IOPLEX
Communications Software

https://www.ioplex.com/
support@ioplex.com
Jespa 2.0.7 R725

Jespa Operator's Manual

This document describes requirements, installation and operating instructions for the Jespa Java software library which implements a variety of security related functionality, mostly applicable to Microsoft AD DS environments.

Table of Contents

Requirements

The following requirements must be satisfied for the Jespa library to function as described in this document.

- Java 1.7 or later with unlimited cypto policy (Java 9+, 8u161+ or 7u171+).
- A Computer account (not a User account) must be created in Active Directory to allow the Jespa library to perform common AD DS operations such as validating NTLMv2 credentials using SecureChannel NETLOGON with the HttpSecurityService (or HttpSecurityFilter). This requirement is described in great detail in other sections.

  Note: Computer account passwords to NOT expire. The “Domain member: Maximum machine account password age” policy instructs domain members to periodically submit a password change but they are NOT required to. The Jespa Computer account will not stop functioning after the policy period.
- NTLM over HTTP cannot traverse proxies without help. See the *Proxying and Load Balancing with NGINX* section and Issue 2 in the *Possible Issues* section.
- Windows SSPI based Single Sign-On (herein referred to as Windows built-in SSO) authentication requires that the user is logged into a Windows device with their AD DS credentials. The target server may also need to be added to the "Local intranet" zone of the browser client.

Validating NTLM Credentials with the NETLOGON Service

One unique feature of Jespa is it’s high quality NETLOGON secure channel implementation. The Jespa NtlmSecurityProvider can validate NTLM credentials using the NETLOGON service on Active Directory domain

controllers just like a Windows server would. However, there are two notable requirements for using this feature:

1. A Computer account must be created for Jespa to communicate with the NETLOGON service. A regular User account will be rejected by the NETLOGON service.

   Note: This account does not, and must not, refer to an actual computer or Windows OS instance.

2. If you are using multiple instances of Jespa to validate Windows credentials, each instance will require it's own Computer account. An instance of Jespa is defined as a ClassLoader with Jespa classes loaded through it. Because webapps usually use their own ClassLoader, you will need to create a separate Computer account for each webapp.

   Technical Details: AD maintains state about each "computer" connected to it. That state is indexed by the NetBIOS hostname presented to the domain controller when the NETLOGON connection is established. If multiple instances of Jespa attempt to use the same Computer account, AD may deallocate what it thinks are redundant connections. If Jespa then tries to use one of those deallocated connections, the "return authenticator check failed" error described in *Issue 12* in the *Possible Issues* section will occur (although Jespa is very good at transparently recovering from this error).

NETLOGON and lsarpc Port Requirements

MSRPC TCP transport uses the Windows end point mapper running on TCP port 135 to determine the ports of various services such as NETLOGON or lsarpc. If Jespa must communicate with domain controllers through a firewall, you may need to investigate precisely which ports are being used and open them accordingly.

Amazon AWS documentation for EC2 hosted domain controllers list the port ranges for the “NetLogonR” service as 49152 – 65535.

To determine which specific ports are actually being used, you can set the property log.level = 4 and then search the log output for “MsrpcEpmMap” for entries like the following:

```
MsrpcEpmMap: ncacn_ip_tcp:10.9.8.70[netlogon] maps to port: 49670
...
MsrpcEpmMap: ncacn_ip_tcp:10.9.8.70[lsarpc] maps to port: 49670
```

Proxying and Load Balancing NTLM SSO with NGINX

NTLM over HTTP cannot traverse proxies without help. For the Jespa HttpSecurityService this means adding a directive to the proxy configuration to set a Jespa-Connection-Id header to a value that includes the remote IP and port of the client. The HSS can use this value to map each request to the correct security context.

The examples that follow use the Jespa-Connection-Id header with NGINX to proxy, forward and load balance traffic to Jespa instances. Any proxy that supports setting a header to the client’s remote IP and port such as Apache or IIS should work as well.

Proxying and Forwarding

To proxy or forward requests to Jespa through NGINX, use nginx.conf directives like the following:

```
server {
    location / {
        proxy_pass http://localhost:8080;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
        proxy_set_header Jespa-Connection-Id $remote_addr:$remote_port;
    }
}
```

The proxy_http_version and Connection header enable connection caching. Otherwise, set the Jespa-Connection-Id header to the remote IP and port of the client. This is a good configuration for off-loading TLS and

unprotected static content.

Note: This Jespa-Connection-Id method does not require persistent TCP connections (Keep-Alive) or HTTP 1.1 between NGINX and Jespa. A valid Jespa-Connection-Id is all that is needed to map each request to the correct security context.

Load Balancing

Load balancing to any number of Jespa instances is also easy with NGINX using nginx.conf directives like the following:

```
upstream backend {
    ip_hash;

    server localhost:8080;
    server localhost:8081;

    keepalive 16;
}
server {
    location / {
        proxy_pass http://backend;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
        proxy_set_header Jespa-Connection-Id $remote_addr:$remote_port;
    }
}
```

The ip_hash directive ensures that clients with the same IP are directed to the same backend server (localhost was used in this example just for testing everything on one machine – use remote server IPs in production).

Note: NGINX Plus has an “ntlm” directive that should also work. However, for Jepsa, using the Jespa-Connection-Id method will be more efficient because it does not require creating tables of connections for each client.

DNS Requirements and Properties

Just like Windows servers, Jespa uses DNS SRV lookups to locate AD DS services to provide fault-tolerance and simplify configuration.

The specific queries are SRV lookups for names that look like the following:

```
_ldap._tcp.dc._msdcs.EXAMPLE.COM
  or
_ldap._tcp.Paris._sites.dc._msdcs.EXAMPLE.COM
```

where “EXAMPLE.COM” is the qualified DNS domain name in which AD DS servers are to be located and, in the second example, “Paris” is the AD Sites & Services "site" that the web server is in.

Note: Even though these names begin with _ldap, Jespa does not use LDAP at all for SSO. It uses MSRPC.

Active Directory Sites and Services

AD Sites & Services are used to geographically partition domain controllers in WAN environments where network communication with distant locations may be relatively slow or restricted. If you use AD Sites & Services, failing to set the dns.site property described below may result in poor performance or failure.

Note: The Jespa Setup Wizard will attempt to retrieve the AD Sites & Service site name for the machine on which it runs and will display it with the other Jespa configuration properties if the wizard completes successfully. Otherwise, if you cannot run the Jespa Setup Wizard on a machine with close network proximity to the server that will be running Jespa, ask your your network administrator if AD Sites & Services is being used and precisely what the “site name” is for domain controllers with good connectivity to your web service.

Note: It is not uncommon for users to report connectivity issues (usually in the form of timeouts) that are completely resolved after setting the dns.site property *correctly*.

DNS Properties

The behavior of most Jespa components is controlled by maps of key-value properties. The following is a table of properties used by Jespa to control DNS behavior.

Note: When entering properties into a properties file of a Jespa component like the HttpSecurityFilter or into a web.xml file, property names MUST to be prefixed with "jespa." like "jespa.dns.servers" and not just "dns.servers".

Name	Description	Example
`dns.servers`	A comma separated list of IP addresses indicating the Microsoft DNS servers that Jespa should use. If multiple DNS servers are supplied and the currently selected DNS server fails to respond, Jespa will transparently fail-over to the next server. If a response to a query is not satisfied by any server in the list, an exception will occur indicating the last error (most likely a timeout). If this property is not supplied, the default DNS server used by the JVM will be used but failover will not be performed. It is better to always set this property. Use the `ipconfig /all` command on the commandline of a Windows server or workstation in the target domain to determine a suitable value for this property.	`192.168.10.10,192.168.20.20`
`dns.site`	The AD Sites & Services "site" representing the geolocation of the Jespa instance. See description above.	`Paris`
`dns.records.path`	The path to a file containing DNS records used to pre-populate the DNS cache. See *The DNS Records File* section below. Note that each backslash in Java properties file must be escaped with an additional backslash.	`C:\tomcat\webapps\example\WEB-INF\dns.txt`
`dns.cache.ttl`	The number of milliseconds that DNS responses are cached. The default value is 5000 ms.	`60000`
`dns.jndifactory.classname`	Specifies the `java.naming.factory.initial` property value used by the Jespa DNS routines. There is no default value for this property. The default behavior is to allow JNDI to select a suitable DNS factory class.	`com.sun.jndi.dns.DnsContextFactory`
`authority.dns.names.resolve`	If set to false, this disables DNS SRV lookups used to resolve the "bindstr" property in which case the bindstr MUST be set to a fully qualified DNS hostname and not a domain name.	`false`
`authority.dns.names.resolve.sld`	If set to true, this enables single-label domain name lookups. The default value is false because single-label DNS domain names are deprecated in AD DS.	`true`

The DNS Records File

Jespa supports bypassing DNS queries using a DNS records file. This can be useful for restricting Jespa to a specific subset of domain controllers such as for navigating firewalls and for debugging purposes. An example of this file follows:

```
# Rotate through only dc1, dc2 and dc3
_ldap._tcp.dc._msdcs.EXAMPLE.COM SRV 0 100 389 dc1.example.com
```

```
_ldap._tcp.dc._msdcs.EXAMPLE.COM SRV 0 100 389 dc2.example.com
_ldap._tcp.dc._msdcs.EXAMPLE.COM SRV 0 100 389 dc3.example.com
```

If the above DNS records file is supplied with the dns.records.path property, DNS SRV lookups for the name _ldap._tcp.dc._msdcs.EXAMPLE.COM will be bypassed and the data supplied in the file will be used instead.

Note: The above example assumes that AD Sites & Services is not being used. If it is and the dns.site property was set to “Paris”, the "Paris._sites." components of the record names would have to be included for the records to match.

The format of each record is always the name, type and then data that depends on the record type separated by one or more spaces. Currently only SRV and A records are supported.

Note: Tabs are not supported. All fields must be separated by only spaces.

The data for SRV records is priority, weight, port and target. The data for A records is simply the dot-quad IP address of the host. This is the same format as DNS zone transfer files.

Note: If the DNS records file is modified, it will automatically be reloaded within 5 seconds.

As illustrated by the example above, multiple records can have the same name. In this case, the Jespa DNS logic will rotate through the records each time the name is queried.

Installation

To authenticate clients using Windows built-in SSO, such as with the HttpSecurityService (or HttpSecurityFilter), a Computer account must be created in AD DS with a known password as described in this section.

Note: Only a Computer account can validate NTLM credentials with the NETLOGON service on AD DS domain controllers. The NETLOGON service will reject a regular User account (with the error described in Issue 4 in the Possible Issues section).

Step 1: Create the Computer Account in AD DS

As a domain Administrator, double-click on the SetupWizard.vbs file from the Jespa package.

Note: The Jespa Setup Wizard will NOT modify your AD schema or do anything other than create a Computer object with the specified DN and set it's password. If you do not want to run a VBS script as Administrator, see the *Alternative Step 1: Creating a Computer Account Manually* section.

Note: We recommend that you run SetupWizard.vbs from a host with close network proximity to the machine that will be hosting Jespa so that the wizard can query the host for configuration properties specific to the target network. Windows Server security policies may prevent SetupWizard.vbs from running successfully in which case a conventional workstation might be used.

The Jespa Setup Wizard will step through creating a new Computer account, set it's password and, if successful, display Jespa configuration properties corresponding to the account just created.

Note: The Jespa Setup Wizard must be executed by either Administrator or a user with permissions sufficient to create the account and set it's password.

Note: When prompted for a DNS hostname of a domain controller, we recommend entering a specific domain controller as opposed to a DNS domain name to reduce the possibility of replication delay issues. The bindstr property can be changed to just the domain name later to provide proper fault tolerance.

Setup Wizard should launch notepad.exe with configuration properties that correspond to the Computer account just created. Consider the following example output:

```
# Generated by the Jespa Setup Wizard from IOPLEX Software

jespa.bindstr = dc100.example.com
jespa.dns.servers = 192.168.44.110,192.168.22.115
jespa.dns.site = Paris
jespa.service.acctname = jespa1$@example.com
jespa.service.password = cav-22^bim.33~kip+92$
```

These configuration properties can be copied directly into the HttpSecurityService properties file.

