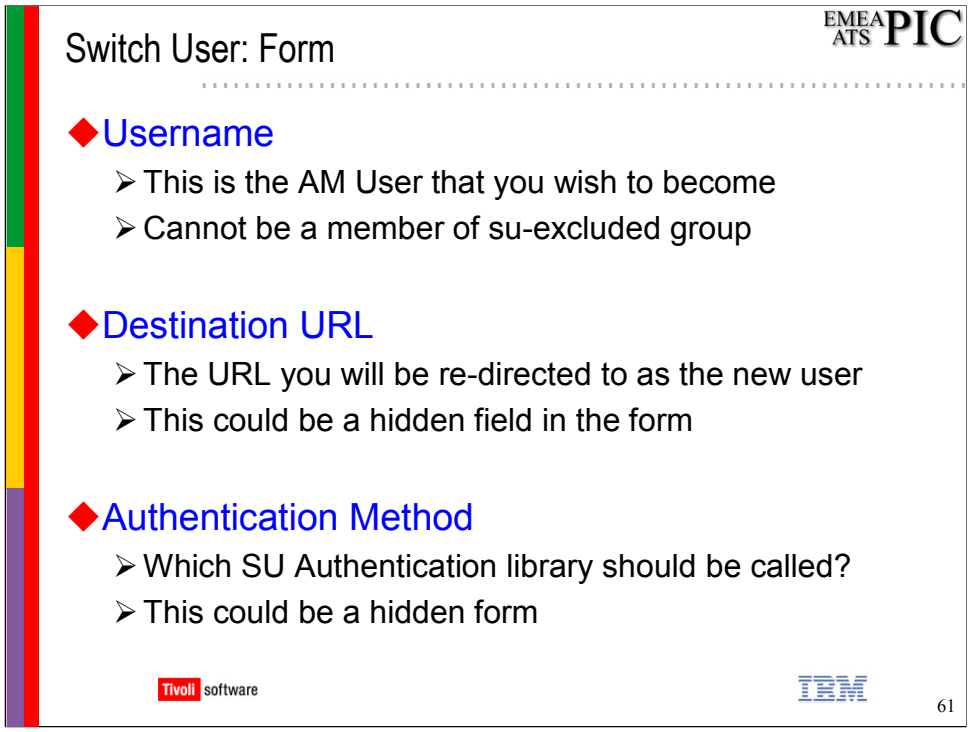


The diagrams above show how the users session entry changes during switch user operation.

When a user session starts an entry for the session is created in the session cache. A session cache entry in indexed by a handle. This is information that identifies the session, e.g. SSL ID, BA Header or Session Cookie. Contained in the session entry is the users credential and information about their session.

When the switch user function is invoked the original user credential and information is moved into another cache. It is replaced in the session entry with the credential and new information for the user that is being switched to. Notice that the session handle does not change. WebSEAL still uses the same information to locate the session. Any requests made on that session will use the new credential for authorization.

When the user goes to the /pkmslogout page (to indicate they wish to end the switch user session) the credential and information in the session entry is discarded and replaced with what was saved away earlier. Notice that the session handle is still unchanged. Any requests made on the session will use the original credential again for authorization.

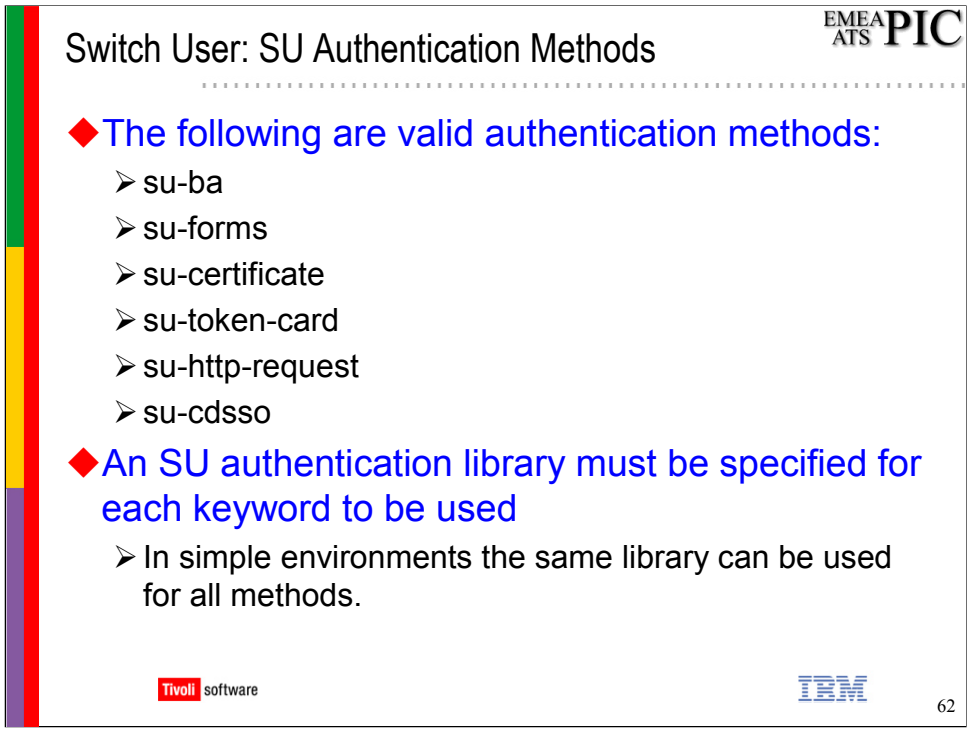


The switch user form that an authorized user requests the switchuser page contains 3 fields. All three fields are required in order to process the switchuser request however the switch user form could be customised to use hidden inputs (or have a pull down menu of choices etc) for some of the fields.

Username: This input is where the username you want to switch to is given. This can be any user that is NOT a member of *su-admins* or *su-exclueded*.

Destination URL: This is the URL that you will be re-directed to once authenticated as the new user. In most cases the switchuser form would be customised so that this is a hidden input containing an appropriate home page – or a page that confirms that switch user was successful.

Authentication Method: When switching to another user no authentication is required but WebSEAL still needs to call an authentication mechanism that will provide the appropriate information for building the users credential. The mechanism to be called is specified here (by keyword). This could also be a hidden form or a set of radio buttons (etc) if required. See next page for details on Authentication Methods.

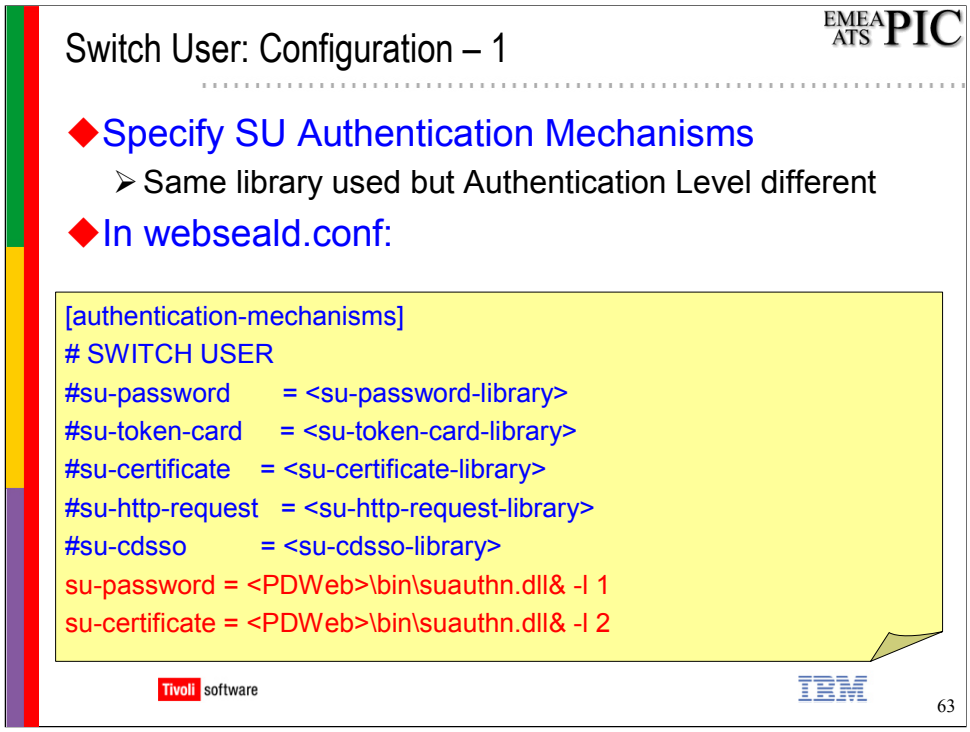


WebSEAL defines a new set of Authentication Method keywords for use with the Switch User function. This allows a different library to be called for each different authentication type. These "SU" libraries are different from standard CDAS libraries because they do not require any authentication information to be passed in.

It is important to have the ability to call different su libraries in an environment where custom CDASs are normally used to specify information to be added to the credential. This information needs to be added to the credential generated for the user being switched to.

Access Manager v3.9 ships a single SU library that can be used for all authentication methods in an out-of-the-box environment. It does not add any information to the credential – it simply returns the User ID passed in and sets the authentication level based on a parameter specified when it is called (see next page).

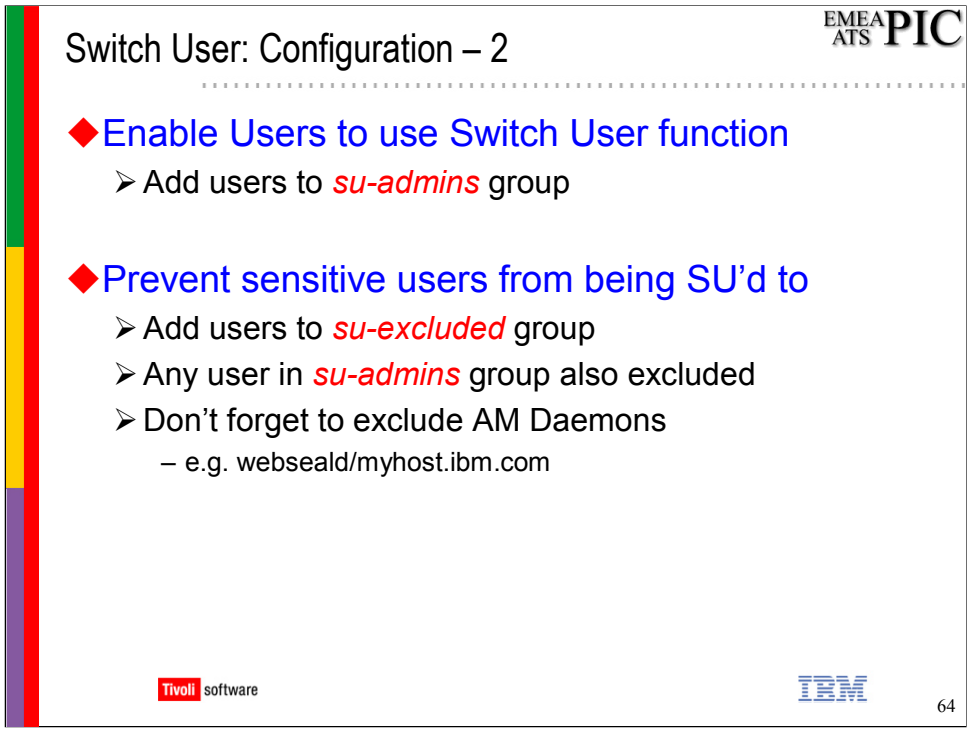
These authentication method keywords are also the keywords that are recognised in the switchuser form to specify which authentication method should be used.



