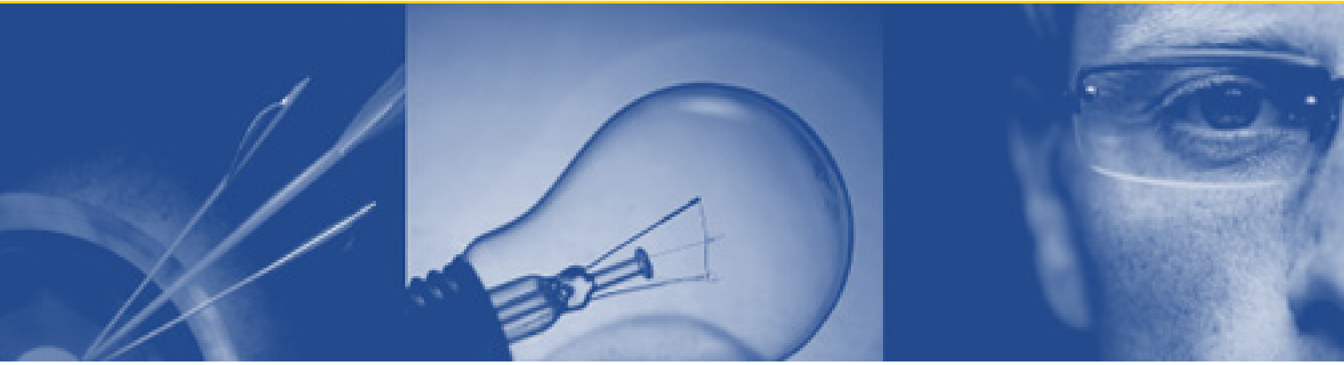
QUEST SOFTWARE®

Vintela Single Sign-On for Java

Reference Manual

Standard Edition 3.3



If you have any questions regarding your potential use of this material, please contact:

Quest Software World Headquarters
LEGAL Dept
5 Polaris Way
Aliso Viejo, CA 92656
USA
www.quest.com
email: legal@quest.com
telephone: 949.754.8000

Please refer to our Web site for regional and international office information.

TRADEMARKS

Vintela Single Sign-On for Java Standard Edition Reference Manual

Software Version 3.3 Doc dated: March 2008

CONTENTS

PREFACE vii

WHO SHOULD READ THIS MANUAL? viii

CHAPTER 1
INTRODUCTION TO VSJ 1

OVERVIEW 2

ABOUT KERBEROS 4

ABOUT ACTIVE DIRECTORY 6

ACTIVE DIRECTORY GROUPS 7

Group types 7

Group scopes 7

Groups and the logon process. 8

Groups and VSJ 9

ACTIVE DIRECTORY SITES 9

MAPPINGS AND OBJECTS 10

Names and mappings 10

Active Directory objects 11

USE OF PRINCIPAL NAMES IN VSJ 11

VSJ AND ACTIVE DIRECTORY 12

SPNEGO AND INTERNET EXPLORER 12

KERBEROS DELEGATION EXTENSIONS 13

WHY DELEGATE? 13

Delegation "trust" in authentication 14

DELEGATION OPTIONS 14

UNCONSTRAINED DELEGATION 15

CONSTRAINED DELEGATION (S4U2PROXY) 15

PROTOCOL TRANSITION (S4U2SELF) 16

HOW DOES VSJ WORK? 19

EXAMPLE DOMAIN 21

CHAPTER 2
PREPARING FOR VSJ 23

PRE-INSTALLATION OVERVIEW 24

NETWORK INFRASTRUCTURE. 25

ACTIVE DIRECTORY ENVIRONMENT 25

Domain Name Service (DNS) 25
Time Synchronization Service 26
Configuring Active Directory for VSJ 27
Setting up the VSJ account 27
VSJ setup using Active Directory tools 27
VSJ setup with VAS on UNIX or Linux 31
Enabling delegation for VSJ. 35
Delegation configuration in different systems 35
Setting up a Java application server host 41
Creating keytab files 41
Setting up a client machine. 43
Operating System 43
Browsers and authentication 44
Setting up Internet Explorer for SSO 44
Windows Integrated Authentication 44
NTLM authentication 45
Troubleshooting your Internet Explorer configuration 45
Chapter 3
Deploying VSJ 47
Getting started with VSJ. 48
To obtain a license for VSJ 48
HTTP header size limits 48
Configuring the VSJ examples 49
VSJ and your web applications 51
Deployment options 51
Deploying in a web application 51
Deploying on the CLASSPATH 52
Deploying in application server specific path. 52
Creating a deployment descriptor for SSO 52
Deploying SSO web components .52
Configuring the VSJ parameters 53
Building a war file for SSO 54
Setting up logging 55
Controlling access to resources 57

Authorization using Active Directory groups 57
Java EE authorization model for servlets/JSPs. 57
VSJ authorization 60
Recommendations for managing authorization 65
Writing access policy files. 65
Overview of policy files 65
Preconditions for writing an XML policy file. 66
Creating the policy XML file 67
Policy XML descriptor elements .70
role 70
include 72
exclude. 73
user. 74
group 74
security-constraint. 74
web-resource-collection 75
auth-constraint. 75

Chapter 4

VSJ Federation and ADFS 77
Features of VSJ Federation 78
Prerequisites for VSJ Federation .78
VSJ Federation installation 79
Obtaining a license for VSJ Federation 79
XML libraries 79
VSJ Federation deployment 80
Configuring VSJ Federation 82
Known Issues with Federation 83

Chapter 5

Security Issues. 85
Basic recommendations 86
Deployment risks. 86
Service unavailability 86
Time synchronization 87
Replication interruptions 87
Resource security 87

Client issues with security 87
Cookies 87
Caching of passwords for basic fallback 88
SPNEGO/Windows Authentication 88
Lifetime of authentication 88
Session IDs 89
Active Directory permissions 90
Basic fallback 92
Keytabs and passwords 93
Authorization 93
Do you need authorization? 93
Securing LDAP 93
Using the *Principle of Least Privilege* 94
Denial of Service 94
Basic fallback 95
Session state 95
Auditing 95
NTLM authentication 96
What is NTLM? 96
Different versions of NTLM 96
NTLM and Internet Explorer 97

Chapter 6
Maintenance and Troubleshooting 99
Maintenance 100
Logging 100
New Users / Groups 100
Account settings 100
Network topology changes 100
Troubleshooting 101
General 101
Active Directory 102
Browsers 102
Internet Explorer browsers 102

Non-Internet Explorer Browsers 103
AUTHENTICATION 104
KEYTABS 106
NETWORK CONNECTIONS 107
CREDENTIAL DELEGATION 107
DEBUGGING 108
APPENDIX A:
CONFIGURATION PARAMETERS 109
APPENDIX B:
USING THE JKTOOLS 119
INDEX Index-1

Preface

As the use of distributed systems increases, users need to access resources that are remotely located. Traditionally, as a user of these remote resources, you have had to sign-on to each one of them in turn. Often, each resource you sign-on to requires a different username, password and authentication technique — as if you don't already have enough passwords and identities to remember!

The much more friendly alternative to these arrangements is a *single sign-on (SSO)* system. On the ideal system of this kind, you need only authenticate once, and then have your authenticated identity securely carried across the network to reach all the resources you need to access.

Two trends in system development have now come together to make this ideal feasible:

- the extending use of the Java Enterprise Edition™, (Java EE) for development work; and
- the widespread availability of Microsoft's Active Directory system for user authentication.

Java EE is a platform for developing Internet, intranet and extranet applications. It provides a standardized architecture that makes reuse possible. Many enterprises have deployed Java EE applications.

In addition, many enterprises are moving to support a standardized authentication infrastructure. In particular, Microsoft's Active Directory® provides an environment based on Kerberos and LDAP, supplying Identity Management services including SSO, a centralized store for identity information.