Note: If you receive an error "The specified domain either does not exist or could not be contacted", this indicates that the machine running the Jespa Setup Wizard does not have access to a suitable domain (such as because the machine is not joined to an AD DS domain).

Note: If you receive an error "The object already exists", this indicates that a Computer account with the same name already exists. Choose a different name.

Note: The SetupWizard.vbs script requires that the LICENSE.txt file is in the same directory. It needs this file to display the EULA.

Alternative Step 1: Creating a Computer Account Manually

To create the Jespa Computer account manually, use whichever utility you prefer to create a new Computer account object such the Active Directory Users and Computers MMC Snap-In. The name of the account should be no more than 15 characters consisting of only the characters A-Z, a-z, 0-9, hyphen (-) and underscore (_).

Note: If you need to create multiple accounts, use a common prefix for all of them if possible. This might help later with

certain security policy settings (see Issue 15 in the Possible Issues section for an example).

Setting the Password Manually using PowerShell

If you have access to a machine with the `addsadministration` PowerShell module installed (usually only servers), you can set the Computer account password with a PowerShell command with the Computer account name (with the $ sign) like:

```
PS C:\Users\Administrator> Set-ADAccountPassword -Reset -Identity jespa1$
Please enter the desired password for 'CN=jespa1,CN=Computers,DC=example,DC=com'
Password: *****************
Repeat Password: *****************
```

IMPORTANT: Password length should be NO LESS than 20 characters and consist of at least three of the four character classes (upper, lower, digits and meta).

Note: Computer account passwords to NOT expire. The “Domain member: Maximum machine account password age” policy instructs domain members to periodically submit a password change but they are NOT required to. The Jespa Computer account will not stop functioning after the policy period.

Setting the Password Manually using the SetComputerPassword.vbs Script

Alternatively you can set the Computer account password with the included SetComputerPassword.vbs script which should work without any special PowerShell modules. Simply start a command prompt and run the following command:

```
C:\temp\jespa-2.0.0>SetComputerPassword jespa1$@example.com cav-22^bim.33~kip+92$
                        Must have the $ sign!^
```

IMPORTANT: Password length should be NO LESS than 20 characters and consist of at least three of the four character classes (upper, lower, digits and meta).

Note: This VBScript does NOT do anything other than set the password on the named Computer account.

Note: To run the above command, the operator will need to be logged in as Administrator or a user with permissions sufficient to set the password on a Computer account.

This account can now be used with Jespa components such as the HttpSecurityFilter as described in Step 2.

Step 2: Test the Computer account with the Example Webapp

If you have an HTTP Servlet container available, the best way to test the Jespa Computer account and installation is to enable and exercise the example webapp.

Note: This step is not required. However if you encounter an issue, it may be more easily diagnosed and resolved if you can test your configuration with the relatively simple example webapp.

To install the example webapp do the following:

1. Copy the `examples/jespa/` directory into your servlet container webapps directory (or soft link to it).
2. Place the jespa-jakarta-2.0.0.jar (or jespa-2.0.0.jar if you’re using a Java EE application server) into `jespa/WEB-INF/lib/`.
3. Edit `jespa/WEB-INF/example.prp` and replace the properties with those generated in Step 1. See *The HttpSecurityService and HttpSecurityFilter* and *The NtlmSecurityProvider Properties* sections for detailed descriptions of what the various properties do.
4. Restart the Servlet container and visit the `jespa/index.jsp` resource.

If the Jespa successfully authenticates the user, the webapp will display the user’s identity with links to exercise functionality of the HttpSecurityService and underlying security provider.

Note: If you get a password dialog at this point, you probably need to add the target hostname in Internet Options on the client (search for and run `inetcpl.cpl`) as detailed in the *Requirements and Windows Settings for Single Sign-On (SSO)* section.

If anything does not work as expected, set the property `jespa.log.level = 4` and `jespa.log.path` to an appropriate location and monitor the log file while you try the webapp.

For maximum security, see the additional installation steps in the *SPN and Channel Bindings (EPA)* section.

Note: If you get a "SAM database ... does not have a computer account" exception at this point, there is something wrong with the Computer account created in Step 1. See the *Possible Issues: Issue 4* section for details.

Step 3. Copy Jespa Into Your Application Files

If the example webapp works as expected, copy the working web.xml directives, the Jespa properties file (or incorporate the properties directly into the web.xml) and Jespa jar file into your project files and test Jespa with your application. For other components like the SASL server or client, JAAS LoginModule or similar, simply copy the Jespa jar file into your project and invoke the code with the appropriate properties.

Upgrading

To upgrade a Jespa installation, download the latest Jespa package from the IOPLEX website and "activate" the jar with your purchased activation key file.

Activating the Jespa Jar File

Before you can deploy the Jespa jar file, the jespa/license.key resource within it must be updated with your purchased activation key file. The default jespa/license.key is the trial key which will expire after 60 days.

To update the jespa/license.key resource, unzip the Jespa package containing the desired jar. Start a command shell and run the following command.

Note: The below command should be typed in one line.

```
$ cd /path/to/jespa-2.0.0
$ java -cp jespa-jakarta-2.0.0.jar jespa.License -u
jespa_premium_license_examplecom_SN2311320210111.key<enter>
License of this jar:
Jespa license.key decrypted successfully:
serialNumber: [SN2314220220504]
   userLimit: [0]
  groupLimit: [0]
  expiration: [Sat Jul 23 20:16:43 EDT 2022]
 description: [Jespa 60 Day Trial Licensing Key]
Supplied key file:
Jespa license.key decrypted successfully:
serialNumber: [SN2311320210111]
   userLimit: [0]
  groupLimit: [0]
  expiration: [0]
 description: [Jespa Premium License Key: 10 inst at "example.com"]
Update this jar with supplied key file? 'Y': y
jespa/license.key resource updated: 444 bytes
Deleting /path/to/jespa-2.0.0/jespa-jakarta-2.0.0.jar and overwriting with
/tmp/jespa_7143269194982998868.tmp
License updated successfully
$
```

Note: If you get a “Failed to decrypt license key” error, this likely means that your activation key is too old for the particular version of Jespa that you are trying to use. You can either use your existing key with a previous version of Jespa or purchase a new key. Or, it is also possible that the key file is corrupted.

Note: The Jespa activation key file is simply stored within the jar file as the "jespa/license.key" resource. Jar resources

can be updated using the jar or zip commands or by using the java.util.zip.* classes.

If the command is successful, the jar has been activated and may now be copied to the number of installations permitted by your purchased activation key.

Validating the Activation Key of a Jespa Jar File

You can validate the activation key of any Jespa jar file with a command like the following:

```
$ java -cp jespa-jakarta-2.0.0.jar jespa.License
License of this jar:
Jespa license.key decrypted successfully:
serialNumber: [SN2311320210111]
   userLimit: [0]
  groupLimit: [0]
  expiration: [0]
 description: [Jespa Premium License Key: 10 inst at "example.com"]
```

The HttpSecurityService and HttpSecurityFilter

The HttpSecurityService component can be used to perform advanced multi-mechanism HTTP authentication and authorization. The HttpSecurityFilter implements the standard Servlet Filter interface around the HttpSecurityService and provides an easy path for existing applications to implement Windows built-in SSO.

Note: The Installation section of this manual describes how to use the HttpSecurityFilter to implement SSO with the Jespa example webapp.

If the client is successfully authenticated either by SSO, by browser password dialog or by a form-based login, the HttpSecurityService will insert a custom HttpServletRequest that overrides getRemoteUser, getUserInRole, etc. See the HttpSecurityService API documentation for details.

HttpSecurityService Mechanisms

The mechanisms used by the HttpSecurityService are defined by the previously described security providers. The following is a summary of which security providers can be used with the HttpSecurityService and what mechanism functionality they provide.

Security Provider	HttpSecurityService Mechanism Functionality
`NtlmSecurityProvider`	Provides HTTP NTLM authentication including Windows built-in SSO. Most browsers including Edge, Chrome and Firefox can perform this type of “built-in” SSO. If for some reason SSO cannot be performed (such as because the client is not logged into an AD DS domain), the browser will present the user with a password dialog.
`LdapSecurityProvider`	Provides explicit login authentication using the traditional LDAP bind method (LDAP cannot be used to implement true SSO). This security provider may be used to authenticate HTTP clients against AD DS or an RFC-based LDAP server such as OpenLDAP.
`ChainSecurityProvider`	A special wrapper around a "chain" of other security providers. This may be used to authenticate HTTP clients against multiple independent authorities at the same time (although only the first element in the chain can do SSO). If authentication is successful, the ChainSecurityProvider will act as an intelligent proxy to the successful provider for the duration of the client's HTTP session.

Note: Always restart client browsers if you change your configuration to use a different mechanism. Otherwise, the browser could cache state that is not applicable to the new mechanism resulting in errant behavior.

For a more consistent site appearance and user experience, the HttpSecurityService may be configured to perform HTML form based logins. It also supports group based access control, anonymous access and other features that are described in detail in the sections that follow.

Note: If you are forwarding requests through another web server like NGINX, see the *Proxying and Load Balancing with NGINX* section for important information.

Note: Once a browser has performed NTLM HTTP authentication with a server, it can proactively re-authenticate POST

requests. This means that you may not be able to selectively filter only some content on a server because the browser may try to authenticate for content that is not protected by the filter and therefore is not prepared to handle the authentication. Ultimately, it is strongly recommend that `<url-pattern>/*</url-pattern>` is used with excludes to selectively exclude certain resources (described below).

Note: If you want to selectively disable the HttpSecurityService, such as because you need to use an alternative authentication mechanism you will need to use the *Anonymous Access* feature to allow the request to pass through without being authenticated.

HttpSecurityService Properties

Like all Jespa components, the HttpSecurityService is controlled using properties. The following table summarizes properties which will be described in more detail in the sections that follow.

Note: Backslashes in Java properties files, such as in path names and qualified group names, must be escaped with an additional backslash.

Note: Setting any of the `http.parameter.*` properties will cause the request input stream to be read which may not be compatible with certain request processing such as REST services.