The slide above shows how, in an out-of-the-box environment, the single SU library shipped with WebSEAL in AM v3.9 can be used to handle all authentication methods. For each method that needs to be used specify the su-xxx keyword and then specify the shipped SU library. The *-l* parameter will be passed to the library and is used to return the authentication level that should be reported while working in the SU session.

The reason why the same library can be used for all the different authentication mechanisms is because no authentication information is passed in. This is the only thing that makes the out-of-the-box CDAS libraries different – they all return similar information.

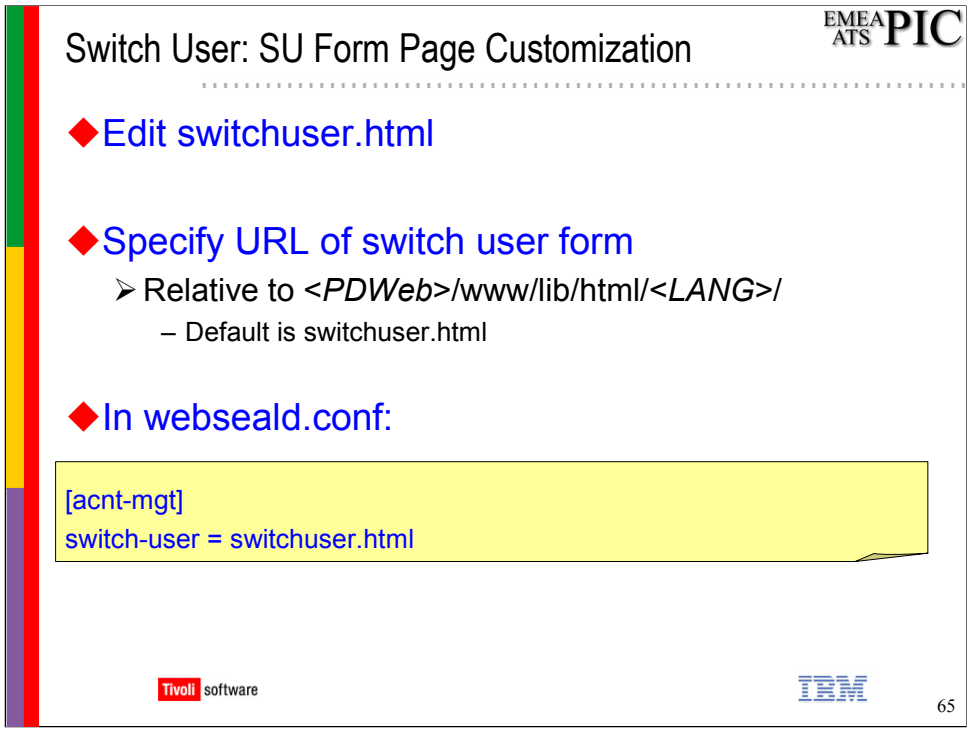
In an environment where custom CDAS libraries are used to return extra information to WebSEAL it may be necessary to write matching SU libraries if the switch user function is to be used with those custom methods.



In order to enable a user to use the switch user function simply add them to the *su-admins* group. This group is created in Access Manager when WebSEAL is installed.

In order to protect user accounts from being switched to add them to the *su-excluded* group. This group is also created in Access Manager when WebSEAL is installed.

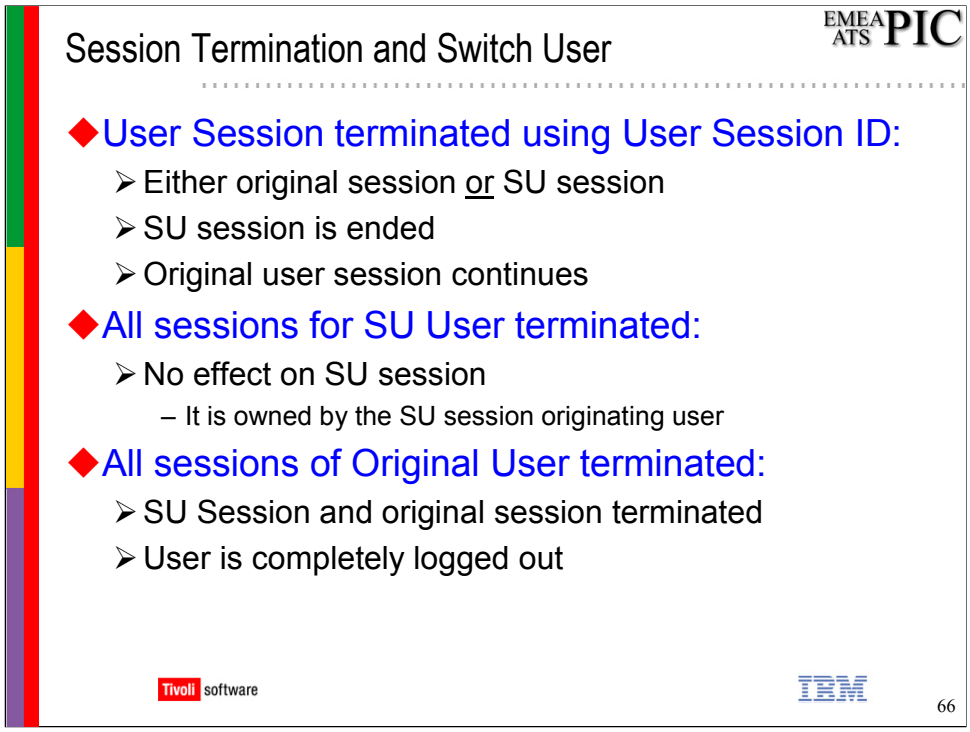
By default no users are made a member of either of these groups. It is recommended that sec_master and all AM Daemons are added to this group so that they cannot be used to get access to Web resources via the switch user function.



Although the switchuser form is requested as /<*su-form-URL*> it does not actually exist in the WebSEAL webspace. WebSEAL recognises requests for the page and loads the page relative to the <*PDWeb*>/www/lib/html/<*lang*> directory – which is where WebSEAL stores all "special" pages. <*PDWeb*> is the PDWeb installation directory and <*lang*> is the language. On US English systems this directory is called 'C'.

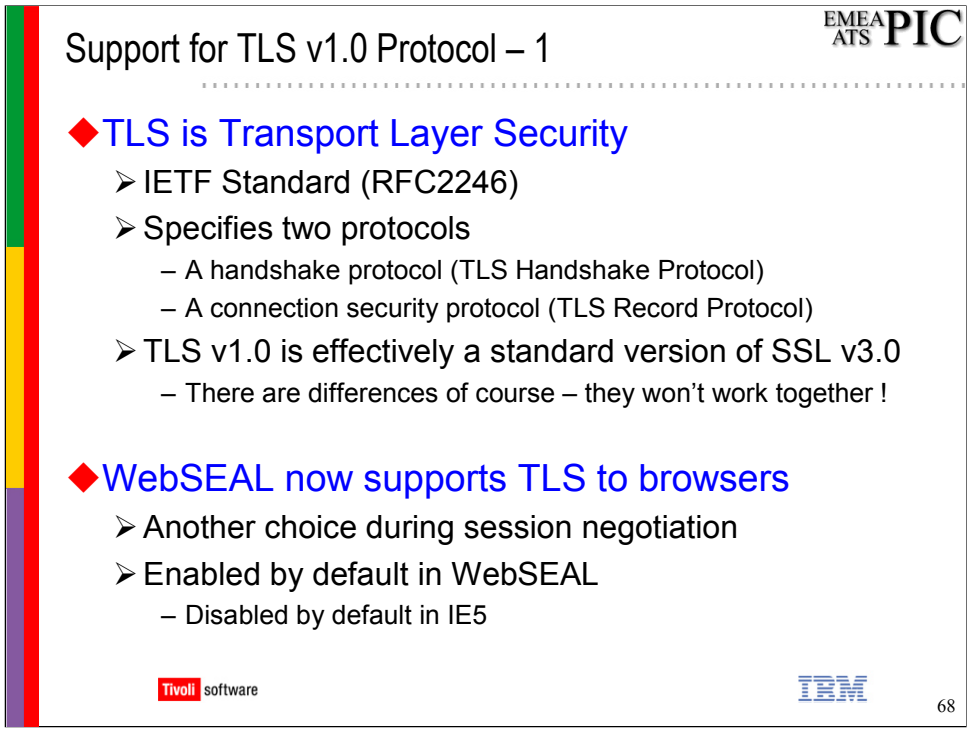
It is possible (and probably desirable) to customise the page that contains the switch user form. The most likely customisations would be to change the Destination URL to a hidden field – to direct the user to an appropriate home page – and change the Authentication Mechanism to a pull down menu or a set of radio buttons.

By default the switch user page is named *switchuser.html* and a template already exists in the directory shown above. If you wish to change the name of this file (which will change the URL that users use to activate the function **and** the name of the file that WebSEAL will look for) then this is done in webseald.conf as shown above.



The slide above describes the behaviour of WebSEAL when it is instructed to terminate sessions (or users) that are involved in a Switch User operation – either as the originator as the user being switched to.


EMEA ATS PIC
Other Enhancements
Tivoli software
IBM
67

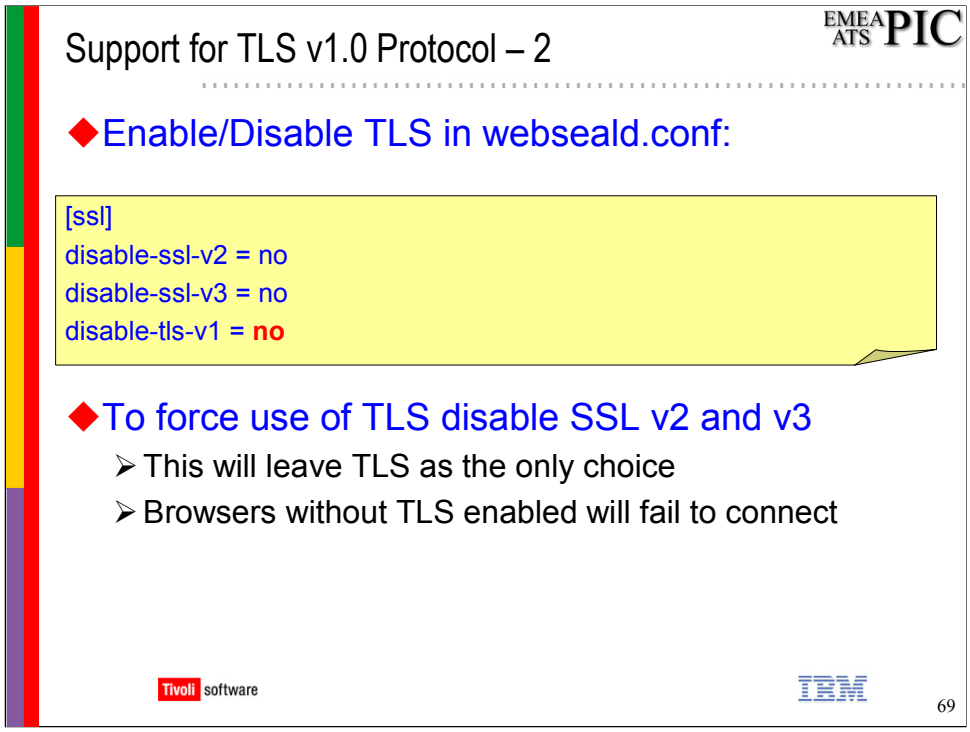


WebSEAL in AM v3.9 can support the Transport Layer Security (TLS) protocol for secure communication to browsers. TLS v1.0 is an alternative to SSL v2 or SSL v3 and is fairly similar to SSL v3. It is an IETF standard (RFC 2246) whereas SSL is not an official standard.