It makes a lot of sense to reuse this infrastructure where possible.

Unfortunately, however, Java EE alone does not provide tight integration with Kerberos, nor with the infrastructure provided by Microsoft's Active Directory which is already deployed or being deployed in many organizations.

That is where Vintela Single Sign-On for Java, from Quest Software, comes into the picture.

VSJ fills the gap between development platform and operating system security. It provides SSO and access management for Java EE applications using Active Directory as their identity store.

It delivers an enterprise-wide method of identification and authorization that can be administered in a consistent and transparent manner.

It allows you access to information systems for which you are authorized — and only those systems.

Who should read this manual?

The *VSJ (Standard Edition) Reference Manual* is intended for developers of VSJ solutions for integrated SSO applications.

You should have a good knowledge of Java programming, and a sound understanding of how Active Directory works in your environment.

1

Introduction to VSJ

- Overview
- About Kerberos
- About Active Directory
- SPNEGO and Internet Explorer
- Kerberos delegation extensions
- How does VSJ work?
- Example domain

This section introduces the concepts involved in VSJ and its associated protocols.

Overview

VSJ provides a mechanism for integrating Java EE applications into a Single Sign-On infrastructure, based on Active Directory.

Once deployed, it can be integrated with your application environment so that it sits between clients registered in your Active Directory system and the Active-Directory-registered services they want to access.

Importantly, all of this occurs without your Java application code having to concern itself with the complex issues of access details and permissions.

VSJ becomes the mediator in the processes of handling web browser information requests directed at your Java application servers, and in the checking of user identity and access rights for these requests. This is possible even when the browser requests may require a complex series or a chain of server accesses — for example, when a web page on one server offers email despatch services directed to another server and, perhaps also requests information from a protected database on a third server.

Without a centralized Single Sign-On system, different applications may require a series of user/password exchanges before access is given. With VSJ, the authorization process is conducted as part of the web browsing process: only one initial sign-on is needed, even where quite complex server requests are involved.

VSJ allows Java EE applications to authenticate users using Kerberos. To do this, it supports the SPNEGO protocol with Internet Explorer and Firefox/Mozilla. And it can support "delegated" credentials to access other Kerberized services within an enterprise domain, as in cases of "chained" access requests.

Active Directory features such as groups and Active Directory sites are supported in a VSJ-based system, and existing groups and sites can be integrated into it. By specifying which users belong to which Active Directory groups, and which Active Directory groups are allowed to access an application, you can apply granular management of access control for large numbers of users.

VSJ uses Active Directory sites to support replication and failover.

By using the VSJ solution you will be able to provide:

- End-to-end authentication between users and backend services
- Authorization of users by using Active Directory groups
- Integration of Java EE applications in an Active Directory/Kerberos-based SSO environment

- A cross-platform solution which supports most operating systems and Java application servers

as well as:

- Delegation of credentials to selected services (*S4U2Proxy*), and
- Secure credentialing for clients signing on from non-Kerberos authentication processes (*S4U2Self*)

where these are supported by your Active Directory host (Windows Server 2003 and Windows Server 2008).

About Kerberos

Kerberos is the underlying network authentication protocol that VSJ and Active Directory use to provide secure communications.

Kerberos is designed to provide authentication for client-server applications across insecure network connections, using strong secret-key cryptography. It allows entities communicating over networks to prove their identity to each other while preventing eavesdropping or replay attacks. Kerberos also provides checks for data stream integrity (detection of modification) and secrecy (preventing unauthorized reading) using cryptographic ciphers such as DES, RC4 and AES.

Kerberos works by providing "principals" (users or services) with tickets that they can use to identify themselves, and secret cryptographic keys for secure communication with other principals. A ticket is a sequence of a few hundred bytes. Each ticket can then be used to secure virtually any other network protocol, thereby allowing the processes implementing that protocol to be sure about the identity of the principals involved.

Kerberos provides for mutual authentication and secure communication between principals on an open network by manufacturing secret keys for any requester, then providing a mechanism for these secret keys to be safely propagated through the network.

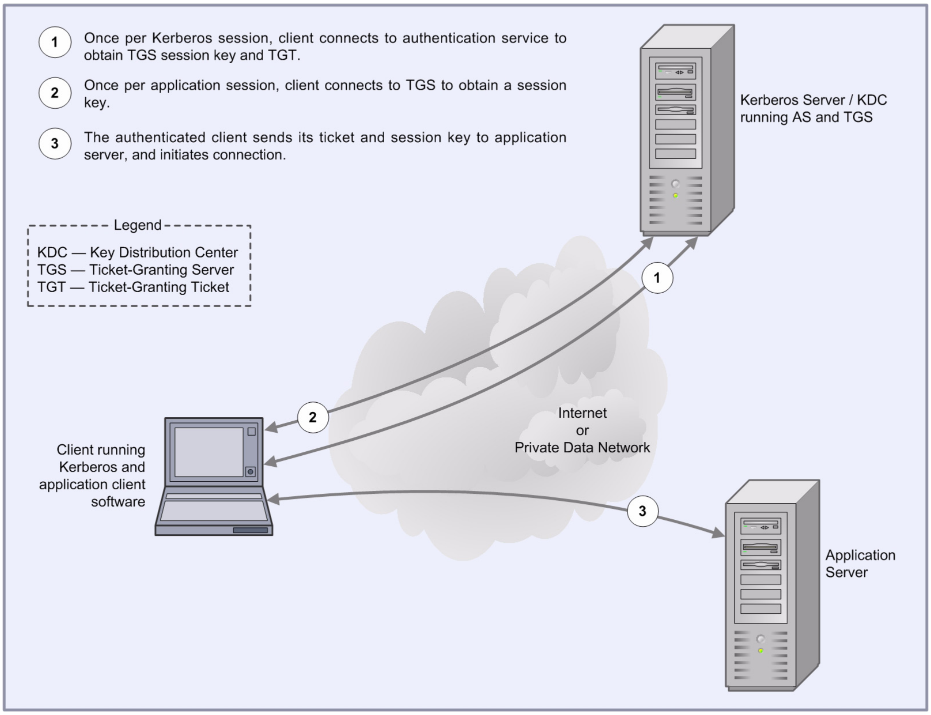
Kerberos does not, strictly speaking, provide for authorization or accounting, although applications may use their secret keys to perform those functions securely. The Kerberos emphasis is on authentication and opening of communication channels. Once a principal is authenticated, its details can be checked to see if it is authorized to access the resource requested.

Figure 1 shows how the Kerberos protocol works.

A principal authenticates itself in Kerberos by using a principal name of the form principal-name@realm and a password. This is typically used to send an encrypted message to the Authentication Service (AS).

The Authentication Service can then authenticate the principal (in Windows, it can check its details in Active Directory) and send back a session key and Ticket Granting Ticket (TGT).

Figure 1: Kerberos Protocol


1 Once per Kerberos session, client connects to authentication service to obtain TGS session key and TGT.
2 Once per application session, client connects to TGS to obtain a session key.
3 The authenticated client sends its ticket and session key to application server, and initiates connection.
Legend
KDC — Key Distribution Center
TGS — Ticket-Granting Server
TGT — Ticket-Granting Ticket
Kerberos Server / KDC running AS and TGS
1
2
Internet or Private Data Network
Client running Kerberos and application client software
3
Application Server

The TGT is like a certificate of identity which allows the principal to gain later access to one or more services. Once the user has supplied a password and obtained the TGT from the AS, authentication to any other service can proceed automatically without the user having to resupply the password. For this reason, Kerberos is sometimes called a Single Sign-On (SSO) service.

A ticket is a credential that enables a principal to gain access to a service. A principal obtains tickets from the Ticket Granting Service (TGS) using the TGT obtained from the AS as described above. The ticket is used to create an authenticator, which is then sent to the service being requested to authenticate the user.