Name	Description	Example Value
`provider.classname`	The name of a jespa.security.SecurityProvider class that the HttpSecurityService should use to perform authentication and any other security functions.	`jespa.ntlm.NtlmSecurityProvider`
`properties.path`	A path relative to the web-app root from which properties should be loaded. Properties loaded from a properties file will override any properties supplied directly to the HttpSecurityService constructor (such as those specified with web.xml init-params when using the HttpSecurityFilter). Note: Property values that begin with file:// will be interpreted as an absolute path relative to the root of the host operating system (regardless of whether or not the application is packaged within a WAR file). Meaning property values like file://H:/myapp/ntlm.prp or file:///s1.busicorp.local/dev/myapp/ntlm.prp should work.	`/WEB-INF/example_ntlm.prp`
`fallback.location`	An absolute path or URL that the client should be redirected to if they are denied access or an authentication protocol error occurs. See *The Fallback Location* section below for details. This resource MUST also be in the excludes list or the client will not be able to access it.	`/jespa/Login.jsp`
`excludes`	A comma separated list of paths relative to the webapp base that are excluded from protection by the HttpSecurityService. See *The Excludes List* section below for details.	`/Login.jsp,/support.html`
`http.parameter.username.name`	The name of the username HTTP request parameter used for form based logins. See the *HTTP Form Based Logins* section below for details.	`username`
`http.parameter.password.name`	The name of the password HTTP request parameter used for form based logins. See the *HTTP Form Based Logins* section below for details.	`password`
`http.parameter.logout.name`	The name of the logout HTTP request parameter that triggers security provider HTTP session state to be removed (this has no effect on the HTTP session itself). See the *Logging Out* section below for details.	`logout`
`http.parameter.anonymous.name`	The name of the anonymous HTTP request parameter used to bypass authentication. See the *Anonymous Access* section below for details.	`anon`
`groups.allowed`	A comma separated list of SecurityProvider specific group or account names that identify clients who are permitted access through the HttpSecurityService. If this parameter is not set, all users will be allowed access. See the *Group*	`EXAMPLE\Domain Admins, EXAMPLE\Engineers`

	Based Access Control section below for details.	
groups.denied	A comma separated list of SecurityProvider specific group or account names that identify clients who are to be denied access. This list is checked before the groups.allowed list. See the *Group Based Access Control* section below for details.	EXAMPLE\Sales, abaker@example.com
jespa.log.path	Specify a log file for debugging purposes (for Windows this might be C:\tmp\jespa.log). If no log file is specified, the default log stream is System.err.	/tmp/jespa.log
jespa.log.level	Use level 3 or higher for debugging purposes only. If no log level is specified, the default is 1. IMPORTANT: Setting this value higher than 3 will greatly increase the size of the log file and limit performance.	2
bindings.targetSpns.policy	Controls how the HSS behaves if the client does not submit a valid SPN. Possible values are: 0 - Disabled. No SPN checking is performed. 1 - Allowed (default). SPNs are checked but clients will not be rejected. The result is logged with log.level = 2 or higher. 2 - Required, SPNs must match an SPN in the bindings.targetSpns list or an SPN created from SubjectAltName entries in the server TLS certificate if any. If the client did not provide a matching SPN, the HSS will return 401 Unauthorized. See the *SPN and Channel Bindings (EPA)* section below and the HttpSecurityService API documentation for details.	2
bindings.targetSpns	A comma separated list of SPNs that this server accepts. An SPN is a scheme and host like HTTP/www.example.com (the scheme is always HTTP even if TLS is being used). If TLS is being used (it should be) and the server certificate has dNSName or iPAddress SubjectAltName entries (it should), they will be used to generate SPNs automatically in which case it should not be necessary to set this property at all (but it can be set to add additional SPNs).	HTTP/as5.example.com, HTTP/10.2.3.4
bindings.cert.hash.policy	Controls how the HSS behaves if the client does not submit valid channel bindings. Possible values are: 0 - Disabled. No channel bindings checking is performed. 1 - Allowed. Channel bindings are checked but clients will not be rejected. The result is logged with log.level = 2 or higher. 2 - Required. The client must submit channel bindings as specified by the bindings.cert.* properties. If the client did not provide matching channel bindings, the HSS will return 401 Unauthorized. See the *SPN and Channel Bindings (EPA)* section below and the HttpSecurityService API documentation for details.	2
bindings.cert.hash	Explicitly sets the channel bindings hash to a specific value in hex. If this property is set, all other bindings.cert.* properties will be ignored. Values for this property may be observed in the Jespa log file (although if a MITM attack was affected the value could be incorrect).	531B8FFAC96ED512AD94EAB3687658A3
bindings.cert.keystore.path	The path to the keystore file containing the certificate from which to calculate the channel bindings hash. If this property is not set, the HSS will not use a keystore and all other bindings.cert.keystore.* properties will be ignored.	etc/alma92as5.p12
bindings.cert.keystore.type	The type of the keystore containing the certificate from which to calculate the channel bindings hash. The default value is	JKS

	PKCS12.	
bindings.cert.keystore.provider	The provider of the keystore containing the certificate from which to calculate the channel bindings hash. The default value is SunJSSE.	SUN
bindings.cert.keystore.password	The password of the keystore containing the certificate from which to calculate the channel bindings hash. If a password is not required, do not set this property at all (which is not equivalent to an empty value).	alma92as5
bindings.cert.pem.path	The path to a PEM file containing the PEM encoded certificate from which to calculate the channel bindings hash. If the file contains multiple certificates, the first certificate with an extended key usage of serverAuth will be used.	etc/alma92as5.pem
bindings.cert.pem	The PEM encoded certificate from which to calculate the channel bindings hash. Linebreaks will be ignored so that the entire certificate can be set on one line such as within a properties file.	-----BEGIN CERTIFICATE----- MIICmDC...77txZ2sZ8=----- END CERTIFICATE-----
bindings.cert.url	An HTTPS URL serving the certificate from which to calculate the channel bindings hash. If other bindings.cert.* properties are not set, the HSS will query this URL (using a "dummy" TLS trust manager) to retrieve the server certificate and calculate the channel bindings hash. Note: While using only this property is convenient, it is not the most secure option. If a MITM attack occurs when the channel bindings are initialized, the incorrect hash could be generated. Properly secured installations should use the bindings.cert.keystore.* or bindings.cert.pem* properties instead. If this value is not set, the value returned by HttpServletRequest.getRequestURL().toString() will be used (but only once each time the HSS is (re)initialized).	https://ext2.example.com

If the HttpSecurityService is being use through the HttpSecurityFilter, these properties and the properties of the SecurityProvider being used, may be set using standard init-params in the web.xml or by using a properties file. If both methods are used, properties loaded from the properties.path file will override any init-params with the same name.

An Example web.xml File

The following example web.xml enables the Jespa HttpSecurityFilter component and uses the properties.path property to indicate that properties should also be loaded from the named file.

Note: The following web.xml and .prp files are also located in the examples/jespa/WEB-INF directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
    version="3.1">

    <display-name>Jespa Examples</display-name>

    <filter>
        <filter-name>HttpSecurityFilter</filter-name>
```

```
        <filter-class>jespa.http.HttpSecurityFilter</filter-class>
        <init-param>
            <param-name>properties.path</param-name>
            <param-value>WEB-INF/example.prp</param-value>
        </init-param>
    </filter>

    <filter-mapping>
        <filter-name>HttpSecurityFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

</web-app>
```

The following is an example of what the WEB-INF/example.prp file might contain:

```
# HttpSecurityService properties
http.parameter.username.name = username
http.parameter.password.name = password
http.parameter.logout.name = logout
#http.parameter.anonymous.name = anon
fallback.location = /jespa/Login.jsp
excludes = /Login.jsp
#groups.allowed = BUSICORP\\Example Users

# Require SPN and Channel Bindings (EPA)
bindings.targetSpn.policy = 2
bindings.cert.hash.policy = 2

# SecurityProvider properties
jespa.log.path = /tmp/jespa.log
jespa.log.level = 4
jespa.account.canonicalForm = 3

jespa.bindstr = dc100.busicorp.local
jespa.dns.servers = 192.168.44.110,192.168.22.115
jespa.dns.site = Paris
jespa.service.acctname = jespa1$@busicorp.local
jespa.service.password = cav-22^bim.33~kip+92$
```

The HttpSecurityService properties in the above properties file are described in the following sections. Any property prefixed with "jespa." will be passed through to the provider specified with the provider.classname property. Most of the provider properties shown above can be derived from the output of the Jespa SetupWizard.vbs script by simply prefixing each property with "jespa.".

Note: Modifications to the properties.path file do not require reloading the webapp. The file will be automatically reloaded within 5 seconds after being modified (although you might need to restart the browser being used to test the changes).

Requirements and Windows Settings for Single Sign-On (SSO)

To achieve the truly password-free experience provided by Windows built-in SSO, the following requirements and settings are required (otherwise the user will be presented with the network password dialog):

1. The target hostname must be added to the "Local intranet" security zone. The target hostname is the hostname portion of URL in the address bar of the browser. This is described further in the next section.
2. The user must be logged into the workstation using their AD DS credentials.

   Note: Of course this condition can only be satisfied if the OS is Windows. If the user is not logged into a domain or if the OS is Mac or Linux, the password dialog will be presented.
3. The browser must support Windows built-on SSO authentication. Most browsers including Edge, Chrome and Firefox running on a Windows OS just call into the Windows SSPI and therefore fully support Windows built-in SSO.

4. The URL used to visit the site may need to be a fully qualified DNS hostname or NetBIOS name. The special "localhost" name or an IP address[1] may not work as expected.

Internet Options and the Local Intranet Zone

For Windows clients to initiate SSO, the hostname or URL prefix of the target server will need to be added to the "Local intranet" zone settings of all clients. Search for and run `inetcpl.cpl or Internet Settings` to launch the Internet Options dialog. Select *Security* > *Local intranet* > *Sites* > *Advanced*. Add the target site (or a wildcard expression that matches the target site) to this list. Some examples of values for this list are:

`https://www.example.com`	Trust one specific site for SSO using HTTPS only (recommended form)
`www.example.com`	Trust the specific site using either HTTP or HTTPS.
`*.example.com`	Trust all sites under the example.com domain.

Note: In a large organization, these settings are usually applied using a global policy object setting (GPO). The exact procedure for this has changed with each major revision of Windows Server and therefore is not detailed here.

The Fallback Location

The HttpSecurityService can redirect clients to a "fallback location" if the browser does not support Windows built-on SSO or if the user clicks "Cancel" on the password dialog. To enable this feature, add the following property to the properties file or filter section but with a value referencing your login page:

```
fallback.location = /jespa/Login.jsp
```

IMPORTANT: The fallback location resource MUST ALSO be added to the excludes property (described in the next section). Although consider that excludes are referenced to the webapp base and not the resource path and therefore may be different (in the Jespa example webapp, the excludes property is excludes = /Login.jsp).

This value is ultimately used to set the window.location JavaScript property on the client (the value is not escaped).

Any application that will be used by an actual person should probably have this property set. If it is not set and and authentication fails for any reason, the client will receive a cryptic 401 Unauthorized or 403 Forbidden HTTP response.

The Excludes List

It is almost always necessary to exclude certain paths from protection by the HttpSecurityService. For example, it must not be necessary to authenticate a client trying to access a login form.

Caution: The paths listed in the excludes list will be completely unprotected.

The excludes property is a comma separated list of absolute paths *relative to the webapp root*.

Excludes may contain DOS-style wildcard expressions uisng * to match zero or more characters (and ? for one of any character). Note that the expression is applied to the entire path and not components of paths. See below for examples.

Consider the following example:

```
excludes = /account/login,/error.html,/images/*.jpg
```

In the above example, if a client tries to access the target `/account/login`, it will not be challenged to authenticate.

1 An IP address should work if the IP is added to the browser's "Local intrAnet" as though it were a hostname and the Subject Alternative Name of the TLS certificate has a matching IP address entry.

Note: Paths with literal commas must be quoted and paths. Quotes must be quoted with literal quotes escaped by an additional quote (same as CSV format). See the HttpSecurityService API documentation for details.

Note: Request paths do NOT include key value "QUERY_STRING" parameters. For example, a request for a path like /controller?action=login will NOT match an exclude like "*action=login*" (because this part of the resource is for parameters and are in fact not part of the path). To exclude a request based on QUERY_STRING parameters, it would be necessary to write code and extend the HttpSecurityService to override the isProtected method. See the jespa.http.HttpSecurityService API documentation for details.

Note: The excludes list will not stop the browser from pro-actively initiating authentication such as because it previously authenticated when accessing a different resource not in the excludes list. The excludes list simply stops the HttpSecurityService from *challenging* the client to perform authentication. This is why the url-pattern directive in the web.xml must be “/*”.

The following are some examples of exclude values (including wildcard expressions):

Exclude Value / Expression	Path Requested	Result Excluded?
`/css/style.css`	`/css/style.css`	true - the path matches exactly and therefore it is excluded from authentication
`/css/Style.css`	`/css/style.css`	false - capital S does not match and therefore the client will be challenged for auth
`/css/style.css0`	`/css/style.css`	false - zero at end does not match
`/css/*.css`	`/css/style.css`	true - matches wildcard expression
`/css/*.css`	`/prod/style.css`	false - prefix does not match
`/account/login`	`/account/login`	true - the path matches exactly
`/account/login`	`/account/Login`	false - the capitol L does not match
`/account/*`	`/account/abaker`	true - the wildcard matches
`/error*`	`/errors/40x.html`	true - the wildcard matches
`*.css`	`/css/style.css`	true - ends with .css
`*.css`	`/tmp/.css`	true - ends with .css (does not matter that previous character is path separator)
`style.css`	`/css/style.css`	false - prefix does not match
`*style.css`	`/css/style.css`	true - suffix matches
`*style.css`	`/css/astyle.css`	true - suffix still matches
`*/style.css`	`/css/astyle.css`	false - suffix does not match
`data?????.dat`	`/data12345.dat`	true - question marks mean one character matches
`data?????.dat`	`/data123.dat`	false - not five characters followed by .dat
`data*.dat`	`/data123.dat`	true - matches wildcard
`da?a.dat`	`/data.dat`	true - letter t matches question mark
`da??a.dat`	`/data.dat`	false - not two characters followed by suffix

SPN and Channel Bindings (EPA)

SPN and channel bindings, which Microsoft calls Extended Protection for Authentication (EPA), are security features that stop relay attacks and MITM attacks.

As of Jespa 2.0, the default policy for this feature is to allow clients that do not provide valid bindings. It is recommended that you fully enable this feature by performing two additional installation steps:

Installation Step 5: Change SPN and Channel Bindings Policy

The HttpSecurityService will automatically attempt to retrieve and use the server TLS certificate to generate SPNs and compute channel bindings. The first TLS request submitted after the server starts should generate log entries (with log.level >= 3) like:

```
EXAMPLE\bcarter successfully authenticated
Loading bindings certificate from URL: https://alma92as5.example.com:8443/jespa/
Computed bindings.cert.hash: 531B8FFCA96ED512AD94AEB3687658A3
Inserted SPNs from X509Certificate: bindings.targetSpns=[HTTP/alma92as5.example.com,
HTTP/alma92as5, HTTP/10.11.12.13:8443, HTTP/alma92as5.example.com:8443, HTTP/alma92as5:8443,
HTTP/10.11.12.13]
targetSpn submitted and matched: HTTP/alma92as5.example.com:8443
bindings.cert.hash submitted and matched
```

If all clients generate both “submitted and matched” log entries, this indicates that SPN and channel bindings checking is working and that you can safely set properties to require that clients provide valid SPN and channel bindings like:

```
# Require SPN and Channel Bindings (EPA)
bindings.targetSpn.policy = 2
bindings.cert.hash.policy = 2
```

Note: When proxying requests through another server like NGINX, the default SPNs will likely be incorrect because the port used by clients will not be the same as the port seen by the HSS. In this case, it will be necessary to explicitly add the correct SPN(s) using the bindings.targetSpns property.

Note: If you are not using TLS, you should still set the bindings.targetSpns* properties and fully enable SPN checking but you will need to explicitly set the list of SPNs.

Note: The curl program is known to not submit an SPN when using NTLM (although it does submit channel bindings).

Installation Step 6: Set Channel Bindings Keystore or PEM Properties

Automatically retrieving the TLS certificate is not the most secure option as an attacker could intercept this request during (re)initialization and defeat the channel bindings protection. The most secure option is to use the `bindings.cert.keystore.*` properties (or `bindings.cert.pem*` properties) to explicitly and safely provide access to the server certificate.

See *The HttpSecurityService Properties* section and HttpSecurityService API documentation for details.

HTTP Form Based Logins

The HttpSecurityService supports username and password HTTP request parameter-based logins for use with conventional HTML login forms.

Note: Setting any of the http.parameter.* properties will trigger the request input stream to be decoded which may not be compatible with certain types of requests like those of REST services. In these instances it will be necessary to write some code to directly invoke a suitable `SecurityProvider.authenticate()` method with the username and password supplied by the client.

Caution: HTTPS should always be used regardless of authentication mechanism but failure to use HTTPS with parameter-based logins is exceptionally insecure.

Note: Any login form resource must also be added to the excludes list.

To enabled parameter-based logins, simply add configuration properties (or web.xml init-params) such as the following:

```
http.parameter.username.name = acctname
http.parameter.password.name = password
```

The above properties will instruct the HttpSecurityService to check for "acctname" and "password" HTTP request parameters and then attempt to authenticate the user using those credentials.

Note: Once a client has authenticated successfully using explicit credentials, that identity will persist until the they logout (described below) or their session times out. This is true even if the client has previously performed SSO authentication or

subsequently initiates SSO authentication. If the client initiates SSO while they are logged in with explicit credentials, the SSO authentication will be accepted but the SSO identity will simply be discarded.

Logging Out

Once authenticated, whether it be through a login form, the browser password dialog or Windows built-in SSO, authentication state will be stored in the HTTP session (using the “jespa.provider.state“ HttpSession attribute key). If this state is removed, the client is effectively logged out. The HttpSecurityService includes a feature to remove this state based on a client supplied HTTP parameter. To enable this feature, simply add a property such as the following:

```
http.parameter.logout.name = logout
```

The above property (or web.xml init-param) will instruct the HttpSecurityService to check for the "logout" HTTP request parameter. If it is present (the actual value is ignored), the authentication state stored in the HTTP session will be removed thereby logging out the client. For example, with the above property set, a request like the following:

https://as1.busicorp.local/account/login?logout=1

would logout the client (and then presumably present the user with a login page).

Anonymous Access

The HttpSecurityService may be configured to allow clients to bypass authentication entirely and assume an "anonymous" identity.

CAUTION: This feature allows any client to bypass authentication. If this feature is enabled, the developer is completely responsible for controlling access to all content accessed through the HttpSecurityService using the getRemoteUser, getUserPrincipal, getAccount or isUserInRole methods which will all return `null` or `false` if the user is "anonymous".

This feature is useful when the site (or part of it) is intended to be open to the public or when the developer needs to disable authentication completely so that an alternative security mechanism can be used later in the request chain.

To enabled this feature, add the following property (or init-param):

```
http.parameter.anonymous.name = anon
```

The above init-param will instruct the HttpSecurityService to check for the "anon" HTTP request parameter. If it is present (the actual value is ignored), and the client has not already authenticated, the client will assume the special "anonymous" identity. For example, if the property is set to "anon", an HTTP request such as:

http://as1.busicorp.local/jespa/?anon=1

will bypass authentication and install the anonymous identity (unless the client was already authenticated).

Note: If you want to implement this without using an HTTP parameter, you will need to write code and override the `isAnonymous` method of the `HttpSecurityService`.

Even though an anonymous client has not authenticated, a special anonymous identity will be stored in the HTTP session. Meaning after a client becomes anonymous, they will remain anonymous until they logout (see the *Logging Out* section).

Windows Group Based Access Control

The HttpSecurityService supports group based access control. This functionality is exposed in two ways:

1. The `HttpServletRequest.isUserInRole` method may be used to determine if the current

authenticated user is in the named group.

2. The HttpSecurityService supports `groups.allowed` and `groups.denied` properties that control who can authenticate through the HttpSecurityService.

Note: There is no feature to control access to specific resources because only an application could define how resources are accessed. Trying to control access based on request path would likely lead to unexpected application specific security vulnerabilities.

Note: The NtlmSecurityProvider also allows user account names to be used in place of group names.

NtlmSecurityProvider Access Control

The NtlmSecurityProvider implements Windows security group based access control. This can be leveraged by the HttpSecurityService through the `groups.allowed` and `groups.denied` properties. Consider the following example:

```
groups.allowed = BUSICORP\\Engineers,BUSICORP\\FinChart Admins,bcarter@busicorp.local
```

In the above example, the `groups.allowed` property is a comma separated list of two qualified Windows group names and the account name of a specific individual. Only users within these groups, or the specifically named user will be permitted to authenticate with the HttpSecurityService and therefore permitted access to the content it protects.

If the `groups.allowed` property is not supplied, the default behavior is to permit access.

The `groups.denied` property is also a comma separated list of group names. If the current authenticated user is found to be in any one of these groups, the access check immediately stops, the `groups.allowed` check is not performed and the user will not be permitted access.

If a client is denied access they will either become anonymous, be presented with a login form (because a 401 Unauthorized response was sent), be redirected to the fallback location or receive a 403 Forbidden response. Precisely what happens depends on what other features of the HttpSecurityService are enabled.

Note: Windows group names should always be qualified with a domain like *BUSICORP\Engineers* to eliminate any possible ambiguous behavior in a multi-domain environment and to reduce the possibility of delays related to the domain controller failing to resolve a name.

Note: If a security group or account name cannot be resolved to it's SID, significant delays may occur. Check the log file (with a log.level >= 2) for "Failed to resolve name" error messages after making changes to your group access lists.

Note: Windows Domain Local Groups that are not in the same domain as the Jespa Computer account will not be in scope (meaning the user will not be considered in that group).

LdapSecurityProvider Access Control

When using the LdapSecurityProvider, groups are specified using DN or RDN strings. In this case, it is important to quote each group name. Otherwise, the commas in DN strings will be interpreted by the HttpSecurityService as group name separators. The following example illustrates how to use DN strings with `groups.allowed` and `groups.denied`.

```
groups.allowed = "CN=Lab Techs,DC=openbook,DC=edu", "CN=Postdocs,DC=openbook,DC=edu"
```

However, if the `bindstr` property includes a base DN, group names may be RDNs relative to that base. For example, the below example is equivalent to the above example.