Since the browser initiates SSL sessions to WebSEAL it is up to the browser which protocol will be used if more than one is available. The default in WebSEAL is to have SSL v2, SSL v3 and TLS v1 all enabled. This means that, out-of-the-box, the only difference that will be seen in AM v3.9 is that if a browser connects to WebSEAL that only supports TLS v1.0 it will now be able to start an HTTPS session where it would have failed before.

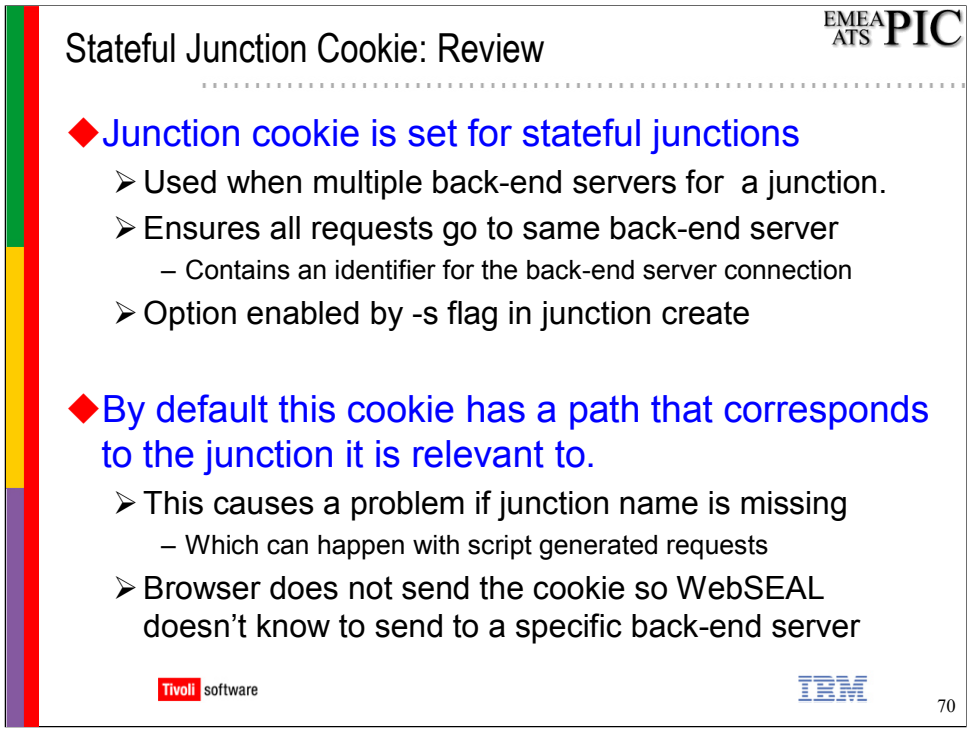
Having WebSEAL capable of using TLS v1.0 makes no difference if the browser prefers to use SSL. In this case an SSL session will be established unless WebSEAL specifically disables SSL.

Note: Internet Explorer 5.x has TLS v1.0 **disabled** by default. This means it will fail to connect to WebSEAL if WebSEAL only has TLS enabled.



WebSEAL can be configured to disable it's HTTPS security protocols on a protocol by protocol basis. This is configured in the *[ssl]* stanza of webseald.conf.

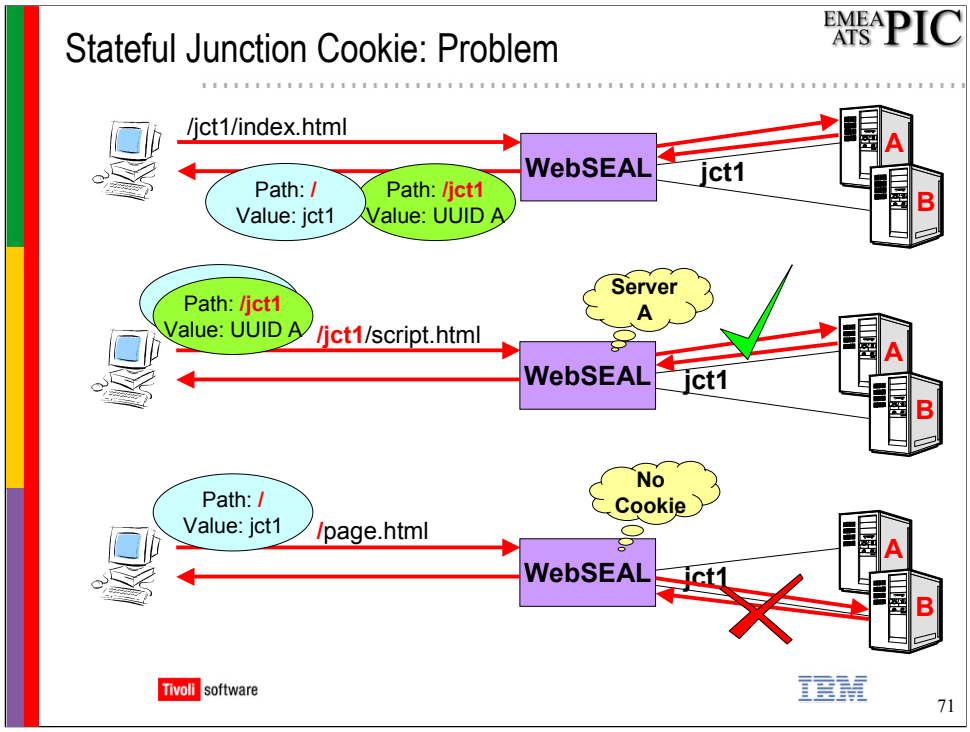
By default all the disable options are set to *no* which means all protocols are available. So force browsers to use a particular protocol simply disable all of the other choices. Browsers that do not support that protocol will fail to connect.



WebSEAL has long had an option of specifying that a junction connects to a stateful application. This is used when there is more than one back-end server **on the same junction** and these back-end servers maintain local state information about a users connection. This means that once a user has been directed to a particular back-end server all subsequent requests need to go to that same server. This is the same as a "sticky" connection in a load balancer.

This function is enabled by specifying the *–s* flag on the junction create command and it means that WebSEAL will set a cookie on the browser when the user first accesses the junction that indicates which back-end server is being used. This stateful-junction-cookie is set with a path that includes the junction name so that the cookie is only sent when requests are made to that particular junction.

This all works fine as long as the junction name is always included in each request. Unfortunately there are times, when dealing with scripts that include server relative links, when the browser may make a request that does **not** include the junction name. WebSEAL is able to handle this (as long as the *–j* flag has been used so that a script-support-cookie is set) but the browser does not send the stateful-junction-cookie with the request because the request did not include the junction name. In this case WebSEAL may send the request to the wrong back-end server. (See next page).

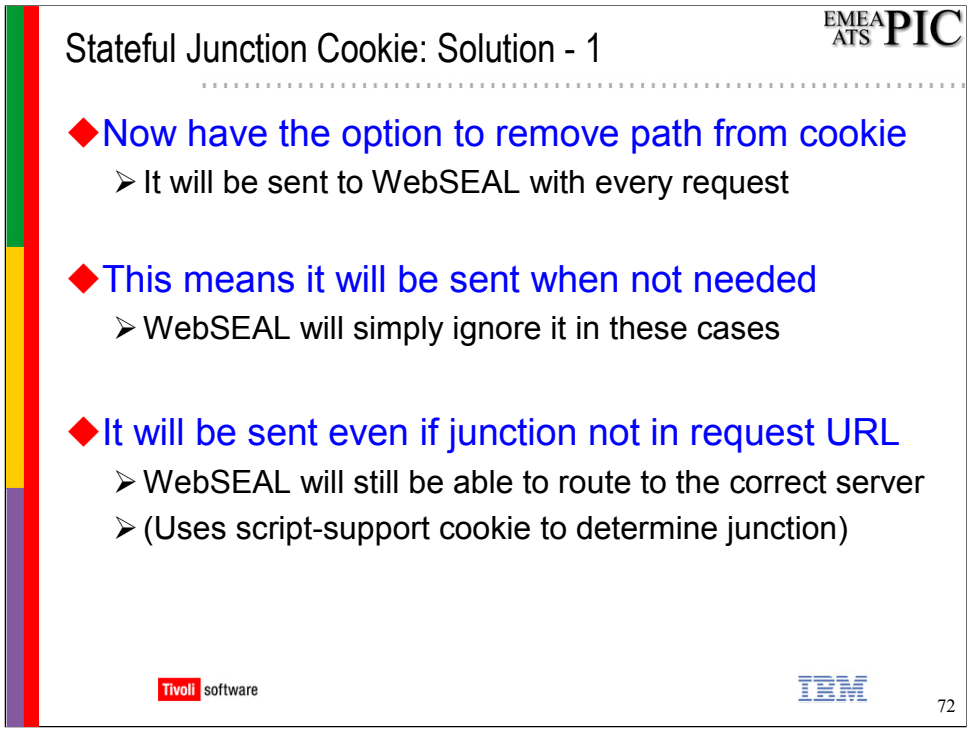


The diagram above shows how the failure situation can occur.

In the first request the user accesses server A on junction, jct1. When WebSEAL sends the response to the user, two cookies are set. The first is a script-support-cookie that contains the junction name and the second is the stateful- junction-cookie that contains a pointer to server A (It is a UUID associated with server A).