The authenticator is used to establish a session key for secure communication. Optionally, the user can also request to authenticate the server. If this happens, the server uses information in the user's authenticator to send back a server authenticator, which the user can use to verify the server's authenticity.

The Kerberos protocol has been widely analyzed and is supported by many vendors including Sun, Microsoft and Oracle.

About Active Directory

The Active Directory service is a core component of the Microsoft Windows operating system. It provides a directory service supporting the Lightweight Directory Access Protocol (LDAP), and has an integrated Kerberos key distribution center to authenticate users.

It allows organizations to share and manage information about network resources and users and provides an SSO environment that integrates with the standard Windows desktop login. In addition, it acts as a single point of management for Windows-based user accounts, clients, servers and applications.

The directory is arranged hierarchically, allowing the division of enterprise resources into different domains. Each resource (that is, user, application), is represented as an object with a number of attributes (for example, the organizational group to which the resource belongs).

The directory may be browsed hierarchically for resources, or each resource can be individually addressed by providing its Distinguished Name. The Distinguished Name is simply a group of attributes that uniquely identify an object within the Active Directory hierarchy (for example, “CN=John Doe, DC=example, DC=com”).

The directory also provides fine-grained security mechanisms to allow administrators to determine exactly what information may be accessed. Users can be restricted to specific objects, or even specific attributes within the directory.

The main benefits of using Active Directory are that it simplifies the management of user accounts, and provides an SSO infrastructure to users. Its support for standard protocols such as LDAP and Kerberos means that it can be used as, or with, a cross-platform solution.

The Kerberos support in Active Directory has been tested to ensure interoperability with the MIT Kerberos implementation used by many UNIX vendors. However, it is worth noting some differences between the Microsoft and MIT implementations:

- *Support for Privilege Attribute Certificates (PACs)*

  Microsoft's Kerberos implementation uses the `AuthorizationData` field of the Kerberos ticket to pass Privilege Attribute Certificates (PACs) to Kerberized applications. Applications that support Microsoft's PAC format can use this information to provide fine-grained access control to services.

- *Integration with LDAP*

  Active Directory's Kerberos features are tightly integrated with its LDAP server. This means that user information, such as groups, can be retrieved using standard tools and APIs.

- *Windows Native Credential Cache*

  Unlike the MIT implementation, the Windows Kerberos implementation uses an in-memory credential cache to store Kerberos tickets and TGTs (the MIT implementation uses a disk file). The implementation is stored in non-paged memory so it is never written to disk. Microsoft provides routines to obtain credentials from this cache through their Local Security Authority API (LSA API).

- *Smartcard / PKI Support*

  Microsoft supports a version of the PKINIT protocol which allows the initial authentication to the directory to be performed using a private key or smartcard.

Active Directory groups

In large organizations, managing the authorization permissions of hundreds or even thousands of users creates a significant problem. One way that Active Directory addresses this issue is through the use of groups. These groups provide a way to classify users according to their roles or activities, and can be used as the basis for authorization permissions.

Group types

Active Directory defines two types of groups: security groups and distribution groups. Distribution groups are mainly used for activities such as sending bulk email, and do not allow permissions and audit settings to be associated with them. Security groups on the other hand are primarily designed for authorization, so are most interesting for VSJ. Microsoft Windows does not use distribution groups internally, although they are available for use by other directory-enabled applications.

Group scopes

Each group has a *scope* which describes both its visibility to other users and applications throughout the enterprise, and the types of principals that can be included in the group. The three group scopes are listed below:

- *Domain Local* — visible only within a domain, and to sub-domains. Domain local groups can include any other type of group, but cannot be included in any of the other groups. Commonly, Domain local groups are used to define the groups and users allowed to access a given local resource.
- *Global* — visible throughout the enterprise, but can only include other users and groups within the same domain (or a parent domain). Global groups should be used to divide users within a domain into categories according to their roles or job functions.
- *Universal* — visible throughout the entire enterprise. Universal groups may include both other Universal groups, and global groups from any domain in the enterprise. For reasons described below, Universal groups are expensive to use, so try to limit the number of these groups. You should use a Universal group wherever it is necessary to create a group that spans one or more domains.

If you are using Active Directory in mixed mode (that is, with both Active Directory and Windows NT domains) the rules for scoping are more restrictive, and you cannot use Universal groups. See your Active Directory documentation for more details.

Active Directory allows nested groups (groups that contain other groups), which can be nested to arbitrary levels. This is convenient, as it provides a way to hierarchically manage users. For example you can have a group for each type of manager in a department, and then create a group called *Managers* that includes all of these sub-groups.

Groups and the logon process

When a Microsoft Windows user logs on to an Active Directory domain, Active Directory builds a token called a Privilege Attribute Certificate (PAC). The PAC contains the list of all the groups the user is a member of, and that are visible in the login domain. This list is a "transitive closure", meaning it includes any groups which the user is a member of due to *nesting* of other groups. For example, supposing we have the following global groups which contain the users "Alice" and "Bob":

- Group1: contains Alice, Bob
- Group2: contains Group1
- Group3: contains Group2

Then the PAC for Alice will contain Group1, Group2 and Group3.

Computing this group membership can take some time and Active Directory may have to query several LDAP servers to determine membership. In addition, Universal groups require a full search of the global catalog, which stores information about every Active Directory object. For this reason, the number of Universal groups should be limited to ensure that logon times do not become too long.

The PAC exists for the lifetime of the initial TGT obtained at logon time (typically 10 hours). For this reason, if a user is added to a new group, the user must log off and then log back on for this group to be used.

Groups and VSJ

When a user accesses an application protected with VSJ, Internet Explorer contacts Active Directory to obtain a service ticket for the server which it needs to contact, or retrieves it from the cache if it is available there.

This service ticket contains a PAC containing all the Universal and Global groups from the user's existing PAC, plus any Domain Local groups defined in the server’s domain.

VSJ can then use the groups defined in the PAC to authorize access to a particular resource. Because the group membership is securely authenticated in the PAC, VSJ can trust this information.

Using the PAC also saves time associated with the LDAP lookups needed to recursively resolve group membership, resulting in a more scalable solution.

Active Directory sites

Sites represent the physical topology of your Active Directory infrastructure based on sub-nets of TCP/IP addresses. They allow for replication, redundancy and load balancing across your Active Directory deployment.

By defining sites in Active Directory, you effectively tell it what your underlying physical network looks like. This allows domain controllers to utilize the underlying network in the most efficient manner possible. It allows Active Directory to conserve bandwidth that is required for other applications within your organization.

However, sites are not related to the structure of the domain, nor do they have to maintain the same namespace — a site may span multiple domains and, conversely, a domain may span multiple sites. While no formal relationship exists between a site and a domain, a site may be given the same boundaries as a domain.

To ensure rapid and reliable network communication, Active Directory offers methods of regulating inter-sub-net, or inter-site, traffic.

The Active Directory physical structure governs when and how replication takes place. As users log on to the network, they are able to reach the closest domain controller site through the previous assignment of sub-net information. The system administrator uses the Active Directory Sites and Services snap-in to manage the topology of replication services.

VSJ supports replication and failover using Active Directory sites.

Mappings and objects

VSJ makes use of a number of Active Directory constructs and objects in its Kerberos-related operations.

These involve:

- Objects representing users, groups, organizations, computers, resources and services as registered in Active Directory account records
- Mappings or unique name references to these objects, including properties held in the accounts themselves.

Names and mappings

User Principal Names and Service Principal Names are recorded in Active Directory and can be used as a way of referring to a client or a service within Kerberos. Active Directory can find full user account details for a UPN or SPN by searching its records for the account that has that name property listed as one of its object attributes.