```
jespa.bindstr = ldap://dc100.openbook.edu/DC=openbook,DC=edu
...
groups.allowed = "CN=Lab Techs", "CN=Postdocs"
```

Customizing the HttpSecurityService

The HttpSecurityService is specifically designed to be extended with new code. By overriding methods like `isProtected`, `isAnonymous`, `isLogin`, `getRequestCredential`, `isAllowedAccess` and others, the behavior of the HttpSecurityService can be heavily customized. See the jespa.http.HttpSecurityService API documentation for details.

CAUTION: If you decide to create a custom solution, be careful to route ALL requests *through* the HttpSecurityService `doFilter` method. Do not call `doFilter` in an on-demand convention where control is allowed to return before executing the primary function of the request. Trying to do this will almost certainly result in unexpected behavior. The HttpSecurityService handles concurrent authentication states, proactive POST re-authentication, suppressing redundant authentications and switching between explicit logins and SSO (among other things).

The NtlmSecurityProvider

The NtlmSecurityProvider provides an implementation of NTLMSSP to initiate and accept NTLMv2 authentication. The acceptor closely mimics Windows behavior using the NETLOGON service over Secure Channel with AES signing and sealing.

This provider implements much of jespa.security.SecurityProvider API including the token-based authentication used by the HttpSecurityService, HttpSecurityFilter, SaslServer, SaslClient and other components. The Account.isMemberOf method mimics Windows behavior using SID-based security group checking.

NtlmSecurityProvider Properties

The table below describes properties used by other components in this manual such as the HttpSecurityService. See the NtlmSecurityProvider API documentation for a complete list.

Note: When used with other Jespa components like the HttpSecurityService, HttpSecurityFilter, SaslServer and SaslClient, these property names will usually need to be prefixed with "jespa.".

Property	Description	Example
`bindstr`	The name of the AD DS domain against which credentials will be validated. This name is used with DNS SRV lookups to locate a suitable domain controller. The NtlmSecurityProvider will bind to the NETLOGON service of that server. **This parameter must be a fully qualified DNS domain name** or the fully qualified DNS hostname of a particular AD DS server. Trying to use an IP address or NetBIOS name for this property will result in an error. The specified domain or domain of the specified server must be the same as that of the Computer account identified by the service.acctname property. If this condition is not true, a "SAM database ... does not have a computer account" error will occur as described in the *Possible Issues: Issue 4* section. See the *DNS Requirements and Properties* section for details regarding how domain controllers are located.	`example.com`
`service.acctname`	The name of the Computer account created in Step 1 of the installation. This name must be the sAMAccountName and domain name separated by an @ sign. The sAMAccountName is the Computer account name with the $ sign at the end. The domain name component is the fully qualified DNS domain name that the Computer account is in. The Computer account must be in the same domain identified by the bindstr property. Note that this cannot be the name of an account used by an actual computer or Windows OS instance. The account must be	`jespa1$@example.com`

	created as described in Step 1 of the installation.	
service.password	The password corresponding to the service.acctname above. This password should be long and random. See Step 2 of the installation for instructions regarding setting this password. Note: Internally this password is encrypted to prevent it from being exposed such as within log files.	cav-22^bim.33~kip+92$
localhost.netbios.name	IMPORTANT: This property is deprecated. If you set this property at all (even to an empty string), you will likely receive the error `Logon failure: unknown user name or bad password` as described in Issue 10 in the Possible Issues section.	DEPRECATED
log.path	The absolute path to a log file used for debugging purposes. See the jespa.util.LogStream class for information regarding how to set a custom PrintStream (such as for interfacing with other Java logging components).	C:\temp\jespa.log
log.level	The level of information to be logged. Log level values are: 0 - nothing 1 - critical (this is the default) 2 - basic info can be logged under load 3 - almost everything n - debugging only	2
account.canonicalForm	A small integer indicating how account names should be canonicalized (the form returned by `NtlmSecurityProvider.getIdentity()`). Values of interest are: 2 - Username - A simple unqualified username like `abaker`. 3 - Backslash - The short NetBIOS domain and username separated by a backslash like `EXAMPLE\abaker` (sometimes referred to as the “down-level login name”). 4 - Principal - The username and DNS domain name separated by an ‘@’ character like `abaker@example.com`. Note: The NtlmSecurityProvider can authenticate clients by their userPrincipalName in which case it will be used if that canonicalForm is used. However, if a client authenticates using their sAMAccountName, a principal name will be constructed from it (the so-called “implicit UPN”) which may not match the userPrincipalName set on their account. Meaning, the canonical name could be different depending on what name the user supplies.	3
domain.trust.cache.values	Depending on the canonical form being used, Jespa may need to canonicalize a domain name (convert the NetBIOS name to the DNS name or vice versa). Unfortunately it is not uncommon for one or both names to be missing from trust information of foreign domains retrieved through the local NETLOGON service. This property can be used to fill-in the missing names and give Jespa a complete set of NetBIOS / DNS domain name mappings. The format of this property value is a comma separated list of colon separated domain name pairs in the form <netbios1>:<dns1>,<netbios2>:<dns2>,<netbios3>:<dns3> and so on.	ENG:eng.busicorp.local,ASDF:asdf.net,RSCH:research.busicorp.local

MSRPC TCP Transport Properties

The following properties are specific to MSRPC TCP transport communication which includes NETLOGON and lsarpc calls used by the NtlmSecurityProvider.

msrpc.tcp.laddr	The IP address of the local interface that Jespa should bind for MSRPC communication with domain controllers. This can be used to bind a network interface other than the default selected by Java.

msrpc.tcp.soTimeout	The SO_TIMEOUT value in milliseconds specifies how long Jespa will wait for a read call to return before throwing an exception. The default value is 60000 (meaning 1 minute).
msrpc.tcp.connTimeout	A value in milliseconds that specifies how long Jespa will wait for a new socket connection to be established. The default value is 60000 (meaning 1 minute).
msrpc.tcp.idleTimeout	A value in milliseconds that specifies how long an MSRPC TCP connection will be left open when there is no activity. The default value is 20000 or 20 seconds. A value of 0 disables the idle timeout functionality. If the network silently drops idle TCP connections, setting this to 0 may, albeit very unlikely, result in “Read timed out” exceptions.
msrpc.useNamedPipe	If set to “true”, Jespa will use SMB1 named pipes for communication with domain controllers. Note: SMB1 is deprecated and disabled by default in most Windows environments. The default value is “false”. Note: Setting this property to true requires that the JCIFS library from https://www.jcifs.org/ is in the application classpath.

Note: When setting these properties in the HttpSecurityService properties file, they must be prefixed with “jespa.” like “jespa.msrpc.tcp.laddr = 10.120...”.

The LdapSecurityProvider

LdapSecurityProvider provides an easy to use interface for developers to create, update, delete and search accounts, groups or other directory entries, check group membership and set or change passwords.

However, because this manual targets operator's and not developers, the descriptions of properties below are mostly specific to the authentication and authorization functionality of the HttpSecurityService. Developer's should refer to the LdapSecurityProvider API documentation for a complete description of this provider.

LdapSecurityProvider Properties

The table below describes some LdapSecurityProvider properties that are important to the HttpSecurityService (and HttpSecurityFilter). For a complete technical description of all properties supported by the LdapSecurityProvider, see the API documentation.

Note: To authenticate Active Directory clients, the NtlmSecurityProvider is superior. It has superior performance and LDAP cannot be used to implement SSO. However, to authenticate HTTP clients against accounts in a non-Active Directory LDAP server such as OpenLDAP, the LdapSecurityProvider may be used.

Note: When setting these properties in the HttpSecurityService properties file, they must be prefixed with “jespa.” like “jespa.ldap.disposition = RFC”.

Property	Description	Example
ldap.disposition	If this value starts with "RFC" as opposed to the default "ADS", the behavior of the LdapSecurityProvider changes significantly. See the LdapSecurityProvider API documentation for details.	RFC2251
bindstr	An RFC 2255 style LDAP URL identifying the authority to which this LdapSecurityProvider will be bound. If the optional base DN is used, some functions will accept an RDN relative to this base (such as the isUserInRole method used within Servlets and JSPs).	ldap://ldap1.openbook.edu/ OU=Engineering,DC=openbook,DC=edu
service.acctname	The username used for authentication. For an RFC based LDAP server this value must be a full DN string. For an Active Directory server, this name	ldapuser15@openbook.edu

	may be a principal name or DN.	
service.password	The password corresponding to the service.acctname.	moonbike69
ldap.authentication.setcredential	If set to true, the LdapSecurityProvider will store the authenticated user's credential (in encrypted form) for use with subsequent operations. Meaning, when used with the HttpSecurityService, this enables impersonation. The default is false to indicate that the service.acctname and service.password properties should always be used.	true
domain.netbios.name	A NetBIOS domain name used for account name canonicalization. See the acctname.canonicalForm property. This is only required if users must be able to supply account names in down-level login name form (like BUSICORP\bcarter).	OPENBOOK
domain.dns.name	A DNS domain name used for account name canonicalization. See the acctname.canonicalForm property. This is only required if users must be able to supply account names in principal form (like bcarter@busicorp.local).	openbook.edu
authority.dns.names.resolve	If this property is set to false, the LdapSecurityProvider will not use DNS SRV lookups to locate domain controllers. For non-Active Directory LDAP servers, it will frequently be necessary to set this property to false. But for proper redundancy and failover, DNS SRV records should be created so that this property can be true.	false
account.canonicalForm	A small integer that controls the format of account names returned by getRemoteUser and other methods that retrieve the identity of the an authenticated user. This value should be one of: 2 - Username - A simple unqualified username like bcarter, 3 - Backslash - A backslash style name like BUSICORP\bcarter or 4 - Principal - A principal style name like bcarter@busicorp.local.	3
log.path	The absolute path to a log file used for debugging purposes. See the jespa.util.LogStream class for details.	C:\temp\jespa.log
log.level	The level of information to be logged. Log level values are: 0 - nothing 1 - critical (this is the default) 2 - basic info can be logged under load 3 - almost everything n - debugging	2

See the example webapp for several example configurations that illustrate how to use the LdapSecurityProvider.

The ChainSecurityProvider

The ChainSecurityProvider is a wrapper around a "chain" of SecurityProviders. The ChainSecurityProvider is most commonly used with the HttpSecurityService to authenticate web clients against multiple independent

security authorities.

> Note: It is not necessary to use the ChainSecurityProvider to authenticate clients against multiple Active Directory domains with the NtlmSecurityProvider if the domains have trust relationships. The NtlmSecurityProvider fully supports cross domain authentication by itself.

When the authenticate method is called (such as by the HttpSecurityService), the ChainSecurityProvider iterates through each SecurityProvider in the "chain" until authentication is successful. All other methods simply call the corresponding method on the currently selected SecurityProvider.

> Note: Only the first SecurityProvider in a chain can perform SSO authentication. To disable SSO in the NtlmSecurityProvider (so that it can be a secondary chain element), set flags.capabilities.accept.ntlmssp = false.

A chain is defined entirely using properties. The `chain.names` property defines the list of chain element names. For each chain element name, the property `chain.<name>.provider.classname` indicates to ChainSecurityProvider the name of the SecurityProvider class to construct for that chain element. For all other properties that begin with `chain.<name>.`, that prefix is removed and the property is inserted into the `Map` used to construct the SecurityProvider for that chain element. Consider the following list of properties:

```
chain.names = BUSICORP,OPENBOOK

chain.BUSICORP.provider.classname = jespa.ntlm.NtlmSecurityProvider
chain.BUSICORP.bindstr = dc100.busicorp.local
chain.dns.servers = 192.168.44.110,192.168.22.115
chain.dns.site = Paris
chain.BUSICORP.service.acctname = jespa1$@busicorp.local
chain.BUSICORP.service.password = cav-22^bim.33~kip
chain.BUSICORP.account.canonicalForm = 3

chain.OPENBOOK.provider.classname = jespa.ldap.LdapSecurityProvider
chain.OPENBOOK.ldap.disposition = RFC
chain.OPENBOOK.domain.netbios.name = OPENBOOK
chain.OPENBOOK.domain.dns.name = openbook.edu
chain.OPENBOOK.bindstr = ldap://192.168.2.119/OU=Engineering,DC=openbook,DC=edu
chain.OPENBOOK.service.acctname = CN=jespa1,DC=openbook,DC=edu
chain.OPENBOOK.service.password = opensaysme44
chain.OPENBOOK.account.canonicalForm = 3
```

The above example shows a chain of two SecurityProviders called BUSICORP and OPENBOOK.

> Note: Chain element names are completely arbitrary although they should not contain characters like equals (=) or brackets (<) which might conflict with configuration file syntax.

In the above example, the BUSICORP chain uses the NtlmSecurityProvider whereas the OPENBOOK chain uses the LdapSecurityProvider configured for an RFC-based LDAP server like OpenLDAP. Using this configuration, the client can authentication with Active Directory using either SSO or explicit credentials or with the RFC-based LDAP server using explicit credentials.

When authenticating clients against multiple independent authorities using the ChainSecurityProvider, it is important to supply domain properties and to use a qualified canonicalForm (3 for backslash form or 4 for principal form). Otherwise, it may not be possible to correctly process account names that exist in both authorities. Displaying names in a qualified form like BUSICORP\cdavis or cdavis@busicorp.local as opposed to just cdavis will also reduce user confusion about which credentials are being used.

Group name syntax is different depending in the SecurityProvider being used. For example, the NtlmSecurityProvider supports names in a form such as "BUSICORP\Wiki Users" whereas the LdapSecurityProvder supports only DN or RDN group names such as "CN=Wiki Users,OU=Engineering,DC=openbook,DC=edu" or just "CN=Wiki Users" if a suitable base is supplied in the bindstr. The appropriate form should be used for each SecurityProvider.

The HTTP Client

The Jespa library includes an HTTP 1.1 client with support for NTLMv2, SPN and channel binding, and other features that might be important in Windows environments.

Note: The Jespa HTTP client does not currently support proxies.

Currently this client is exposed only through the HttpURLConnection API. The following code fragment demonstrates how to use the Jespa HTTP client to perform a simple GET request:

```
HttpURLConnection conn = new HttpURLConnection(new URL(urlstr));

InputStream input = conn.getInputStream();
byte[] buf = new byte[64];
int n;

while ((n = input.read(buf, 0, buf.length)) > 0) {
    System.out.write(buf, 0, n);
}
input.close()
```

See the examples directory for numerous other examples that use NTLM credentials, POST requests and more.

The client is also exposed as an HTTP `java.net.URLStreamHandler` for use with `java.net.URL`.

Note: IOPLEX believes that the URLStreamHandler mechanism is clumsy and that it should not be used.

To install the Jespa HTTP client for use with `java.net.URL`, add the "jespa" package prefix to the `java.protocol.handler.pkgs` system property as illustrated by the following code:

```
String pkgs = System.getProperty("java.protocol.handler.pkgs");
if (pkgs == null) {
    pkgs = "jespa";
} else {
    pkgs += "|jespa";
}
System.setProperty("java.protocol.handler.pkgs", pkgs);
```

This should be performed very early in your program such as in your main method. Otherwise, it may also be suitable to specify it on the command-line like:

```
java -Djava.protocol.handler.pkgs=jespa MyHttpClientProgram
```

Using NTLM Authentication with the HTTP Client

The Jespa HTTP client supports NTLM authentication (including all variations of NTLMv2) and leverages the JAAS subject based security model to access credentials. More specifically, the client will initiate NTLM authentication if the server requires it and a `jespa.security.PasswordCredential` object is present in the current Thread's Subject. The JAAS and `javax.security.auth.Subject` documentation details how exactly to perform this bootstrapping procedure but the `examples/` directory contains several examples that use a variety of credential bootstrapping methods. In particular look at how `examples/HttpGet.java` uses the `jespa.security.RunAs` utility class.

To initiate NTLM authentication either the credential account name must include the AD DS DNS domain name or the `jespa.domain.dns.name` System property (or `http.auth.ntlm.domain` for compatibility with the default Java HTTP client) must be set to the AD DS DNS domain.

Note: The short NetBIOS domain name will not work with these properties.

The SASL Client and Server

The Jespa library provides standard javax.security.sasl SaslClient and SaslServer implementations. Also included are `SaslClientFactory` and `SaslServerFactory` classes that adds NTLM authentication, integrity and confidentiality to the stock LDAP client used by JNDI (see the *Using NTLM Security with JNDI / LDAP* section below).

The following line of code installs the `SaslClientFactory` and `SaslServerFactory` classes through a JCA provider.

```
java.security.Security.addProvider(new jespa.security.JCAProvider());
```

Note: The Jespa JCA provider is used only to install the SaslClientFactory and SaslServerFactory. Jespa does not currently implement any other JCA operations. JCA providers should not be confused with the Jespa SecurityProvider interface. The two "provider" interfaces are largely unrelated.

The SaslClient and SaslServer may also be used directly. See the API documentation for details.

Like the Jespa HTTP client, the Jespa SASL client also leverages the JAAS subject based security model to retrieve the credential that should be used. More specifically, the client will initiate NTLM authentication if a `jespa.security.PasswordCredential` object is present in the current Thread's `Subject`. The JAAS and `javax.security.auth.Subject` documentation details how exactly to perform this bootstrapping procedure, but the `examples/` directory contains several example programs that use a variety of credential bootstrapping methods. In particular, look at how `jespa.util.RunAs` is used within `examples/SaslTest.java`.

Using NTLM Security with JNDI / LDAP

Provided that the `SaslClientFactory` has been installed and the calling thread's `Subject` has the necessary `PasswordCredential`, the final step is to specify a JNDI `Context.SECURITY_AUTHENTICATION` property of "GSS-SPNEGO" when building a `DirContext` as illustrated by the following example:

```
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL, url);
env.put(Context.SECURITY_AUTHENTICATION, "GSS-SPNEGO");
DirContext ctx = new InitialDirContext(env);
...
```

Jespa properties that are prefixed with "jespa." will be passed to the NtlmSecurityProvider through the Hashtable parameter (or the SaslClient and SaslServer Map parameters). For example, instead of `service.acctname` the property name would be `jespa.service.acctname`.

Note: Jespa does not (currently) implement SPNEGO. But Windows advertises and uses the "GSS-SPNEGO" SASL mechanism name when in fact raw NTLMSSP is used.

Note: Jespa also includes an advanced LDAP API. See the jespa.ldap.* API documentation for details.

The LoginModule

Jespa provides a javax.security.auth.spi.LoginModule implementation that supports NTLM authentication.

See the jespa.security.LoginModule API documentation for details.

Possible Issues

This section describes possible issues you may encounter using Jespa and how to fix them. If you encounter an issue that you think should be listed here, please contact *support@ioplex.com* and tell us about it.

Issue 1: The "jespa.http.HttpException: 401 Unauthorized" exception

If the Jespa HTTP client will not perform NTLM authentication, make sure that the Active Directory DNS domain name is supplied through either a) the credential account name (such as alice@example.com) or b) the System property `jespa.domain.dns.domain` (or `http.auth.ntlm.domain`). The short NetBIOS domain name will not work.

Issue 2: The "Not a Type 1 Message" exception

NTLM over HTTP cannot traverse proxies, load balancers, connectors or similar without help. If you are proxying, forwarding or load balancing in front of Jespa, see the *Proxying and Load Balancing with NGINX* section.

The NTLM HTTP authentication protocol is a multi-request "handshake" like this example GET sequence:

```
C: GET /jespa/Whoami.jsp
S: 401 Unauthorized
   WWW-Authenticate: NTLM

C: GET /jespa/Whoami.jsp
   Authorization: NTLM TlRMTVNTUAABAAAAB4IIAAAAAAAAAAAAAAAAAAAAAAA=
S: 401 Unauthorized
   WWW-Authenticate: NTLM TlRMTVNTUAACAAAAAAAAA...AC4AZgBvAG8ALgBuAGUAdAAAAAAA

C: GET /jespa/Whoami.jsp
   Authorization: NTLM TlRMTVNTUAADAAAAGAAYAFgA...AAAYABgAcAwATPGDh0OrwyZcJ9CEoJw==
S: 200 OK
```

This protocol requires a persistent and unique TCP connection. Meaning Keep-Alive behavior is required. If the connection is closed / broken during the “handshake”, authentication will fail with “Not a Type 1 Message” or “Not a Type 3 Message”.

Verify that the Java app server has Keep-Alive enabled.

If there is a proxy between clients and Jespa, the only remedy would be to adjust the proxy configuration. See the *Proxying and Load Balancing with NGINX* section.

A less common cause of this error are internal redirects. If a request is internally redirected back through the HttpSecurityService with the `WWW-Authenticate` header present, the HttpSecurityService will attempt to process the header token again.

Clients Other than Browsers

If you are using a client other than a browser (like a Java program or PowerShell script), a possible cause for this exception is failure to maintain session state on the server. In particular, make sure the client is using cookies.

For example, when using the C# HttpWebRequest API, you must set a CookieContainer as illustrated by the following example code fragment:

```
HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
request.CookieContainer = new CookieContainer();
...
```

When using PowerShell, you will need to use -SessionVariable and possibly -WebSession arguments as

illustrated by the following code fragment:

```
$acctname = "theacctname@busicorp.local"
$password = "The_paS$_W0rd"
$secpass = ConvertTo-SecureString $password -AsPlainText -Force
$cred = New-Object System.Management.Automation.PSCredential($acctname, $secpass)
$result = Invoke-WebRequest -Credential $cred -SessionVariable session -UseBasicParsing $url
... subsequent requests using the $session ...
$result = Invoke-WebRequest -Credential $cred -WebSession $session -UseBasicParsing $url
```

When using VBScript with the WinHttpRequest API, the cookie header must be saved to a variable as illustrated by the following code fragment:

```
Dim url, cookie

url = "https://www.example.com/jespa/secure/Whoami.jsp"
cookie = null

Function GetURLText(url)
    Dim http, hdrs

    Set http = CreateObject("WinHttp.WinHttpRequest.5.1")
    http.Open "GET", url, False
    http.SetAutoLogonPolicy 0

    If (IsNull(cookie) = False) Then
        http.SetRequestHeader "Cookie", cookie
    End If
    http.Send

    hdrs = http.GetAllResponseHeaders()
    If (InStr(hdrs, "Set-Cookie: ")) Then
        cookie = http.GetResponseHeader("Set-Cookie")
    End If

    GetURLText = http.ResponseText
End Function
```

Issue 3: The browser will not perform automatic authentication (SSO)

See the *Requirements and Windows Settings for Single Sign-On (SSO)* section.

Issue 4: The "SAM database ... does not have a computer account" exception

If you receive the following error:

> "The SAM database on the Windows NT Server does not have a computer account for this workstation trust relationship."

this indicates that the domain controller was located using the bindstr property but could not find the Computer account identified by the service.acctname property. There are several possible causes for this error which can be unexpected and include the following:

1. Make sure that the Computer account is in the domain identified by the bindstr property. For example, if the Computer account is jespa1$@eu.example.com but the bindstr property is example.com and eu.example.com is a sub-domain of example.com, you may receive this error. To fix this issue you need to either change the bindstr to match the Computer account domain or recreate the Computer account in the correct domain.
2. If you have just created the Computer account it may require some time to replicate to the domain controller selected by Jespa. To resolve this issue, adjust the bindstr property to temporarily specify the fully qualified DNS hostname of the specific domain controller on which the account is known to exist or wait for directory replication to complete.

3. Check the service.acctname format. The service.acctname is the sAMAccountName (which for a Computer account always ends with a $ sign) followed by an @ sign followed by the account's DNS domain like jespa1$@example.com.
4. Make sure that you created a Computer account and not a User account. Only a Computer account can bind the NETLOGON service.

Issue 5: The "format of the specified computer name is invalid" exception

This error may occur if the bindstr parameter is not a fully qualified DNS domain name or fully qualified DNS hostname. For example, this error may occur if the operator tries to use an IP address for the bindstr property. This error can also occur if authority.dns.names.resolve = false is set and the bindstr is not a fully qualified DNS hostname.

This error can also occur if you are using a DNS records file with invalid record data. For example, if the hostname of an SRV record does not correspond to the true hostname of a real server, this error can occur. See The DNS Records File section for details.

Issue 6: Windows SID based access control does not work as expected

There are a few conditions that can lead to unexpected behavior when using Windows group access checks:

1. Domain Local groups not in the Computer account domain will not be in scope. There are four different types of security groups: Universal, Global, Domain Local and Builtin. When a user authenticates using NTLM, the domain controller supplies a fully expanded list of group SIDs for all Universal and Global groups but only Domain Local groups from the domain of the Jespa Computer account used. Domain Local groups of a user in a different domain will not be included. This issue is not unique to Jespa - this is how Windows access control has always worked[2].
2. If group membership has been changed, the user must logout of their workstation and back in for the changes to take effect.
3. Group names should be qualified with a domain name like EXAMPLE\Engineers and not just Engineers. Check the Jespa log file for error messages like "Failed to resolve name: EXAMPLE\Engineers".
4. Backslashes in properties files must be escaped with an additional slash like EXAMPLE\\Engineers.

Issue 7: The "page cannot be displayed" error using the HttpSecurityService

There could be numerous causes of this error. However, one possible cause is that the client is not configured to perform Windows built-in authentication whereas we server security policy requires it.

Issue 8: The "Failed to locate authority for name: EXAMPLE" error

This error most likely indicates that you need to set the dns.servers property. See the DNS Properties section and review your DNS configuration in general.

Issue 9: POST data is not submitted when using the HttpSecurityService

This condition was common when using Internet Explorer which would not submit POST data until after the NTLMSSP handshake completed and it had previously authenticated with the target. The only way to stop this behavior was to either close IE and restart it or set a DisableNTLMPreAuth registry value as described in Microsoft KB251404. However, because IE is no longer supported by Microsoft and generally not used, this should no longer be an issue.

Another possible cause of this condition is using the http.parameter.* properties which will cause the app server to decode the request input stream. Some request types like REST services or certain application servers like ColdFusion may exhibit this behavior. If this is applicable, disable the http.parameter.* properties so that the input stream is not prematurely decoded.

2 Note that if the authentication mechanism were Kerberos and not NTLM, groups would be compiled as Kerberos tickets transit the trust relationships between domains and therefore Domain Local groups of the user's domain would be in scope instead.

Issue 10: The "Login failure: unknown user name or bad password" exception

This error almost certainly means exactly what it says - the account does not exist or the supplied password is simply incorrect.

One likely cause of this error is old credentials stored in a browser credential manager or keychain. Clear any saved passwords for the target site and try again.

> Note: AD has a “minimum password age” policy that does not permit a password to be changed within some time period which is most commonly 1 day. The operator will not receive an error when trying to change a password within this period. If you have reset the Jespa Computer account password more than once, it could be that the change was not effective.

> Note: Computer account passwords do NOT expire. The “Domain member: Maximum machine account password age” policy instructs domain members to periodically submit a password change but they are NOT required to. The Jespa Computer account will not stop functioning after the policy period.

This error can also occur if you set the localhost.netbios.name property. This property is deprecated and must not be set.

This error can also occur if you set the domain.netbios.name property on the NtlmSecurityProvider. The domain.netbios.name and domain.dns.name properties of the NtlmSecurityProvider are read-only and must not be set.

Issue 11: The "account used is a Computer Account" exception

This error indicates that there is a problem with the Computer account used by Jespa. The most efficient way to resolve this issue is to simply create a new Computer account as described in Step 1 of the Installation section and then delete the old account.

Issue 12: The "NetrLogonSamLogon return authenticator check failed" exception

This error occurs when the same Computer account is used to communicate with the NETLOGON service from two different Jespa instances. See the *Validating NTLM Credentials with the NETLOGON Service* for additional information.