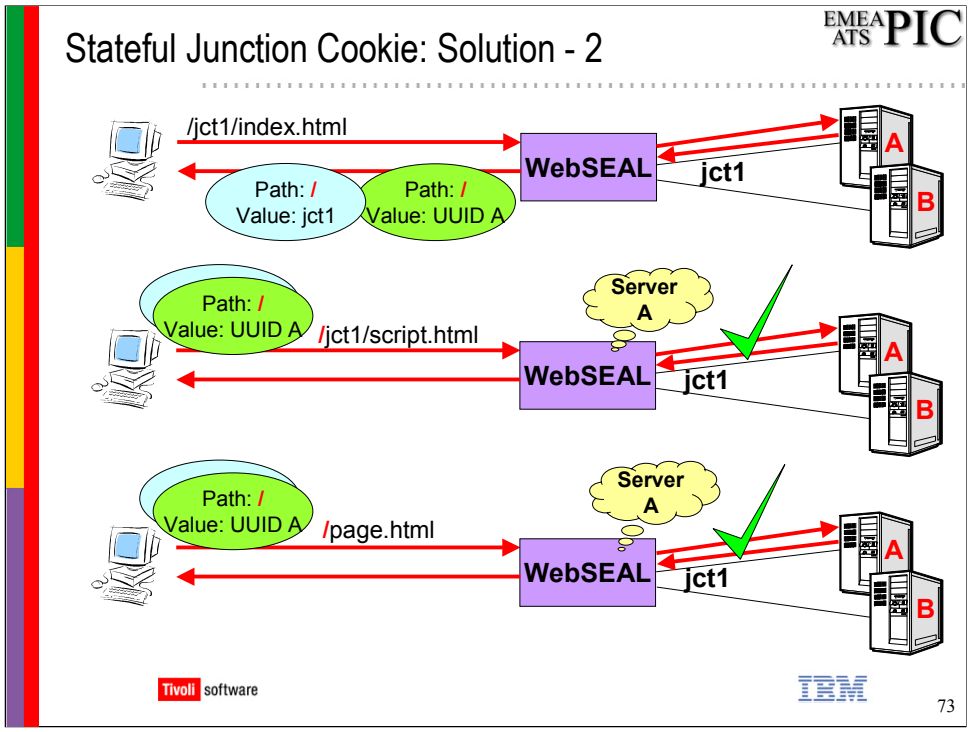
When the browser sends a normal request that includes the junction name both cookies are sent along because the path matches them both. WebSEAL sees the stateful-junction-cookie and knows that the request must be directed to server A.

If the browser is directed (via script) to a link that does not include the junction name, only the script-support-cookie is sent to WebSEAL. The request does not start with **/jct** so the stateful-junction-cookie is not sent. The script-support-cookie tells WebSEAL that the request needs to be sent to jct1 (so the request can still be directed correctly) but since the stateful-junction-cookie is not present the request may be sent to the wrong server on jct1. If this happens then the stateful application will fail because only server A has the application state for the user.



The solution to the problem of the stateful-junction-cookie not being sent is to remove the path from the cookie so that it is sent with all request to the WebSEAL server just like the script-support-cookie. This is now an option in AM v3.9.

Turning on this option means that the stateful-junction-cookie will be sent with all requests to the WebSEAL server – even when it is not required. This does not cause any real problems since WebSEAL will simply ignore it.



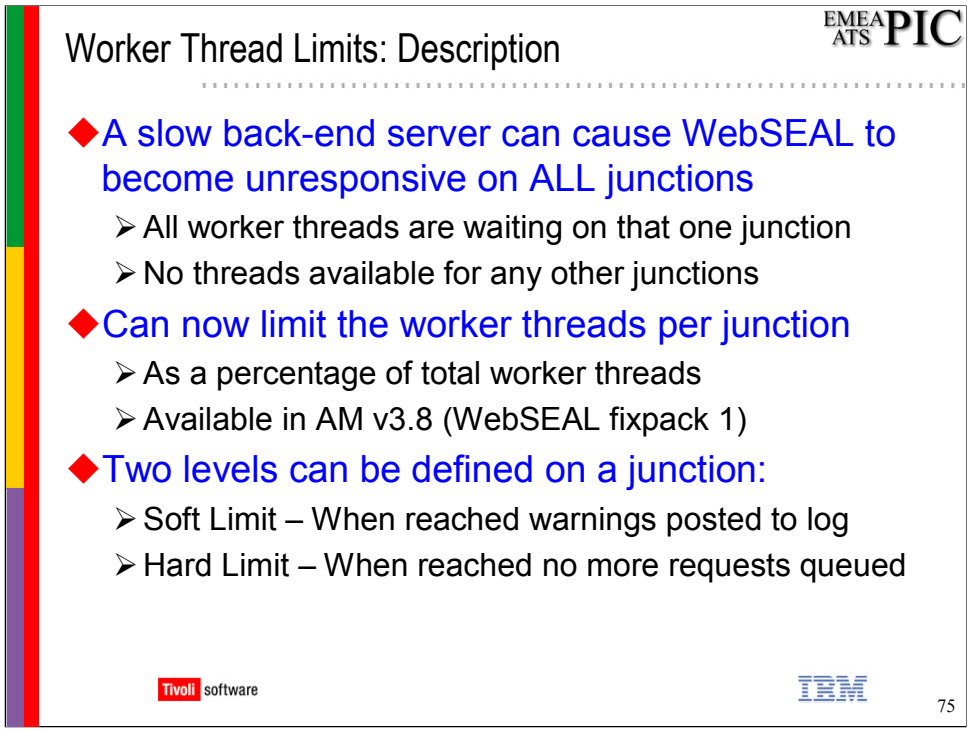
The diagram above shows how the failure situation is now avoided.

Everything is the same as before except that now the path of the stateful-junction-cookie is / rather than /jct.

When the browser is directed (via script) to a link that does not include the junction name, both the script-support-cookie and stateful-junction-cookie are sent to WebSEAL. The script-support-cookie tells WebSEAL that the request needs to be sent to jct1 and the stateful-junction-cookie tells WebSEAL that it should be directed to server A. The problem is avoided.

By default this new support is disabled – this means that the behaviour of WebSEAL remains unchanged from previous versions unless the configuration is specifically changed.

To stop WebSEAL from setting the path in stateful-junction-cookies set the *disable-stateful-cookie-path* to *yes* in the *[server]* stanza.



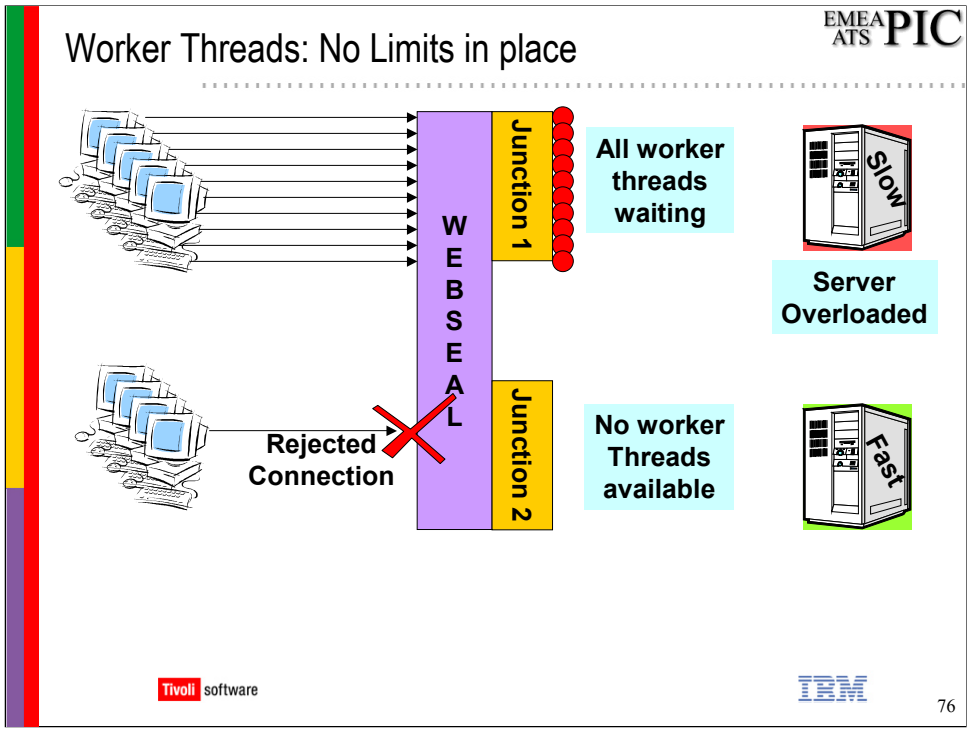
In an WebSEAL environment that contains a number of junctions with backend applications of different speeds it is possible for an application that takes a long time to respond to requests to use all of the WebSEAL worker threads and stop all other junctions from functioning. This is because all of the worker threads end up waiting on the slow junction and there are none free to handle requests from the faster junctions.

In WebSEAL v3.8 fixpack 1 and now in WebSEAL v3.9 it is possible to limit the percentage of the worker threads that can be assigned to a single junction. This can be used to ensure that worker threads are fairly distributed. Limits can be set globally and over-ridden on a per junction basis.

Two levels can be set on each junction:

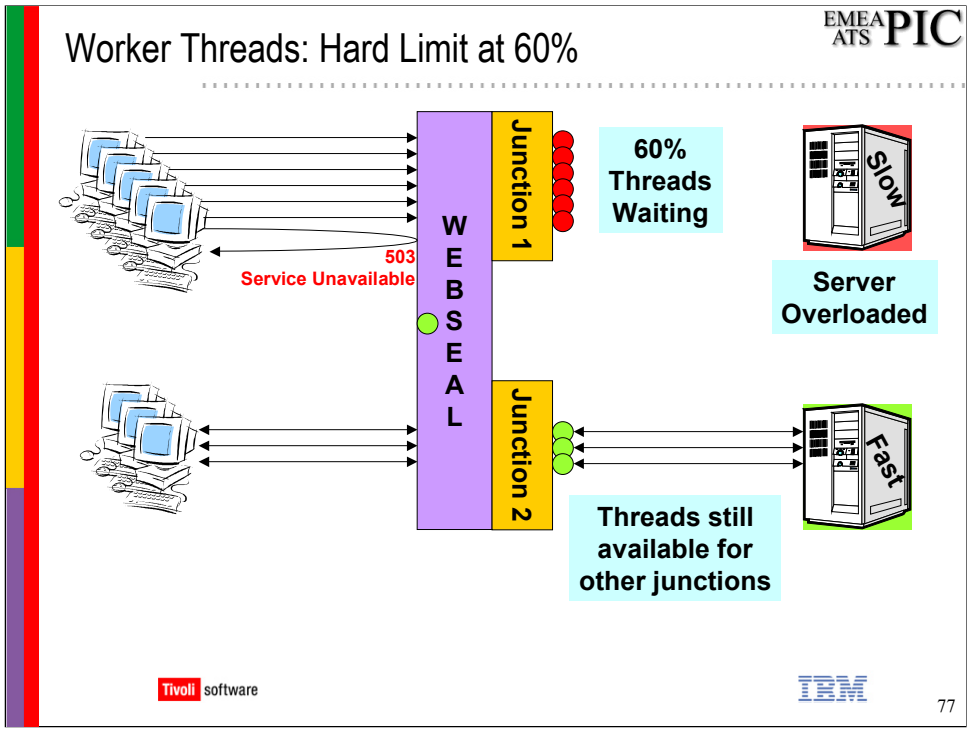
Soft limit – when this limit is reached a WARNING message is logged by WebSEAL.

Hard Limit – When this limit is reached subsequent requests for resources on that junction will be refused. An ERROR message is logged by WebSEAL and an error is returned to the client.



The diagram above shows the failure situation. The WebSEAL server has two junctions. One of these connects to a back-end server that takes much longer to service requests than the other.