User Principal Names (UPNs)

A UPN provides a name of a user and is used as the Kerberos client principal name. It consists of an RFC822 or Email-style name and domain, separated by an '@' symbol (thus, `fred@host493.example.com`).

The UPN must be unique throughout the Active Directory forest.

The UPN is the name that appears in the Kerberos Ticket Granting Ticket (TGT) returned by Active Directory for a client.

Service Principal Names (SPNs)

Most often, an SPN is of the form: `<service type>/<host>`. For example:

```
HTTP/appservhost1.example.com
```

The SPN is used when requesting a Kerberos ticket for a particular service. The client browser uses the hostname from the request URL to construct this SPN value. This SPN value is used to request a service ticket from Active Directory. You will typically need to map an SPN to an Active Directory account using the `setspn` tool.

Active Directory objects

SPNs may be mapped to two different types of objects in Active Directory:

- Computer Objects and
- User Objects

Computer Objects refer to computers which have been joined to a given Active Directory domain (for example, a Microsoft Windows machine joined using the Network Identification Wizard, or a UNIX machine joined using VAS).

When the Computer Object is created, a corresponding SPN in the form `HOST/<machine_name>` is mapped to that computer object. It is also possible to map additional SPNs to the computer object if required.

An Active Directory User Object may refer to an actual physical person, or a *Service Account* which is an object created for the purposes of running a particular service. In the latter case, you would map the SPN to the service account using the `setspn` tool.

One service account of this kind is in an account set up for VSJ.

Use of Principal Names in VSJ

When running VSJ on Microsoft Windows it is only possible to configure VSJ to use the HTTP SPN mapped to a Service Account User Object. (This is because it is not possible for VSJ to access the key material for the Computer account stored in the LSA database.)

For more details on setting up VSJ to use a Service Account, see Setting up the VSJ account).

However, when VSJ is used with Vintela Authentication Services (VAS), either the HOST/ or HTTP/ mechanisms can be used (see VSJ setup with VAS on UNIX or Linux for more details).

VSJ and Active Directory

VSJ has been specifically designed to work optimally with Microsoft's Active Directory. It will use the default configuration to discover the services, hosts and port numbers necessary to integrate with Active Directory. To fine-tune your deployment you can configure VSJ to use Active Directory configuration for items such as sites and services, domain controller servers, etc.

SPNEGO and Internet Explorer

In addition to support for Kerberos through its Active Directory service, Microsoft has also provided extensions to Internet Explorer that allow it to participate in a Kerberos-based SSO environment. When a Web server receives a request from an Internet Explorer browser, it can request that the browser use the SPNEGO protocol to authenticate itself. SPNEGO in effect is a way of negotiating what protocol is appropriate for Internet Explorer (and now, other browsers) to use to establish initial credentials with a server running a Kerberos-based service.

SPNEGO performs a Kerberos authentication which is tunneled over HTTP, allowing the browser application to pass a delegated credential (acting for the Windows user running the IE instance) so that a Web application can log in to subsequent Kerberized services on the user's behalf.

When an HTTP server wants to perform SPNEGO, it returns a "401 Unauthorized" response to the HTTP request with the "WWW-Authenticate: Negotiate" header. Internet Explorer then contacts the Ticket Granting Service (TGS) to obtain a service ticket. It chooses a special Service Principal Name for the ticket request, which is:

```
HTTP/webserver
```

The returned ticket is then used to construct an SPNEGO token which is encoded and sent back to the server in the HTTP headers. The token is unwrapped and the ticket is authenticated. If mutual authentication is required, the Web server can return an additional SPNEGO token for the client to verify. Once authenticated, the page corresponding to the requested URL is returned.

SPNEGO provides a useful mechanism for extending an SSO environment to Web applications. It is already supported in Microsoft IIS for authentication to ASP or Web pages. In addition, the ability to delegate credentials means that a Web application can log in to further services transparently on the user's behalf, providing full end-to-end authentication. Lastly, SPNEGO and HTTP can be used for authentication with Microsoft .NET SOAP clients, and the HTTP Negotiate extension of SPNEGO is supported in browsers such as Mozilla and Firefox.

Kerberos delegation extensions

Changes in Windows Server 2003 introduced extended features for the use of delegated credentials in Active Directory operations. These extensions, referred to as *S4U2Proxy* and *S4U2Self*, also apply to Windows Server 2008, and are supported in VSJ.

Why delegate?

When a user requests information in a web page from a browser, more than one application or server may be involved. It may not be possible for the first server receiving the request to provide all details for the page requested. That first server, for example, may have to seek some information for example, from a database on a second server.

Any dependency pattern of this kind then has implications for the process of Kerberos authentication. A Kerberos requirement is that at each stage of the requesting process, the parties involved have to be able to authenticate with each other. In practical terms, without use of a "shortcut" such as delegation, this could be very difficult to achieve.

For example, the original client may have no knowledge at all of any secondary stages in the request process: it just knows that it is supposed to get information from the first server it approaches, and relies on a single request (for example, for a web page).

Theoretically, one approach might be for the client to have to directly and separately authenticate with every information source: but that approach is clearly impractical and may be impossible.

Delegation "trust" in authentication

In order to deal with the problems involved when multiple points of authentication of this kind may be needed, Active Directory allows you to establish a form of trust between clients and services. What this means is that a client may be prepared to permit one or more services to act on its behalf to establish authentication. There are configuration options on each account which handle permissions of this kind, and which can allow for a form of "delegated identity".

A client can specify that other services can act on its behalf, as its "delegate" in establishing authentication. It may do this in a general sense with a setting saying the client’s identity may be delegated -- that is, generally, other services may "impersonate" it for authentication purposes. This is known as "unconstrained" delegation.

In some circumstances, however, the client can set more refined limits on delegation, and can specify a list of those services that it allows to act on its behalf in the course of authentication processes, thereby establishing that those **not** specified are not given such permission.

Delegation options

The limits that can be configured for delegation now involve two or three basic configuration options, varying with your Active Directory platforms:

For Windows 2000 Server, Windows Server 2003 and Windows Server 2008:

- Delegation disallowed — a Kerberos service is not to be trusted for delegation at all
- Delegation allowed for all services — use the *Unconstrained* option for Kerberos using a TGT

and, for Windows Server 2003 and Windows Server 2008:

- Delegation only allowed for a limited set of services — delegation to specified services only (the *Constrained* option), with sub-options for either using *Kerberos only*, or allowing *any authentication protocol* (the *Protocol transition* option)

For the last of these, further configuration permits a selection of Service Principal Names (SPNs) for the services for which delegation is allowed.

Unconstrained delegation

In this form of delegation, known as *Unconstrained delegation*, a client which has established its identity can transmit that identity to other entities, asking them to act on its behalf in the authentication process.

As far as the Active Directory domain controller is concerned, each server which has received such a delegation from a client is assumed to **be** the client, because it can present the client’s Ticket Granting Ticket, or TGT. As far as Active Directory is concerned, at the authentication level, the server is indistinguishable from the client itself.

Unconstrained delegation of this kind can imply multiple security risks and the possibility of unlimited "spoofing" of the client if a server which acts as a delegate and holds a TGT falls vulnerable to a security attack.

Kerberos Delegation extensions introduced with Windows Server 2003 and included in Windows Server 2008 now make it feasible to avoid these risks, and VSJ supports the new "Constrained" form of delegation as an additional option, known as the *S4U2Proxy* extension.