```
SecurityProviderException: NetrLogonSamLogon return authenticator check failed
    at jespa.ntlm.Netlogon.validate0(Netlogon.java:191)
    at jespa.ntlm.Netlogon.validate(Netlogon.java:229)
    at jespa.ntlm.NtlmSecurityProvider.authenticate(NtlmSecurityProvider.java:417)
    ...
```

To correct this issue, you must create a separate Computer account for each Jespa instance.

Issue 13: The “java.lang.NoClassDefFoundError: jcifs/smb/...” exception

UPDATE: JCIFS is no longer required as of Jespa 1.2.5. Jespa now uses raw TCP transport for MSRPC by default. JCIFS is only required if you want to use the older SMB1 named pipe transport.

This error will occur if you have set msrpc.useNamedPipe = true but the JCIFS library is not in your classpath. Download and install the JCIFS library from https://www.jcifs.org.

Issue 14: The “jespa.util.NtException: Access is denied.” exception

If you receive the following exception in the log:

```
Caused by: jespa.util.NtException: Access is denied.
     at jespa.ntlm.Netlogon.connect(Netlogon.java:372)
```

The most likely explanation is that your Computer account password is simply incorrect.

Note: AD has a “minimum password age” policy that does not permit a password to be changed within some time period which is most commonly 1 day. The operator will not receive an error when trying to change a password within this period. If you have reset the Jespa Computer account password more than once, it could be that the change was not effective.

Note: Computer account passwords are NOT required to meet password complexity requirements policy. Meaning if you receive an “Access id denied” error using a Computer account, password complexity is not the cause of this error.

Note: Computer account passwords do NOT expire. The “Domain member: Maximum machine account password age” policy instructs domain members to periodically submit a password change but they are NOT required to. The Jespa Computer account will not stop functioning after the policy period.

Note: It is not uncommon for the operator to insist that the password is correct only to find out later that resetting it or deleting the Computer account and running SetupWizard.vbs again resolved the issue. Watch carefully for errors when running scripts and PowerShell commands.

Issue 15: The “jespa.util.NtException: 0xC0000418” exception

This error will occur if NTLM is restricted by domain policy:

```
jespa.util.NtException: 0xC0000418
  at jespa.ntlm.Netlogon.validate0(Netlogon.java:706)
  at jespa.ntlm.Netlogon.validate(Netlogon.java:793)
  at jespa.ntlm.NtlmSecurityProvider.authenticate(NtlmSecurityProvider.java:1407)
```

This NT status code means `STATUS_NTLM_BLOCKED "The authentication failed because NTLM was blocked."`

To resolve this issue, you must add each Jespa computer account to the following group policy setting:

```
Computer Configuration > Policies > Windows Settings > Security Settings > Local Policies >
Security Options
```

Enable the following setting:

```
Network security: Restrict NTLM: Add server exceptions in this domain
```

and add the name of each Jespa computer account like "jespa1" (without the $ sign). If you use the same prefix for all of your Jespa accounts, you may also use a wildcard like "jespa*" to add an exception for all Jespa computer accounts and avoid adjusting security policy in the future.

Issue 16: SPN BINDINGS FAILURE and CHANNEL BINDINGS FAILURE Errors

These errors indicate that the SPN and / or channel bindings security checks are failing.

```
No match for targetSpn submitted: HTTP/alma92as5.example.com
SPN BINDINGS FAILURE: SPN did not match bindings.targetSpns, SPN not submitted or
bindings.targetSpns was not successfully initialized.
No match for bindings.cert.hash submitted: F230EC6440516D4519DD119C3A59C7B6
CHANNEL BINDINGS FAILURE: channel bindings did not match bindings.cert.hash, channel bindings not
submitted or bindings.cert.hash was not successfully initialized.
```

Aside from actual attacks, the most likely cause of these errors is missing or incorrect binding properties. More specifically, these errors will occur if you are forwarding requests through another server like NGINX.

Note: You can simply disable these security checks and the errors produced by setting the properties:

```
bindings.targetSpn.policy = 0
bindings.cert.hash.policy = 0
```

However, the SPN failure is easy to fix. Simply search the log file (with log.level >= 2) for “No match for targetSpn” and set the bindings.targetSpns property to the SPN found in the log:

```
bindings.targetSpns = HTTP/alma92as5.example.com
```

An easy way to fix the channel bindings failure is to search the log file (with log.level >= 2) for “No match for bindings.cert.hash” and set the bindings.cert.hash property to the hash found in the log:

```
bindings.cert.hash = F230EC6440516D4519DD119C3A59C7B6
```

Note: This hash will need to be updated when the certificate is updated (unless you use the bindings.cert.url property instead).

While these methods are quick and easy, they could be incorrect (the values in the log could have been submitted by an attacker). For maximum security, carefully review the SPN and Channel Bindings (EPA) section.

Example Code

All source code included with the Jespa package may be modified, used and distributed freely in accordance with the Copyright statement at the top of each source file. These Copyright statements are an exception to the standard Jespa EULA (see the LICENSE.txt file) which states that Jespa may not be redistributed in any form without written permission from IOPLEX Software.

The MyHttpSecurityFilter Example

The jespa.examples.MyHttpSecurityFilter class illustrates how to create a custom HTTP security filter that demonstrates:

- How to extend the HttpSecurityService directly. Note that the HttpSecurityService does not implement the Filter interface. So, unlike this example, you can integrate the HttpSecurityService into non-filter based solutions and custom Servlet containers. See the jespa.Http.HttpSecurityService API documentation for details.
- How to protect the service.password by encrypting it so that it does not appear as plaintext in the configuration.
- How to disable the filter using a simple boolean property.
- How to use an inner class as a "dummy" FilterChain to perform work *after* the request has passed through the HttpSecurityService (as opposed to using a separate Filter later in the chain). This can be used to retrieve the SecurityProvider and Account associated with the authentication.

All of the init-params are the same as the HttpSecurityFilter with the exception of the following:

Name	Description	Example
jespa.log.path	The path to a log file to which all Jespa logging messages should be written. This example uses a log4j DailyRollingFileAppender to create a new log file every hour.	/tmp/jespa.log
my.disabled	If set to "true", the filter will be completely disabled. Specifically, the doFilter method will just call the next doFilter method in the chain	true

	without performing any security checks or even fully initializing HttpSecurityService. The default behavior is to not disable the filter. Note: If this property is set in the properties.path file, once it is set to true, the Filter cannot be un-disabled without reloading the webapp because the properties file is ultimately processed in HttpSecurityService.doFilter.	
my.service.password.encrypted	The encrypted form of the service.passsword. The procedure for determining the encrypted password is described in the next section. If this value is not supplied, the usual unencrypted service.password is required.	qoDELaPeSvpZJVHWuaX

Setting the MyHttpSecurityFilter Encrypted Password

The MyHttpSecurityFilter example allows the service.password to be supplied in an encrypted form to prevent operators from easily viewing the plaintext Computer account password. However, to determine the encrypted form of the plaintext password, you must perform the following procedure:

1. Temporarily specify the desired password as the plaintext service.password property. Then also set the my.service.password.encrypted property to anything (such as "xyz") and set the jespa.log.level = 3.
2. Initialize the filter. When the filter detects both password properties are set, it will encrypt the service.password value and write it to the log file.
3. View the log file and locate the entry that looks like the following:

   ```
   2022-04-06 12:25:06: my.service.password.encrypted = qoDELaPeSvpZJVHWuaX
   ```

   Now set the indicated value as the my.service.password.encrypted property, delete the service.password property, restore the jespa.log.level value and restart the webapp.

The LdapSearch Utility

The LdapSearch utility may be used to query LDAP servers like AD DS servers.

The LdapSearch commandline syntax is defined as follows:

```
Usage: LdapSearch [-f <propfile>] [-v <level>] [-d ADS|RFC] [-x] [-t] [-a <authtype>] [-u
<username> -p <password>] ldap://host/base?attrs?scope?filter
```

The following table describes each option in detail:

Parameter	Description
`-f <propfile>`	Specifies a properties file. See the LdapSecurityProvider API documentation for a detailed table of possible properties. In practice, the most likely properties to be used with this utility class are the `dns.servers` and `dns.site` properties which in fact should be specified so that the implementation can properly locate a suitable LDAP server given only a domain name in the LDAP URL. Other properties that might be used are `ldap.disposition`, `service.acctname` and `service.password` (although these can also be specified on the commandline using the `-d`, `-u` and `-p` parameters) and perhaps `ldap.search.maxcount` and others.
`-v <level>`	Indicates the log level for "verbose" output. An ideal value for debugging usage issues is 4.
`-d ADS|RFC`	Specifies the "disposition" of the target server. With respect to this utility, this parameter controls attributes definitions and authentication behavior. The default value is "ADS" but if the server is not Active Directory, a value that starts with "RFC" will need to be specified (see examples below).
`-x`	Disable integrity and confidentiality. This is useful for debugging network communication such as with packet capture software. This options should otherwise probably not be used. Note that even if this option is not used, the server may choose to not negotiate confidentiality or integrity.
`-t`	Use TLS confidentiality. This option requires a trust store containing the PKI certificate exported from the LDAP server. The trust store file can be specified using the a commandline parameter like `-Djavax.net.ssl.trustStore=dc1.keystore`.
`-a <authtype>`	Specifies the JNDI `Context.SECURITY_AUTHENTICATION` such as "simple", "GSSAPI", etc. The default authentication type depends on the LDAP disposition of the server (specified with the `-d` option or `ldap.disposition` property). If the disposition starts with "ADS", the default authentication behavior is to use the Jespa NTLM and SaslClient infrastructure which provides NTLMv2 authentication and 128 bit session security. If the disposition starts with "RFC", the default authentication type is "simple". In this case, the plaintext password and communication should be secured with TLS confidentiality using the `-t` option (although it may not be required by the server).
`-u <username>`	The username used to authenticate with the target server. If this parameter is not specified, the `service.acctname` property from the properties file will be used.
`-p <password>`	The password corresponding to the above username used to authenticate with the target server. If this parameter is not specified, the `service.password` property from a the properties file will be used.

For a detailed description of RFC2255 style LDAP URLs with many examples, see the LdapSecurityProvider API documentation.

LdapSearch Command Examples

The following trivial example illustrates how to query the RootDSE properties of an Active Directory server.

```
C:\>java -cp jespa-2.0.0.jar jespa.ldap.LdapSearch 'ldap://192.168.2.110/'
```

For simplicity, an IP address is used in the LDAP URL. However, ideally a domain name or possibly a hostname should be supplied instead. A properties file should also be supplied (using the `-f` parameter) with DNS properties so that a suitable server can be properly located at runtime. Because this query is for the RootDSE and because no credentials were supplied, this example uses an anonymous bind.

The following example illustrates how to query a User account in Active Directory.

Note: This command must be typed on a single line.