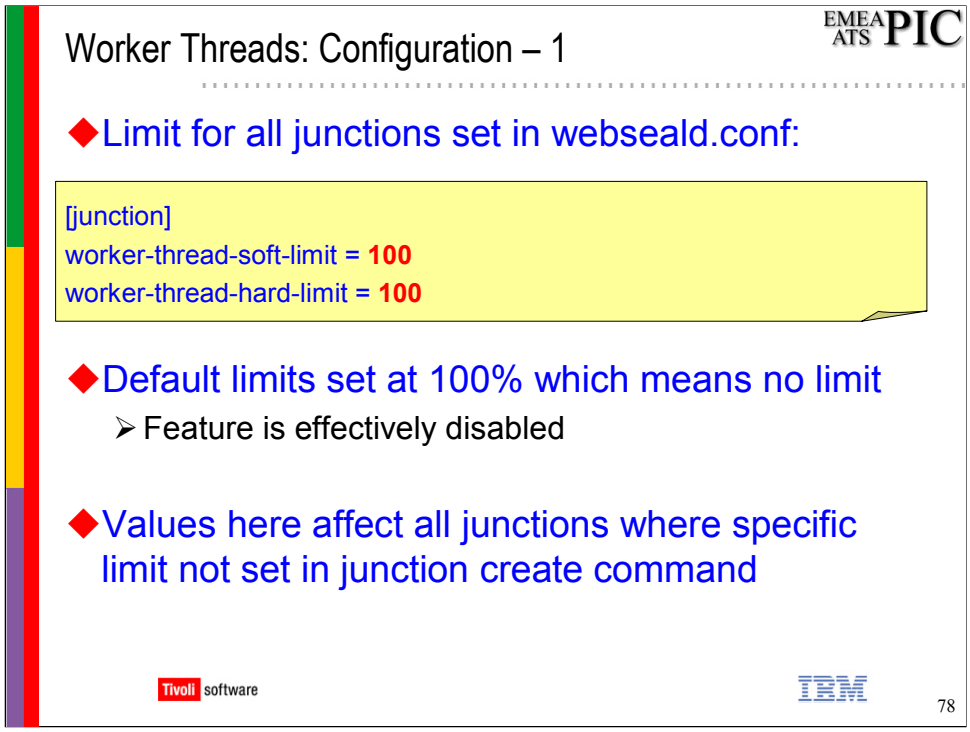
Each time a request comes in for a resource on the slow junction a worker thread is assigned which sends the request and then has to wait for the response. Requests to the fast junction are dealt with quickly and the worker thread is free again. If there are a large number of requests to WebSEAL eventually all of the threads may be waiting on the slow junction. This means that further requests to WebSEAL – even those for resources on the fast junction – are refused.



This diagram shows how a hard-limit set at 60% can prevent the failure situation occurring.

Under the same conditions as in the previous example, the number of threads waiting on the slow junction rises as more and more requests are outstanding. The difference is that when the number of threads waiting reaches 60% of the total threads available further requests for resources on that junction are refused. The client receives a 503 – Service Unavailable response. This is a valid response because that junction already has more requests than it can handle. Even if there are free worker threads they will not be assigned to that junction.

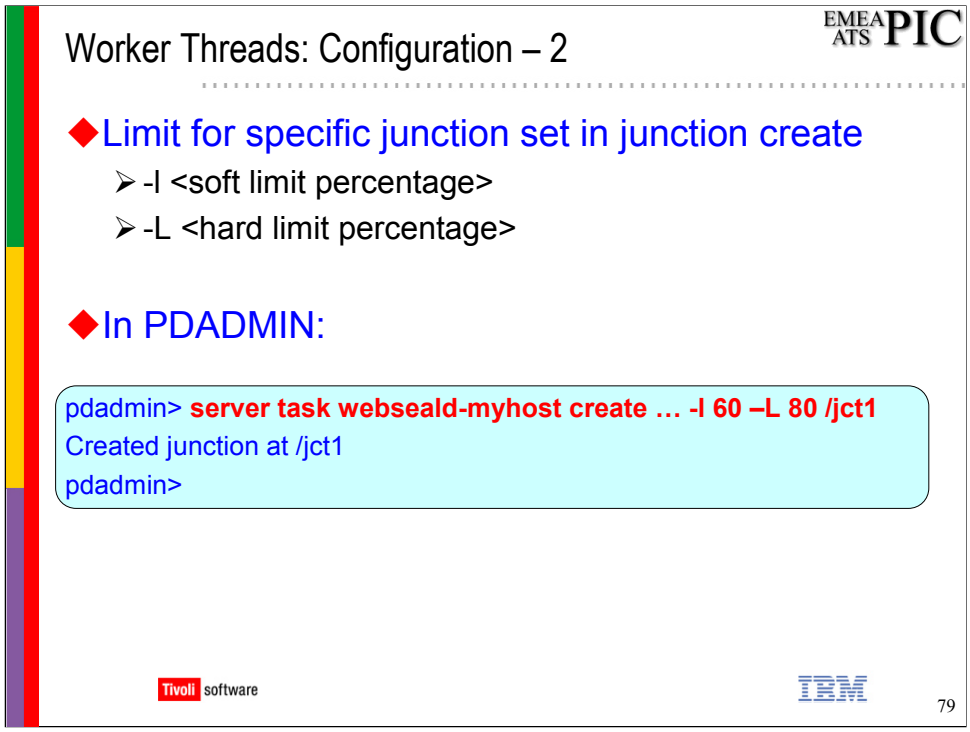
Since, at most, 60% of the worker threads can be assigned to the slow junction there will always be at least 40% of the threads available for the other junction. Requests for resources on that junction can continue to be serviced normally.



A global configuration for all junctions can be set in the *[junction]* stanza of webseald.conf. This affects all junctions that do not have a specific configuration in their definition (see next page).

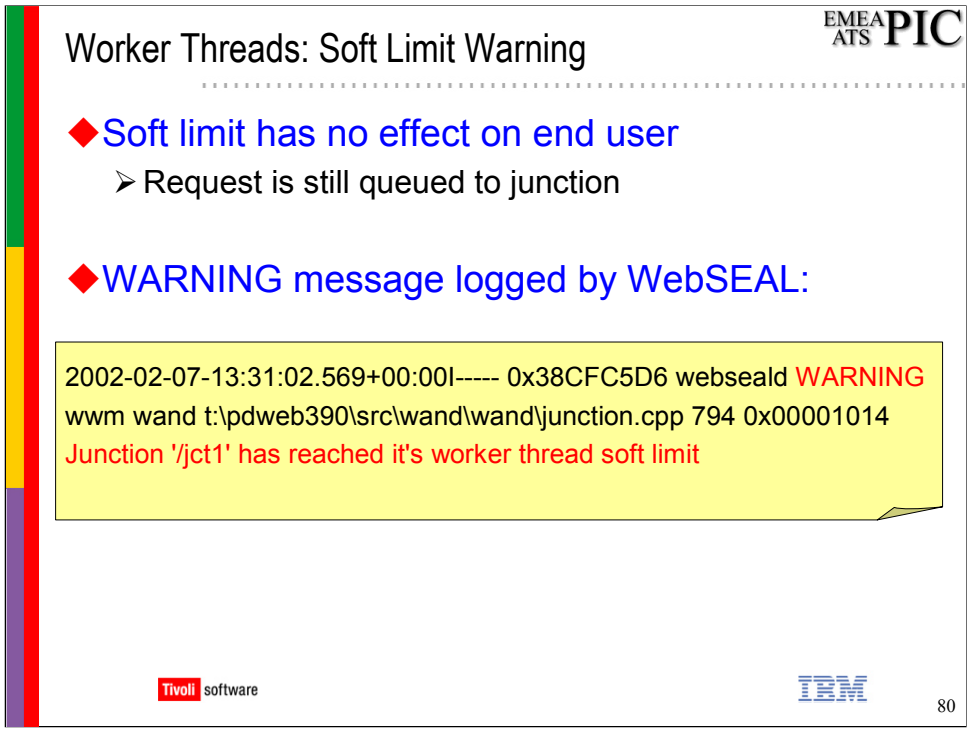
By default both the hard and soft limit in webseald.conf is set to 100. This effectively means that there is no limit for any junction and so the function is disabled.

To change the limits simply change the appropriate parameter to a value less than 100.



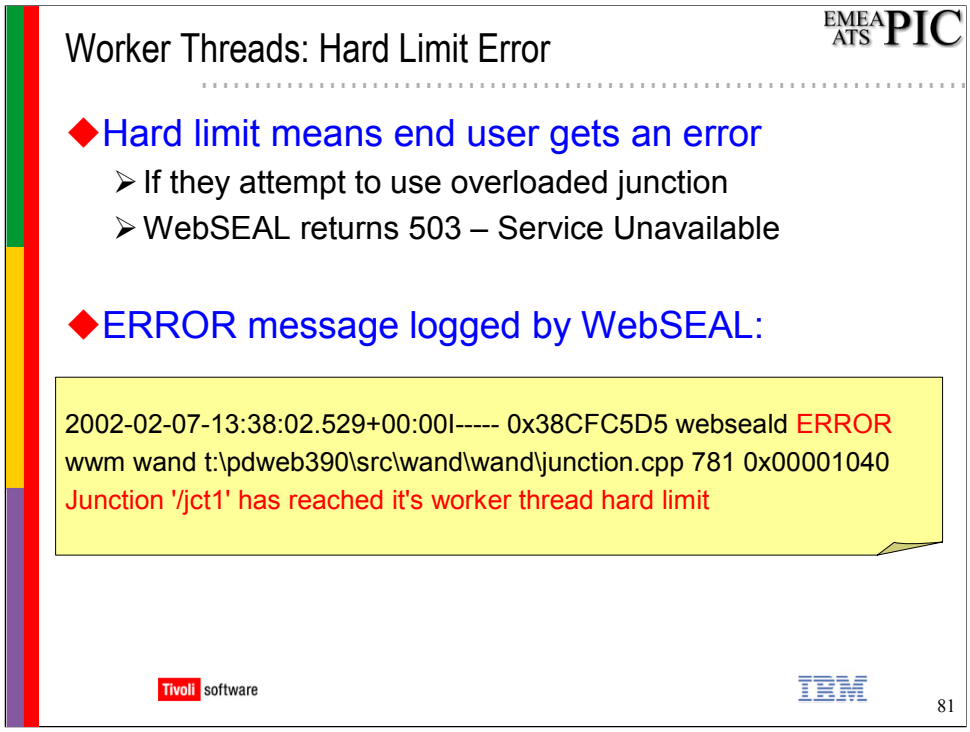
To set a soft and hard limit on an individual junction specify the values when the junction is created. Values specified on the junction override the value specified in webseald.conf.

The lower-case l is used to specify the soft limit and the upper-case L is used to specify the hard limit.



The slide above shows the warning message that is logged by WebSEAL when a soft limit is reached. The warning specifies the junction name.

The request is still queued to the junction – the end user is not aware that this limit has been reached.



The slide above shows the error message that is logged by WebSEAL when a shard limit is reached. The error specifies the junction name.