Constrained delegation (S4U2Proxy)

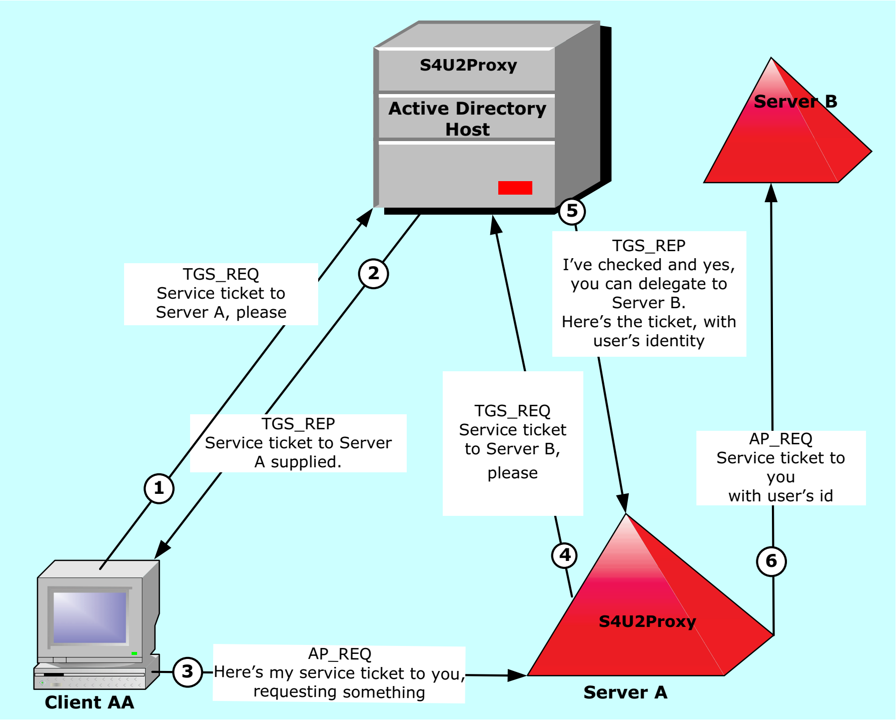
In versions of Active Directory that support constrained delegation, delegation can be refined so that it works through the use of a series of "service tickets" instead of the transmission of the user's TGT. Using these tickets, it permits delegation of the client’s identity directed to one or more servers, but only to those specifically selected by configuration as being permitted to be authentication delegates.

In constrained delegation, the client does not send the TGT, it simply sends the service ticket. The server can then present this service ticket, along with the server's own TGT, to Active Directory in order to request a service ticket using constrained delegation to another service. Active Directory will only grant service tickets from the server to a specific list of services that have been previously configured.

Constrained delegation may be repeated through multiple tiers. For each tier, a domain controller is responsible for issuing a new service ticket and checking whether the authenticated server is permitted to perform constrained delegation to the next server.

Figure 2 is a simplified summary of the sequence of events involved in the Constrained delegation (*S4U2Proxy*) processes.

Figure 2: Constrained Delegation with S4U2Proxy


S4U2Proxy
Active Directory Host
Server B
5
TGS_REQ
Service ticket to Server A, please
2
TGS_REP
I've checked and yes, you can delegate to Server B.
Here's the ticket, with user's identity
TGS_REP
Service ticket to Server A supplied.
TGS_REQ
Service ticket to Server B, please
AP_REQ
Service ticket to you with user's id
1
4
6
S4U2Proxy
AP_REQ
Here's my service ticket to you, requesting something
3
Client AA
Server A

Protocol transition (S4U2Self)

Clients that need access to Active Directory services can include external clients, which for a variety of reasons do not have immediate access to facilities for a full exchange of credentials under Kerberos.

Some clients may use alternative authentication methods by choice or design (for example, certificate-based or HTTP digest authentication, NTLM authentication, or Federation (see VSJ Federation and ADFS).

Some external clients may operate through a firewall, where Kerberos operations are generally considered inappropriate, but where the client is still required to authenticate, and there is a need for Single Sign-On access to internal services.

Such external clients might be able to be granted access via Active Directory's Kerberos environment if the internal server were to be passed the client's full logon details or its TGT.

However, this is not a preferred approach, and on a strict application of security principles, might be seen as serious breach of the trust rules for a secure system.

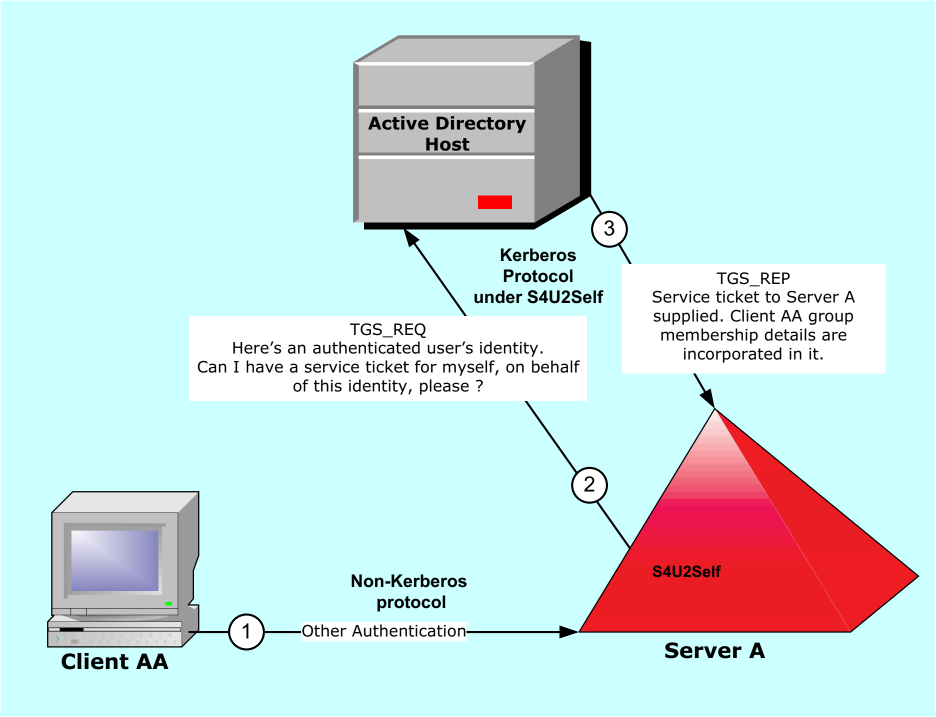
In Windows Server 2003 and above there is a solution for this: the "protocol transition" (S4U2Self) extension.

This allows the server itself to collect sufficient information about the client to establish a logon token which clears it for access without the need for a user password.

Under this arrangement, the server, rather than the client, requests a Kerberos ticket **for itself**, on behalf of the "otherwise-authenticated" client.

Figure 3 illustrates the event sequence involved (in simplified terms).

Figure 3: Protocol transition and S4U2Self


Active Directory Host
3
Kerberos Protocol under S4U2Self
TGS_REP
Service ticket to Server A supplied. Client AA group membership details are incorporated in it.
TGS_REQ
Here's an authenticated user's identity.
Can I have a service ticket for myself, on behalf of this identity, please ?
2
S4U2Self
Non-Kerberos protocol
1
Other Authentication
Client AA
Server A

The ticket returned to the server is a service ticket for the server concerned, but it contains the user's authorization data, and it is of a form capable of being used for the *S4U2Proxy* extension — that is, a ticket to be used for delegated credential ling on other servers.

VSJ now provides support for *S4U2Self* as well as *S4U2Proxy*, and for the use of both extensions in conjunction with each other.

How does VSJ work?

In VSJ Standard Edition, VSJ is implemented as a Java servlet filter.

On startup:

On initialization, the VSJ servlet filter uses the given configuration parameters to determine:

- the principal created for VSJ operations
- the credentials of the VSJ service principal, and
- various security options

For each request:

1. If the request is not yet authenticated:

   VSJ performs the appropriate authentication for the mechanism specified (that is, in the 'Authorization' header of the request). It may attempt, in preferred order:

   - **SPNEGO:**

     The mechanism name given for this is "Negotiate", and the mechanism token is a SPNEGO token.