```
C:\>java -cp jespa-2.0.0.jar jespa.ldap.LdapSearch -f busicorp.prp
   'ldap://busicorp.local/DefaultNamingContext??sub?(&(objectCategory=person)
   (sAMAccountName=hmuller))'

CN=Hans Müller,CN=Users,DC=busicorp,DC=local:
  displayName: Hans Müller
  givenName: Hans
  sAMAccountType: 805306368
  mobile: 9083773378
  primaryGroupID: 513
  objectClass: [top,person,organizationalPerson,user]
  badPasswordTime: 2018-12-16 12:24:22
  objectCategory: CN=Person,CN=Schema,CN=Configuration,DC=busicorp,DC=local
  cn: Hans Müller
  userAccountControl: 590336
  userPrincipalName: hmuller@busicorp.local
  dSCorePropagationData: [2019-09-17 13:47:25,2017-05-10 22:42:52,1600-12-31 19:04:17]
  codePage: 0
  distinguishedName: CN=Hans Müller,CN=Users,DC=busicorp,DC=local
  whenChanged: 2019-04-19 19:37:21
  whenCreated: 2017-05-02 13:51:19
  pwdLastSet: 2019-12-11 12:11:17
  logonCount: 309
  accountExpires:
  lastLogoff: 1600-12-31 19:00:00
  lastLogonTimestamp: 2019-04-19 19:37:21
  objectGUID: bWYA2RAjQUeXz34GLCvGTg==
  sn: Müller
  lastLogon: 2019-09-21 13:27:16
  uSNChanged: 2896697
  uSNCreated: 15617
  objectSid: S-1-5-21-2799971297-2625803212-4394051119-1431
  countryCode: 0
  sAMAccountName: hmuller
  instanceType: 4
  memberOf:
["CN=Engineers,CN=Users,DC=busicorp,DC=local","CN=Group598,CN=Users,DC=busicorp,DC=local",...,"CN
=Remote Desktop Users,CN=Builtin,DC=busicorp,DC=local"]
  badPwdCount: 0
  name: Hans Müller
```

The above query will use NTLMv2 authentication with 128 bit transport encryption unless the server negotiates otherwise. This example uses the special '`DefaultNamingContext`' base identifier (which only works with AD DS servers). This example also shows how to use a properties file which ideally should contain `dns.servers` and `dns.site` properties so that a suitable domain controller can be properly located. Also, if a properties file is

used, the credentials can alternatively be supplied with the `service.acctname` and `service.password` properties instead of with the `-u` and `-p` parameters. The output of this example illustrates how the LdapSecurityProvider's builtin attribute definitions help format values such as attributes that are single-valued as opposed to multi-valued (shown with square brackets) and attribute values like times and the `objectSid` attribute which would otherwise not be easily interpreted by the average user.

The following example uses the same query as the first example but uses Kerberos authentication.

Note: Using the builtin Java Kerberos infrastructure to authenticate with AD DS is generally harder to use because the Java Kerberos implementation does not provide SASL transport security, it requires a krb5.conf file and Kerberos in general is sensitive accessibility of AD DS servers by clients, DNS and time differences between systems.

```
C:\>java -cp jespa-2.0.0.jar jespa.ldap.LdapSearch -a GSSAPI
'ldap://dc1.busicorp.local/DefaultNamingContext??sub?(sAMAccountName=hmuller)'
```

The above example assumes a Kerberos TGT is present in the user's Kerberos ticket cache. Alternatively, credentials may be supplied explicitly using the `-u` and `-p` parameters although the username must be in a principal name form with the domain in UPPERCASE such as `-u bcarter@BUSICORP.LOCAL`.

The following example illustrates how to query a user account in an RFC based LDAP server like OpenLDAP.

```
C:\>java -cp jespa-2.0.0.jar jespa.ldap.LdapSearch -d RFC -u 'CN=Alice
Baker,OU=Research,DC=openbook,DC=edu' -p opensaysme
'ldap://192.168.44.110/OU=Research,DC=openbook,DC=edu??sub?(uid=cdavis)'

cn=Chris Davis,ou=Research,dc=openbook,dc=edu:
  givenName: Chris
  sn: Davis
  userPassword: e0X1cJ1ENldvYkcvaTF63JNEZRpcH2JtNXpRPT0=
  uidNumber: 1003
  gidNumber: 5000
  objectClass: [inetOrgPerson,posixAccount,top]
  uid: cdavis
  cn: Chris Davis
  homeDirectory: /home/users/Research/cdavis
```

The first major difference between this example and the Active Directory example is that the server "disposition" is set using `-d RFC`. This indicates to the Jespa LDAP SecurityProvider that authentication and attribute definitions suitable for RFC-based servers should be used. Because the default authentication method for RFC-based servers is a "simple" bind, the username must be a full DN. Also, TLS encryption should be used. However, for simplicity, TLS is excluded in this example as it would require that a certificate be generated on the LDAP server, exported and then imported into a suitable Java `trustStore` file (see the `-t` parameter description). For simplicity, an IP address is used in this example. If a hostname were used, either DNS SRV records for LDAP would be required or, if a properties file is used, the property `authority.dns.names.resolve = false` would be required to disable SRV lookups in which case an FQDN hostname could be used.

The following LdapSearch example command illustrates how to retrieve the sAMAccountName of all users in Active Directory:

```
C:\>java -cp jespa-2.0.0.jar jespa.ldap.LdapSearch -f jespa.prp
'ldap://dc100.busicorp.local/DefaultNamingContext?sAMAccountName?sub?(objectCategory=Person)'
CN=Administrator,CN=Users,DC=busicorp,DC=local:
  sAMAccountName: Administrator
CN=Guest,CN=Users,DC=busicorp,DC=local:
  sAMAccountName: Guest
CN=SUPPORT_388945a0,CN=Users,DC=busicorp,DC=local:
  sAMAccountName: SUPPORT_388945a0
CN=krbtgt,CN=Users,DC=busicorp,DC=local:
  sAMAccountName: krbtgt
```

```
CN=Bob Carter,CN=Users,DC=busicorp,DC=local:
  sAMAccountName: bcarter
CN=Hans Müller,CN=Users,DC=busicorp,DC=local:
  sAMAccountName: hmuller
CN=host_ls1,OU=s,DC=busicorp,DC=local:
  sAMAccountName: host_ls1
CN=Baker\, Alice,CN=Users,DC=busicorp,DC=local:
  sAMAccountName: baker_alice
CN=IUSR_DC100,CN=Users,DC=busicorp,DC=local:
  sAMAccountName: IUSR_DC1
...
```

The following example illustrates how to query the RootDSE of an OpenLDAP server using the special + attribute to retrieve otherwise hidden attributes.

```
C:\>java -cp jespa-2.0.0 jespa.ldap.LdapSearch -d RFC2251 'ldap://192.168.44.110/?*,+'
```

For other examples of LDAP URLs, see the LdapSecurityProvider API documentation.

Providing NTLM Services without Active Directory

With a set of usernames and plaintext passwords, Jespa can act as it's own domain authority and validate NTLM credentials submitted by clients. From the perspective of clients (like browsers), this method is identical to authenticating with Active Directory. Jespa is simply computing the correct NTLM response with the plaintext password and comparing it byte-for-byte with the client supplied response. The highest level of security is negotiated just as it would with AD. All of this is completely transparent to clients.

The HttpSecurityFilter, SaslServer, JAAS LoginModule and other service components support a `provider.classname` property that allows the operator to specify the security provider that should be used without changing any code. The jespa.ntlm.NtlmSecurityProvider uses the NETLOGON service and requires a Computer account. However, you can extend the NtlmSecurityProvider class and install it with the `provider.classname` property to validate credentials using an alternative source of plaintext passwords. Reasons for doing this might include:

- You are using a local isolated database of passwords and do not need to validate credentials with AD
- You are only developing the application and do not yet need to actively validate credentials with AD
- You do not currently have permission to create the Computer account in AD

The MyNtlmSecurityProvider Example

Jespa includes a jespa.examples.MyNtlmSecurityProvider class that validates credentials with a single set of credentials supplied as properties. The source code to this class is located at `src/jespa/examples/MyNtlmSecurityProvider.java`. This minimal example is intended to illustrate precisely how to create a custom NTLM security provider that acts as an AD domain authority.

The MyNtlmSecurityProvider class is an example but it is also fully functional and may be used with most of the aforementioned services such as the HttpSecurityFilter, SaslServer, etc by simply setting the `provider.classname` to "jespa.examples.MyNtlmSecurityProvider" and then setting properties "jespa.domain.dns.name", "jespa.domain.netbios.name", "jespa.my.username" and "jespa.my.password" to match the acceptable credentials.

Appendix A: How to Collect Diagnostic Information

If you are experiencing a issue with Jespa, IOPLEX Software support will likely ask for diagnostic information in the form of at least a Jespa log file (and possibly a network packet capture file).

Collecting a Complete Jespa Log File

To acquire a Jespa log file, set the following properties in your Jespa properties file:

```
jepsa.log.level = 4
jespa.log.path = /path/to/jespa.log
```

If the path was added or changed, you may need to restart the web app (or app server) for the change to be recognized.

Now trigger the behavior or error of interest and send the Jespa log file to IOPLEX support as an email attachment. Preferably try to include the security provider properties which are logged when the webapp is first initialized. You may wish to globally search and replace all instances of hostnames and IP addresses or other information that you think might be sensitive. Passwords are not logged (the password.encrypted fields use a random key that changes each time Jespa is initialized). If the file is larger than a few MB, compress it into a regular .zip first.

Obtaining a Network Packet Capture

Some issues cannot be diagnosed from a log file only. In some instances IOPLEX support may ask for a network packet capture.

If you are using Linux, the most efficient tool for capturing network traffic is tcpdump. Start a capture as root using a command like the following:

```
# tcpdump -s 0 -w jespa.pcap ! port ssh
```

Perform the operation you wish to capture, such as visiting your Jespa protected website, and then press Ctrl-C to stop the capture. A jespa.pcap file should be created in the current directory.

To obtain a packet capture on Windows, you can use the free WireShark application or any of several other capture utilities for Windows.

WireShark is a separate application that must be downloaded and installed from a third-party website but it has advanced features and a UI that will allow you to visually navigate and filter network traffic. For Windows, simply download and install the package from the WireShark website:

https://www.wireshark.org/

WireShark is a standard package on most Linux distributions.

To obtain a capture using WireShark, run it as Administrator or root and select *Capture* > *Start*, trigger the communication of interest and then select *Capture* > *Stop*. Finally select *File* > *Save As ...*, select the file type of "Wireshark/tcpdump... - libpcap" and save the file with a .pcap extension. Send the resulting .pcap file to IOPLEX support.

> Note: You must run the capture software on the machine sending and receiving the communication of interest. For example, if you wish to capture communication between a web server using the HttpSecurityService and an Active Directory server, you will need to run your capture on the web server (for production systems, using tcpdump or netsh on Windows may be more appropriate as opposed to installing WireShark).
>
> Note: If you are asked to capture communication between Jespa and an Active Directory domain controller, you may need to set the property `jespa.netlogon.useSealing = false` so that the NETLOGON communication is not encrypted.

Note: It is preferred that the capture only be allowed to run for the shortest time possible so as to isolate the traffic of interest. Similarly it is preferred that other programs are not running (or at least not generating traffic) while the network capture is running.

Ideally, you should collect a log file and a corresponding packet capture simultaneously so that the information in the log can be correlated with data in the capture. This will provide IOPLEX support with the most comprehensive diagnostic information possible.

Note: If you are trying to diagnose an undesirable change in behavior between two versions of Jespa, you can also obtain *two* sets of captures and logs - one of Jespa working and one of Jespa failing and send all four (4) files as attachments to support@ioplex.com with your explanation of the problem.