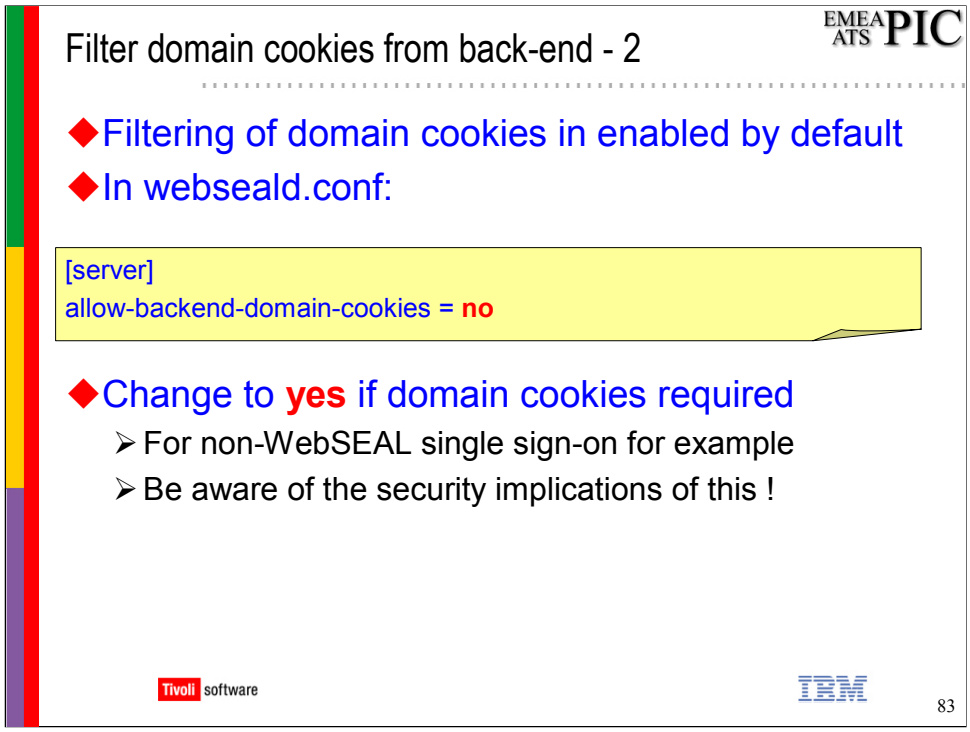
In the case of a hard-limit being exceeded the request is not queued to the junction. Instead an error is returned to the browser.

Some Web applications use cookies as a method of single sign-on to other applications in the same domain. IBM WebSphere , for example, does this using its LTPA cookie to allow a user to access multiple different WebSphere servers that share the same user registry with only a single authentication. When the user authenticates a domain cookie is set at the browser. If the user accesses a different server in the same DNS domain the cookie is sent with this request and can be used to identify the user without their interaction.

Normally a cookie used for authentication is encrypted using a secret key only known to valid servers. This ensures that a rogue user cannot create new cookies to gain access. However, this does not stop someone who can obtain a valid cookie from replaying it to get access – they don't need to read it to use it.

Domain cookies are more at risk from theft than host cookies because any Web Server in the DNS domain will receive the domain cookies of a user that accesses it. A rogue user who can set up a Web server in the same domain as the legitimate servers – and persuade users to visit it – can collect cookies and use them for access.

By default WebSEAL converts domain cookies set by junctioned servers into host cookies because of these considerations. The back-end server that set the cookie will still receive it with each request by the cookie will not be sent to other hosts in the DNS domain.

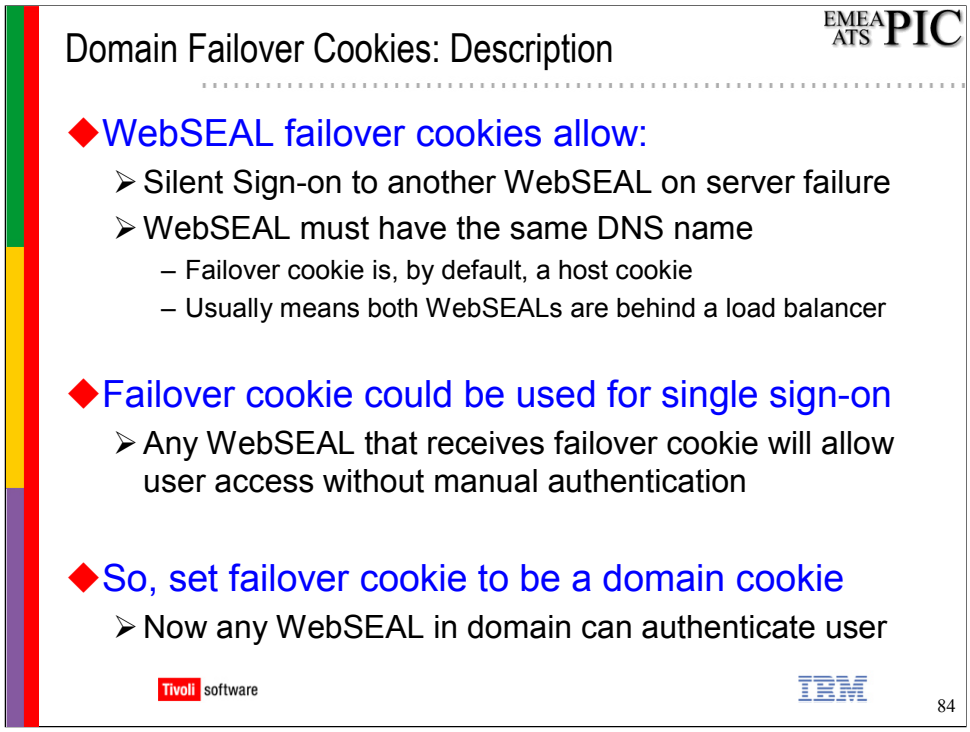


In some cases it might be required that back-end servers be allowed to set domain cookies on the browser. This might be because single sign-on is required with other servers that are not protected by WebSEAL or for some other reason such as user tracking.

WebSEAL now provides the option to allow domain cookies to be set by back-end servers. This configuration option, *allow-backend-domain-cookies* is in the *[server]* stanza of webseald.conf. By default it is set to *no*.

If the option is changed to *yes* then WebSEAL will not modify any cookies set by the back-end server. This affects all junctions.

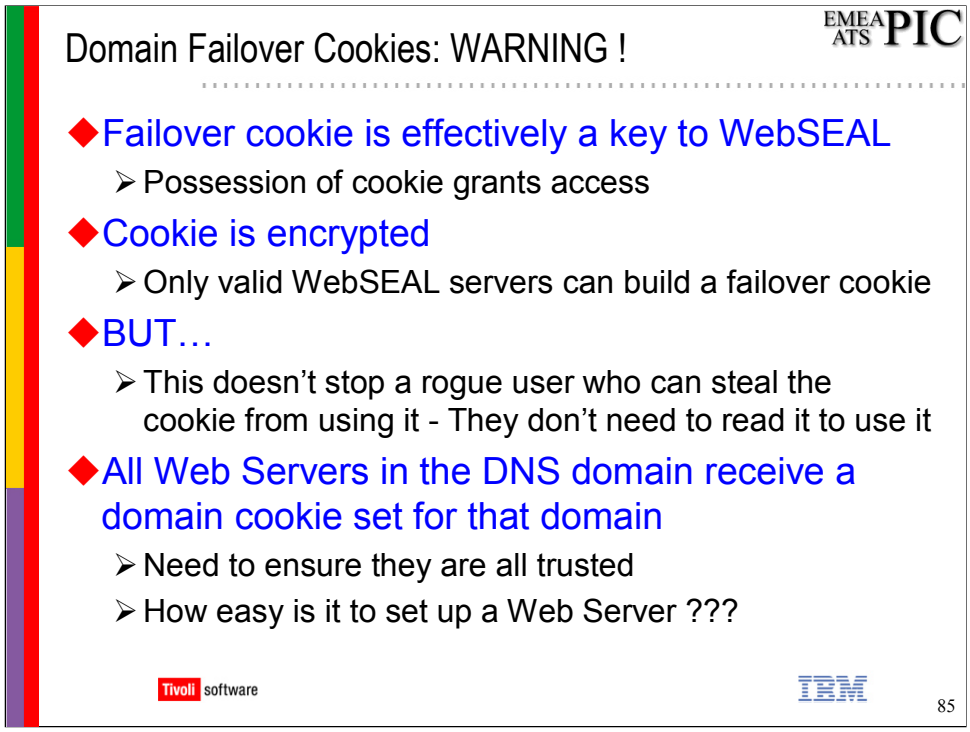
Before modifying this parameter be sure that you understand the security implications of using domain cookies in your environment.



In AM v3.7 the concept of failover cookies was introduced. These were intended to allow a user that was re-routed to another WebSEAL server as a result of a failure to access that server without having to re-authenticate. Both WebSEALs had to appear to have the same DNS name because, in the interests of maximum security, the failover cookie was a host cookie so it would not be sent to a server with a different name.

Since the introduction of the failover cookie, many Customers have requested that it be possible to use it for single sign-on in the same way that WebSphere uses the LTPA cookie. The change to make this possible is fairly small – simply change the fail over cookie to be a domain cookie rather than a host cookie. The cookie is set by the WebSEAL that authenticates the user and any WebSEAL that receives it will allow access without performing an authentication.

In AM v3.9 a configuration option has been added to allow the failover cookie to be a domain cookie – this allows it to be used for single sign-on.

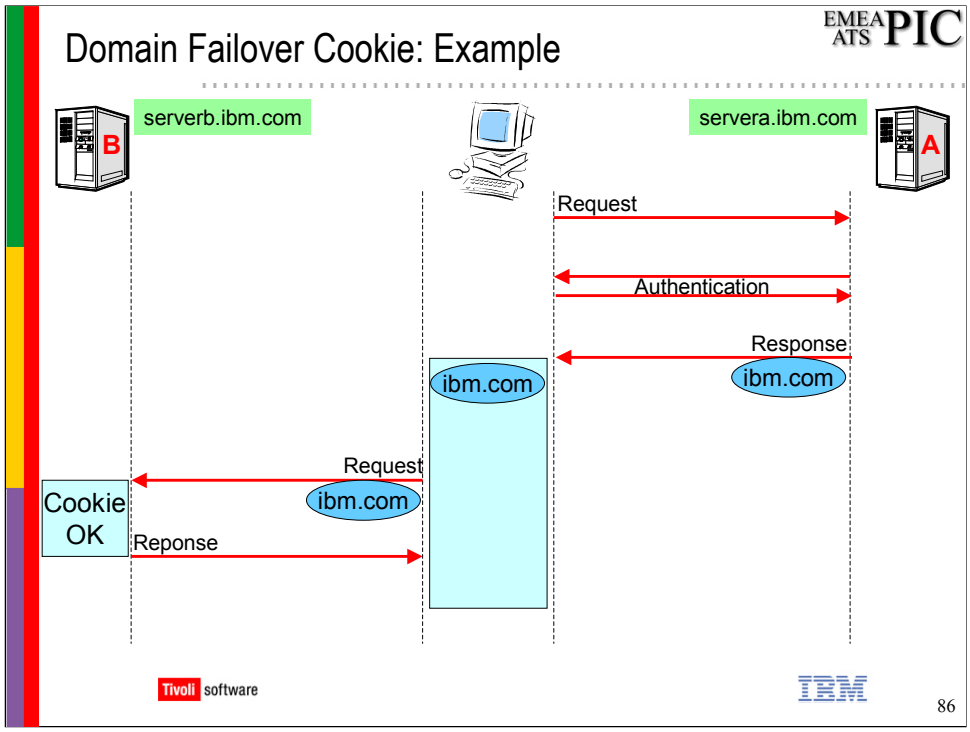


Careful consideration should be taken before deciding to use WebSEALs failover cookie (or any vendors cookie-based SSO solution) for single sign-on.