     SPNEGO handshake occurs; the SPNEGO token in the request is processed, and SPNEGO tokens may be returned to the browser in the "WWW-Authenticate' field of the response.

     Eventually, if authentication succeeds, credentials are obtained.

     The session state is updated with the obtained credentials and other user information.

   - **NTLM** (if configured):

     The mechanism name is "NTLM" or "NEGOTIATE", and its mechanism token is an NTLM token. (See What is NTLM?)

     An NTLM handshake occurs, with messages exchanged between the client and server. These include negotiate, challenge and authenticate packets.

     Once authentication occurs, a credential is obtained and stored in the session state for future authentication requests.

   - **Basic fallback** (if configured):

The mechanism name here is "Basic", and its mechanism token consists of a base64-encoded username and password.

It works by converting username and password to a Kerberos principal and Kerberos password, requesting a TGT from the Active Directory domain controller's in-built key distribution service.

It also requests a service ticket from the same source.

It stores credentials in the session state data for future authentication requests.

2. Once the request is authenticated, access to the specified resource is checked:

   If no access policy has been set, access is automatically granted.

   If an access policy has been set:

   - Account information about the authenticated user is obtained (for example, group membership, last login, etc.).
   - If such information is not already available, it is obtained via LDAP queries on the domain controller or information in the service ticket.

     If NTLM is used, all information is obtained via LDAP queries on the account of the `user@domain`. Otherwise, the service ticket contains Microsoft authorization data (a Privilege Attribute Certificate or PAC).

     The PAC contains information such as last login time, group membership, etc. Active Directory may be queried via LDAP to convert information in the PAC to other forms (specifically, to convert a SID in the PAC to a name or vice-versa).

   - The access policy is examined to verify that the authenticated user has been granted access to the specified resource.

Example domain

VSJ supports complex Active Directory environments with multiple domains including both cross-realm and cross-forest scenarios.

However, the examples in this manual illustrate a single Active Directory domain called `EXAMPLE.COM`, whose DNS domain is `example.com`.

The example application servers in this domain have the hostnames `appservhost1` (with fully qualified domain name `appservhost1.example.com`), `appservhost2` etc.

Given this:

- as a convention we will use `vsj_appservhost1` as the name of the service account for VSJ running on the host `appservhost1`
- `HTTP/appservhost1.example.com` is an SPN for `appservhost1`
- VSJ's `idm.principalAtRealm` configuration parameter should be set to the value `vsj_appservhost1@EXAMPLE.COM`

Figure 4 shows points where these names apply.

Figure 5 illustrates a simplified version of the initial process involved when a client requests a URL from a service under VSJ, showing the use of naming conventions outlined above.

Figure 4: Overview: Active Directory and VSJ configuration and terminology

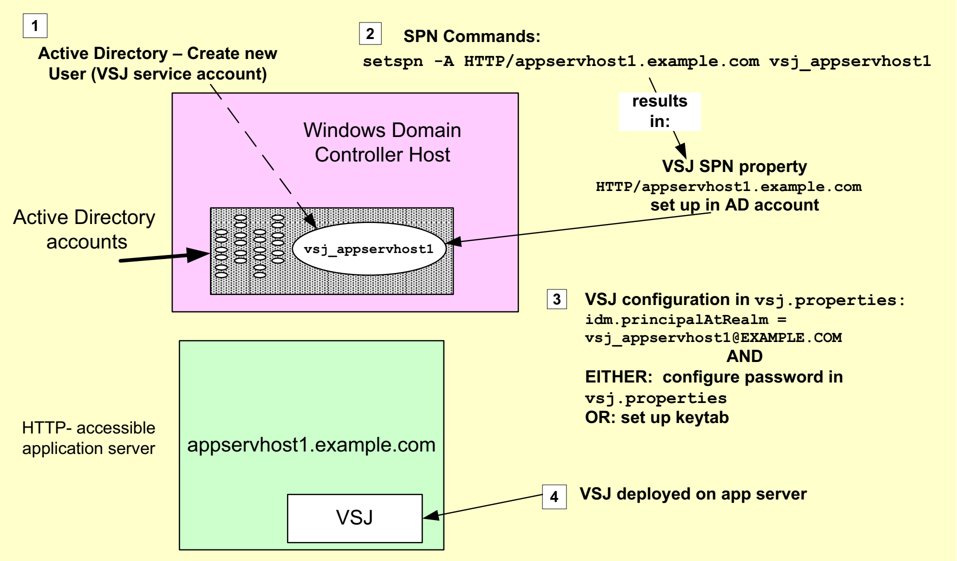
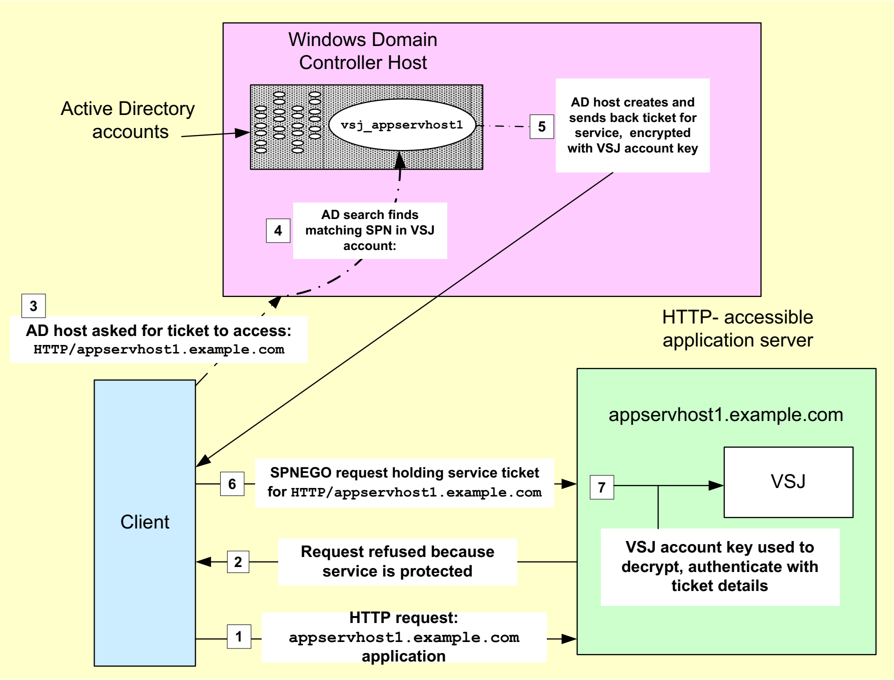

1
Active Directory – Create new User (VSJ service account)
2
SPN Commands:
setspn -A HTTP/appservhost1.example.com vsj_appservhost1
results in:
VSJ SPN property
HTTP/appservhost1.example.com
set up in AD account
Windows Domain Controller Host
Active Directory accounts
vsj_appservhost1
3
VSJ configuration in vsj.properties:
idm.principalAtRealm =
vsj_appservhost1@EXAMPLE.COM
AND
EITHER: configure password in
vsj.properties
OR: set up keytab
HTTP- accessible application server
appservhost1.example.com
VSJ
4
VSJ deployed on app server

Figure 5: Access to a URL via VSJ (simplified)


Windows Domain Controller Host
Active Directory accounts
vsj_appservhost1
5
AD host creates and sends back ticket for service, encrypted with VSJ account key
4
AD search finds matching SPN in VSJ account:
3
AD host asked for ticket to access:
HTTP/appservhost1.example.com
HTTP- accessible application server
appservhost1.example.com
Client
6
SPNEGO request holding service ticket for HTTP/appservhost1.example.com
7
VSJ
VSJ account key used to decrypt, authenticate with ticket details
2
Request refused because service is protected
1
HTTP request:
appservhost1.example.com
application

2

Preparing for VSJ

- Pre-installation overview
- Network infrastructure
- Configuring Active Directory for VSJ
- Setting up a Java application server host
- Setting up a client machine

This chapter discusses the environment needed for a VSJ deployment. It includes requirements relating to setup of Active Directory, Java application server hosts and client machines.

Pre-installation overview

Before you install VSJ successfully there are a number of conditions that must be met. You will need:

- A network architecture which:
  - provides host and client machines suitable for Active Directory operations (See Active Directory environment).
  - includes a Domain Name Service (See Domain Name Service (DNS) below), and
  - provides reliable time synchronization throughout (Time Synchronization Service).
- Installation and configuration of Active Directory on your Windows servers (Configuring Active Directory for VSJ), including Setting up the VSJ account. If you opt to allow delegation, you should also refer to the section on Enabling delegation for VSJ.
- A Java application server supported by VSJ (see the Release Notes and application-specific documentation), and relevant port access on any firewall between the application server and the Active Directory machine. For production systems, creation of a keytab file on your application server (Creating keytab files) is a recommended option.
- At least one working client capable of supporting SPNEGO or NTLM authentication (see Setting up a client machine). This may involve installation of a supported web browser (see the Release Notes), and its configuration for SPNEGO or NTLM authentication (see Browsers and authentication).

Network infrastructure

Active Directory environment

In order to work with VSJ you will need:

- An Active Directory domain.
- A host running a supported Java application server.
- A client machine joined to the Active Directory domain and with a supported web browser installed.

All machines must have access to a Domain Name Service and a Time Synchronization Service, as outlined in detail below.

Note that:

- The client should be a different host than the application server host; if they are the same, Internet Explorer will perform NTLM instead of SPNEGO.
- As general rule, you should not run VSJ on the same host as Microsoft IIS because it is difficult to configure SPNEGO to work to both of them.

Domain Name Service (DNS)

VSJ uses DNS lookups to retrieve important information about Active Directory domains and hosts, for example: a DNS SRV query for "`_ldap._tcp.EXAMPLE.COM`" to find all the domain controllers for the `EXAMPLE.COM` domain.

If you are running VSJ on a Windows machine joined to Active Directory, or on UNIX or Linux with Quest's Vintela Authentication Services (VAS), DNS should already be configured correctly.

Otherwise, check whether the DNS server that the machine is using supports SRV resource records such as:

- For locating the domain controller(s) for a given domain (EXAMPLE.COM): `_ldap._tcp.EXAMPLE.COM`
- For locating the domain controller(s) for a given domain (EXAMPLE.COM) in a given Active Directory Site (Brisbane): `_ldap._tcp.Brisbane._sites.EXAMPLE.COM`

- For locating the global catalog(s) for a given domain (EXAMPLE.COM): `_ldap._tcp.gc._msdcs.EXAMPLE.COM`
- For locating the global catalog(s) for a given domain (EXAMPLE.COM) in a given Active Directory Site (Brisbane): `_ldap._tcp.Brisbane._sites.gc._msdcs.EXAMPLE.COM`

Note: If VSJ is unable to locate the DNS servers automatically, use the `jcsi.kerberos.nameservers` system property to explicitly specify one or more of the DNS servers that VSJ should use. See Appendix A: Configuration Parameters for more information.

Time Synchronization Service

The Kerberos protocol requires that the system clocks on all machines — Active Directory domain controllers, clients, and VSJ-enabled application servers — be within the allowable Active Directory Kerberos clock skew (5 minutes by default).

Time synchronization may be provided automatically if VSJ is running either:

- on a Windows machine joined to Active Directory, or
- on a UNIX or Linux machine running VAS

Otherwise, application server clocks will need to be kept within the allowable clock skew (for example, 5 minutes) of the Active Directory domain controller.

Note: Clock drift can be particularly severe for hosts running in virtual machines.

Configuring Active Directory for VSJ

Before you deploy VSJ, you will need to have access to an Administrator account on Active Directory to establish the required VSJ-specific configuration.

Setting up the VSJ account

In order for VSJ to authenticate clients, VSJ must be represented as an object in Active Directory. There are two ways to create this object:

- If you are running VSJ on a UNIX or Linux machine that also has Quest Software's VAS (Vintela Authentication Services) product installed, you have the option of using VAS to help with the setup process for VSJ. This alternative setup method is outlined in the VSJ setup with VAS on UNIX or Linux section.
- Otherwise, setup involves configuration using the *Active Directory Users and Computers* interface and the use of Active Directory's `setspn` tool on your Active Directory domain controller.

The following sections describe the steps for setting up the VSJ account in Active Directory.

VSJ setup using Active Directory tools

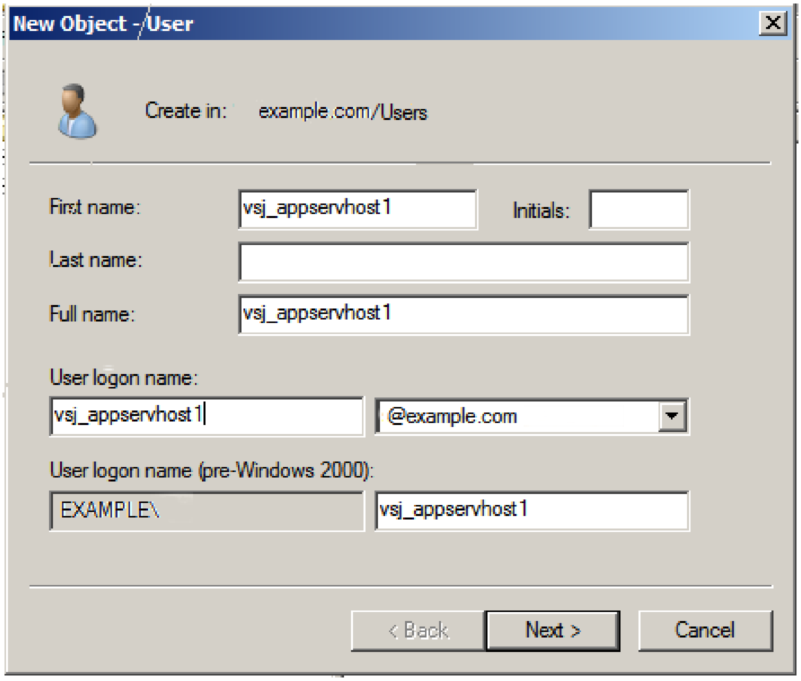
Creating a VSJ account

To create an Active Directory account for VSJ, log onto a domain controller for the Active Directory domain and perform the following steps:

1. Click the *Start* menu, point to *Programs*, and then *Administrative Tools*, and click *Active Directory Users and Computers*.
2. Click the *Users* folder to display a list of users, on the *Action* menu, point to *New*, and then click *Use*r.

   This opens the New Object-User window.

Figure 1: New Object-User window (Windows Server 2008 example)



New Object - User

Create in: example.com/Users

First name: vsj_appservhost1

Initials:

Last name:

Full name: vsj_appservhost1

User logon name: vsj_appservhost1 @example.com

User logon name (pre-Windows 2000): EXAMPLE\ vsj_appservhost1

3. Enter a name and logon name for the new service, and click *Next*.

   The user name should consist of standard alphanumeric characters and no whitespace, as it needs to be entered in a command prompt later.

4. On the next screen, enter a password for the service. Ensure that *User must change password at next logon* is **not** selected, and *Password Never Expires* **is** selected. Click *Next*, and then *Finish*.