Although the failover cookie is encrypted so that it can only be generated by a legitimate server a failover cookie is effectively a key to WebSEAL. Anyone who presents the failover cookie (before it expires) will be given access to WebSEAL **without further authentication**.

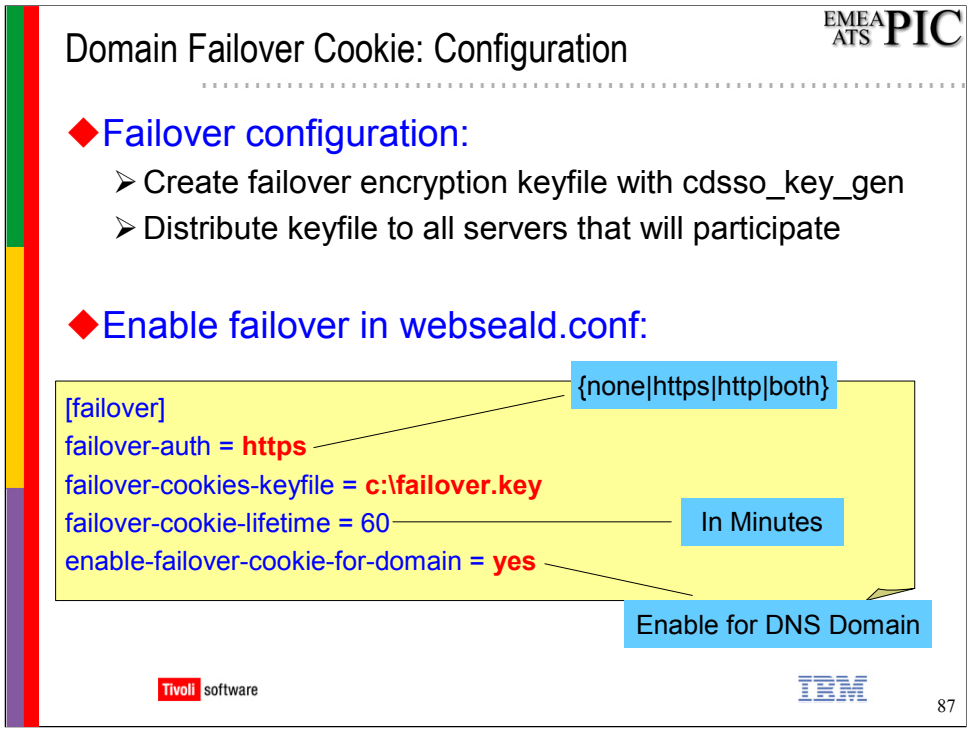
Browser vulnerabilities may make it possible to steal a cookie from a browser but even without this consideration a domain cookie is still vulnerable to a rogue in the same DNS domain as the legitimate servers. If users connect to this server the failover cookie will be sent, quite legitimately to that server. The owner of this server could then collect these cookies and use them to get unauthorized access to WebSEAL.

Ask yourself how easy it would be for someone to set up an unauthorized WEB server in the same DNS domain as your WebSEAL servers. You need to consider this before using domain failover cookies if you are protecting sensitive information.



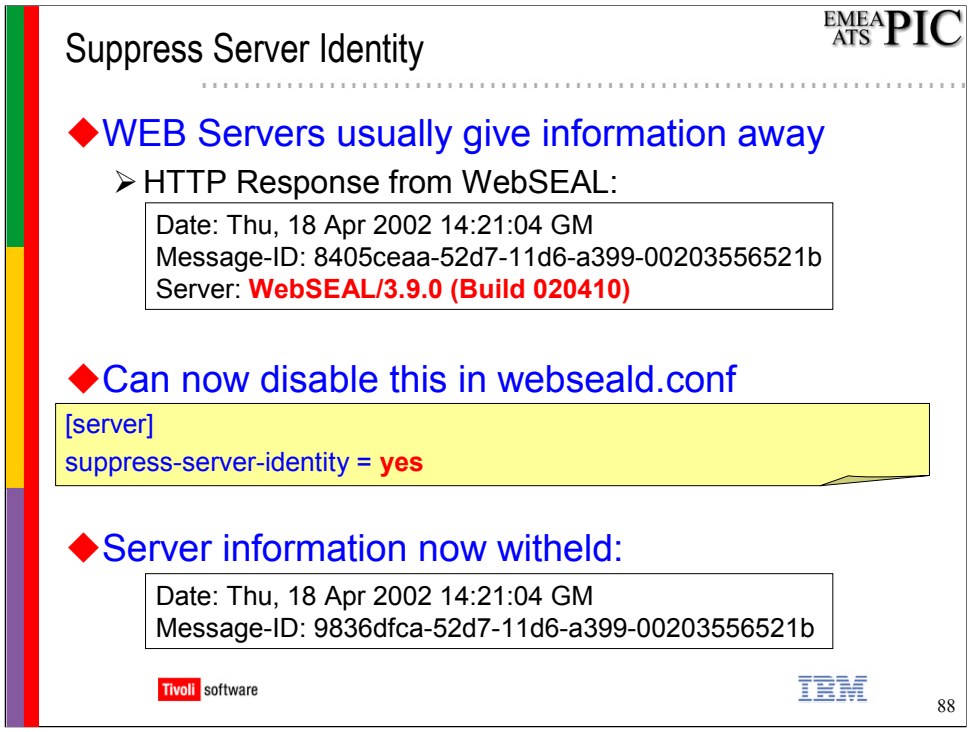
The diagram above shows how a domain failover cookie allows single sign-on to multiple WebSEAL servers.

The cookie is set by the WebSEAL that authenticates the user. When the user connects to another WebSEAL in the same DNS domain the failover cookie is included and this is used to authenticate the user rather than a second manual authentication.



To enable domain failover cookies configure failover as in previous releases. This means creating a failover keyfile on one machine (using the cdsso_key_gen utility) and then copy the file onto all WebSEAL servers that will take part in the SSO environment. In webseald.conf enable failover for http, https or both and give the filename of the failover keyfile.

In order to enable failover cookies to be domain cookies change the *enable-failover-cookie-for-domains* to *yes*.



Most Web Servers include identity information in the responses they send back to WEB clients. This information is not usually visible to the end-user (it doesn't show up in the page source) but it's easy to get (for example simply TELNET to port 80 and type HEAD / HTTP/1.1 [Enter]).

This information about the server allows attacks to be targeted to known vulnerabilities that are specific to that WEB server software. This potentially decreases the time taken to compromise the server.

In WebSEAL v3.9 it is possible to specify that WebSEAL should not send the Server field in it's responses. This is done by changing the *suppress-server-identity* parameter to "yes".

By default the parameter is set to "no" and so the server information is sent with every response.

WebSEAL Statistics and Tracing

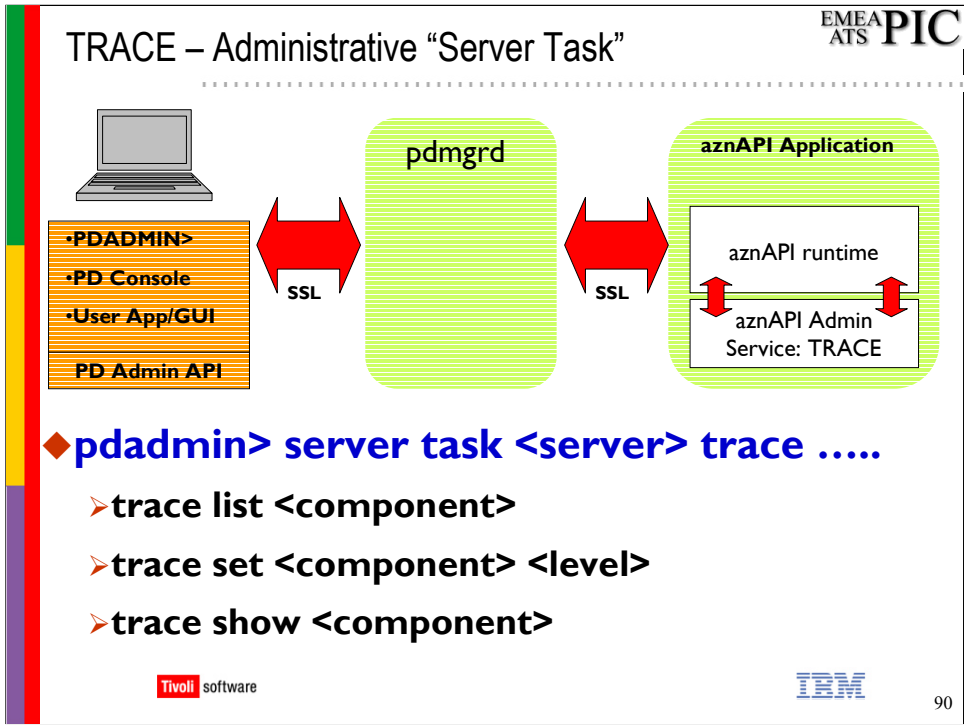
EMEA ATS PIC

- Statistics available from WebSEAL
  - Authentication, Cache performance etc
  - Real time or snap-shots to file
- Debug tracing available from WebSEAL
  - Most useful is HTTP trace
  - Shows flows User<->WebSEAL<->backend
- Introduced in PD v3.8
- Documented in AM v3.9
  - Chapter 3 in WebSEAL Administration Guide
  - Information on interpretting statistics

Tivoli software

IBM

89



The TRACE command is implemented as an aznAPI *Administration Service Plug-In*. That is, a plug-in extension to the aznAPI runtime for the purposes of administering the server.

The dynamic trace command is available for all AM Servers except Policy Server. For Policy Server, use a 'trace' command in pdmgrd.conf - syntax given a few foils up from here.

An aznAPI application passes its Administration Service definitions to PDMGRD during the *azn_initialize()* call.

Use *server show <server name>* to see what Administration Services are registered for any AM Server.

Use *server listtasks <server name>* to see the tasks supported by those Administration Services.

PDMGRD routes the *server task* command to the named <server> for execution, and the aznAPI invokes the appropriate plug-in to execute the command.

TRACE syntax

EMEA ATS PIC

- trace list <component>
  - displays the component hierarchy starting at the given component
- trace set <component> <level> > [file <file>]
  - Trace enabled or disabled by setting the <level> for a given component. <level> is value of 0-9. For example

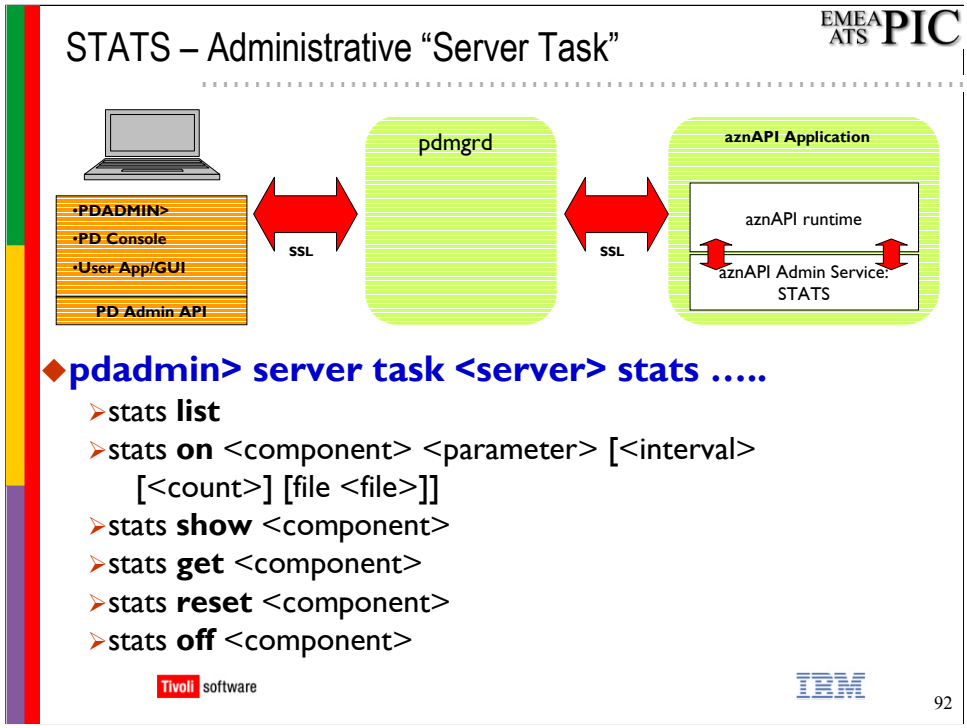
    server task <server name> trace set pd.acl 9
  - sets the Event Category of trace.pd.acl to level 9.
  - trace level 0 disables trace for that component
  - file <file> is an optional destination parameter. The default destination is stdout
- 'trace show' <component>
  - displays components with active trace levels for hierarchy starting at the given component

Tivoli software IBM 91

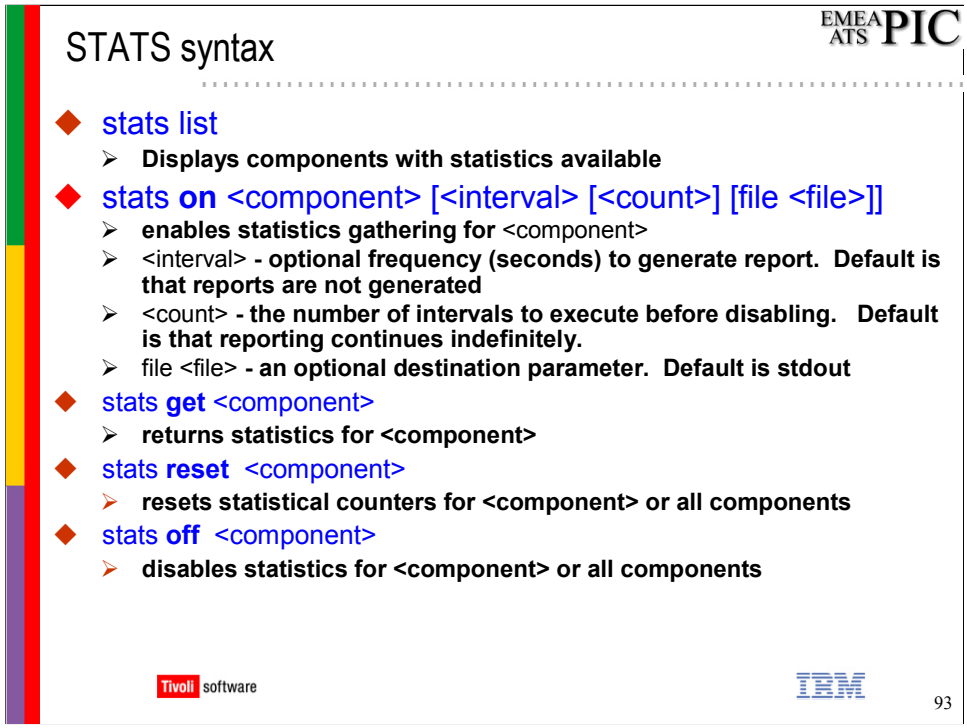
For example, the WebSEAL HTTP trace is generated by component

trace.pdweb.http

at level '2'.



Like the TRACE command, STATS is implemented as an aznAPI *Administration Service Plug-In*. In fact, it's the same service plug-in.



1.1.2. stats show task

This task shows a list of all components with stats enabled or the stats status for a single component.

Syntax: pdadmin> server <server> stats show [<component>]

Where:

<server> identifies the server for which stats are to be enabled

<component> is a component string as described above - only trace categories at or beneath this point are shown.

1.1.4. stats off task

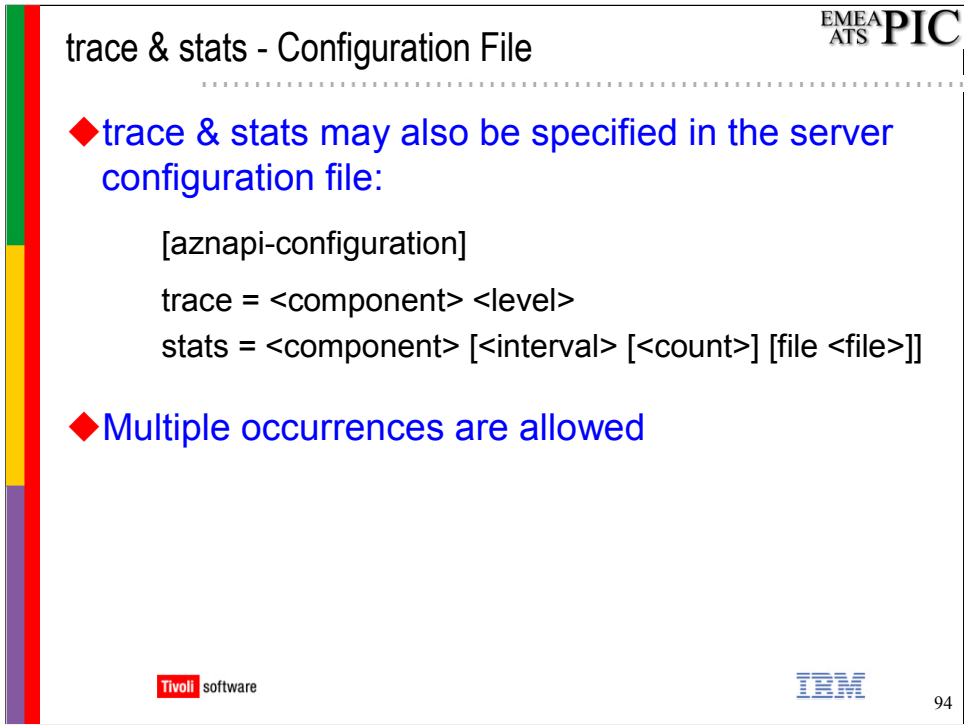
This task disables statistics for a component.

Syntax: pdadmin> server <server> stats off [<component>]

Where:

<server> identifies the server for which stats are to be enabled

<component> is the component for which stats are to be disabled – if not specified all stats are turned off



Enabling Trace events in the configuration file only supports STDOUT as a logging destination.
This is different to the pdadmin command line,
‘ pdadmin> server task <server_name> trace’
command which does allow a logging destination to be specified as part of enabling the creation of trace data.
In the configuration file the default logging agent for a 'trace' line item is stdout, if and only if there are no 'logcfg' statements that create a destination for those trace events. If a ‘logcfg’ statement does exist its destination is associated with the ‘trace = <component> <level> events.
For example, this statement in the .conf file

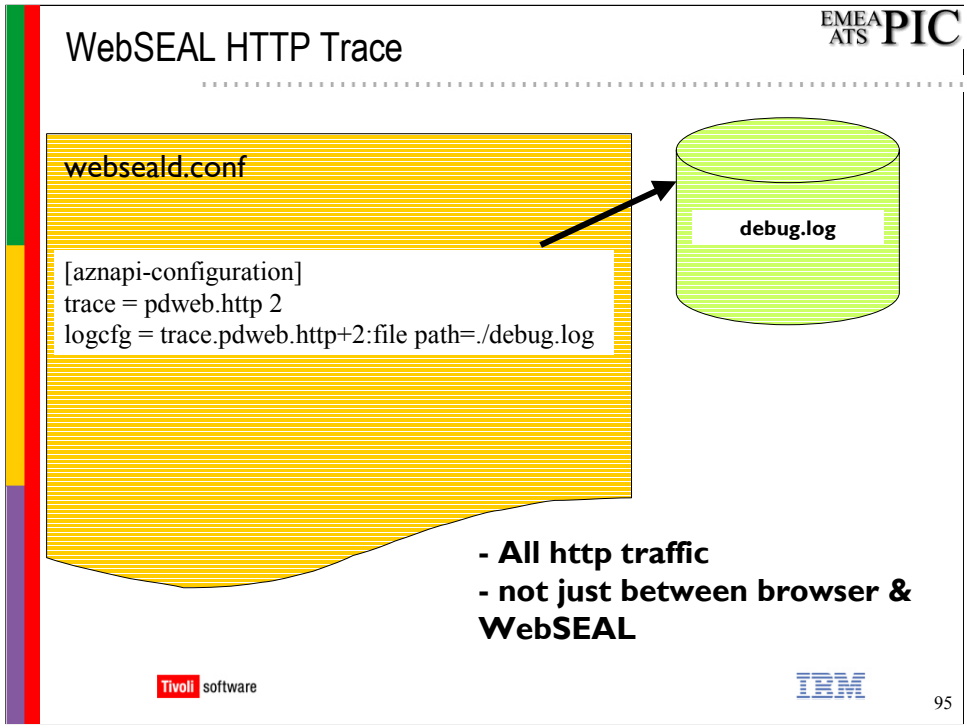
```
trace = pdweb.http 2
```

is equivalent to

```
trace = pdweb.http 2
logcfg = trace.pdweb.http+2:stdout
```

To define additional logging destinations, turn on trace generation and specify all the event recording agents explicitly.

```
trace = pdweb.http 2
logcfg = trace.pdweb.http+2:stdout
logcfg = trace.pdweb.http+2:file path=./trace.log
```



This is the trace that creates an entry for every http request/response sent by WebSEAL. It is formatted to show entries as

1) Browser → WebSEAL
2) WebSEAL → Backend
3) WebSEAL ← Backend
4) Browser ← WebSEAL

It is a very useful trace for problem analysis.

User Authorisation of Trace & Stats commands

EMEA ATS PIC

- Use of Trace command requires the “t” (Trace) bit on object:

  /Management/Server.

- The same “t” bit is also required for use of the “stats” command

Tivoli software

IBM

96