5. Right-click the user you just entered in the *User* folder list, and then click *Properties*.

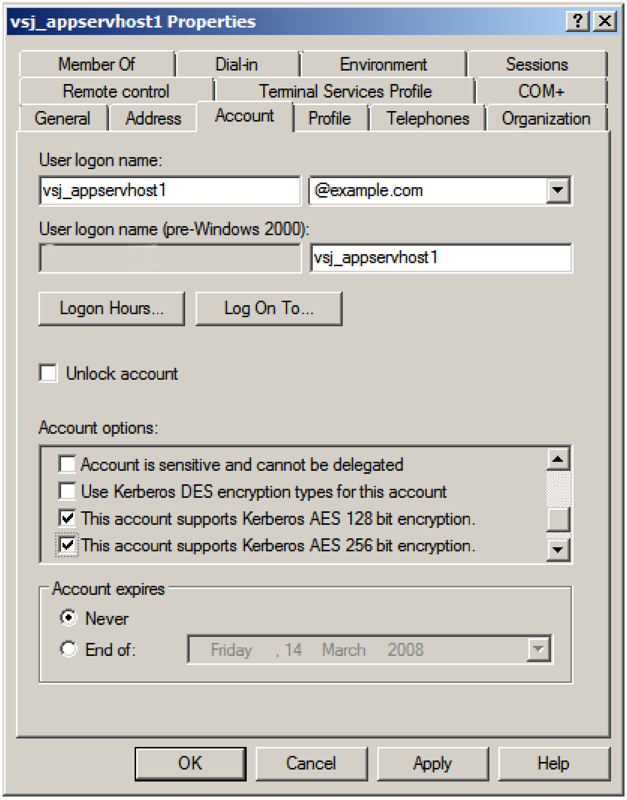
   A dialog box displays.

6. Select the *Account* tab.

7. In the *Account options* area, scroll down to review the available encryption options for Kerberos operations. (See notes on options available below).

8. When option choices are finalized here, Click *OK*.

Figure 2: Account tab for user (Windows Server 2008 example)



User logon name: vsj_appservhost1 @example.com

User logon name (pre-Windows 2000): vsj_appservhost1

□ Unlock account

Account options:

- □ Account is sensitive and cannot be delegated
- □ Use Kerberos DES encryption types for this account
- ☑ This account supports Kerberos AES 128 bit encryption.
- ☑ This account supports Kerberos AES 256 bit encryption.

Account expires

- ● Never
- ○ End of: Friday , 14 March 2008

Kerberos encryption types for Active Directory

The default Kerberos encryption type used by Active Directory is RC4.

Single DES (56 bit) encryption is available for compatibility with other Kerberos implementations, but not recommended as the preferred method.

If the Domain Controller you are configuring is running at the Windows Server 2008 domain functional level, the newer and stronger AES 256 bit and AES 128 bit Kerberos encryption types are available, and appear in your configuration panel. The Kerberos AES encryption types are not available in Windows 2000 Server and Windows Server 2003 environments.

When more than one Kerberos encryption is configured for your system, the strongest form is generally preferred. So turning on Kerberos AES 256 encryption will make it the type of choice.

In general, the recommended order of suitability and strength of Kerberos encryption types for VSJ is:

1. AES 256
2. AES 128
3. RC4
4. DES

Setting Service Principal Name (SPN) mappings

For a client (for example, Internet Explorer) to be able to authenticate to VSJ, it needs to know the user account for the VSJ service, as created in Creating a VSJ account above. A browser for example, does this by looking up a Service Principal Name (SPN) in a form like `HTTP/appservhost1.example.com`. In order for that to succeed, you must map the SPN to the VSJ account. This action is taken on your domain controller.

To create a mapping between the VSJ account and an SPN:

1. Obtain the `setspn` utility and ensure it is available on the command PATH.

   For Windows 2000, this tool is available from the Windows 2000 resource kit. For Windows Server 2003, it is included in the installation CD under `/Support/Tools/`.

   It is installed on the command PATH by default with Windows Server 2008.

   For more information on the availability and installation of this utility, check the Microsoft site at http://support.microsoft.com.

2. Launch a command prompt on your domain controller. Run `setspn` with arguments based on the following format:

   ```
   setspn -A HTTP/appservhost1.example.com vsj_appservhost1
   ```

   ```
   setspn -A HTTP/appservhost1 vsj_appservhost1
   ```

   where:

   - `appservhost1.example.com` is the fully-qualified hostname of the application server where VSJ is to be installed.
   - `appservhost1` is the unqualified hostname (short name) of the server where VSJ is to be installed.
   - `vsj_appservhost1` is the name of the user account you have previously created for VSJ.

Note: The "`setspn -A`" command does **not** check existing mappings before creating a new one, and may silently create duplicates. An error message in the form "`Server not found in Kerberos database`" may then appear if you attempt to access a duplicated mapping, as though the specified SPN doesn't exist. You will need to eliminate duplicated entries before a mapping will work.

If running a Windows Server 2008 domain, you can substitute commands in the form "`setspn -S`" or "`setspn -F -S`" for "`setspn -A`".

"`setspn -S`" checks for duplicate SPN mappings within the current domain before adding a new mapping. "`setspn -F -S`" checks over the entire forest.

SPN mapping and DNS aliases

If you use multiple hostnames to refer to the Java application server (for example, if you use name-based virtual hosting), you should use `setspn` to create an SPN mapping for each hostname involved.

For example, assume that:

- you have an application server on `appservhost1.example.com`, and
- that application server also has a DNS alias, `appservhost1alias.example.com`.

For each of those hostnames, you should map both its fully qualified domain name and its unqualified hostname (short name). If you have a DNS canonical name and one or more DNS aliases, you should set up SPN mappings both for the alias(es) and for the canonical name.

Thus, for the `vsj_appservhost1` account, you should map the following SPNs:

```
HTTP/appservhost1.example.com
HTTP/appservhost1
HTTP/appservhost1alias.example.com
HTTP/appservhost1alias
```

VSJ setup with VAS on UNIX or Linux

As an alternative to the steps outlined above, VSJ supports integration with Quest's Vintela Authentication Services (VAS) to allow you to simplify installation on VAS-enabled UNIX or Linux hosts. The following sections describe how to perform this setup.

Vintela Authentication Services (VAS)

The Vintela Authentication Services (VAS) system allows UNIX and Linux users to be authenticated using Active Directory. It provides integration with the UNIX Pluggable Authentication Modules (PAM) and Name Service Switch (NSS) systems.

A system administrator enables VAS on a UNIX host by joining it to the Active Directory domain using the `vastool` utility. This creates a computer account object in Active Directory along with a host principal and keytab that can be used to authenticate service tickets that are presented to Kerberos/VAS-enabled applications.

VAS keytabs

VAS keytab files are created in the `/etc/opt/quest/vas` directory. Each keytab file is named according to the service that uses it. For example, the host principal keys are stored in the `/etc/opt/quest/vas/host.keytab` file. VAS keytab files are stored using the standard Kerberos keytab file format and may be used by third party applications including VSJ.

vastool

`vastool` is a command line program that allows you to configure various components of VAS, access information stored in Active Directory, and perform a variety of tasks such as the creation of user accounts and keytabs.

`vastool` is located at `/opt/quest/bin/vastool`. In order to run `vastool`, you must specify `vastool` options, a command to run, and the options for that specific command.

While `vastool` supports a wide variety of commands, the following are of most use when installing VSJ with VAS or adjusting its configuration:

- `service` — manage service accounts in Active Directory
- `info domain` — display the Active Directory domain to which this host is joined
- `info site` — display the name of the local Active Directory site

Configuring VSJ to use the VAS HOST SPN

One of the simplest ways to configure VSJ to run on a VAS enabled host is to set up your configuration so that VSJ can authenticate using the HOST principal installed when you join a VAS-enabled machine to the Active Directory domain.

To do this:

1. Run the application server with sufficient permissions to access the host keytab `/etc/opt/quest/vas/host.keytab` (usually root permissions).
2. When you configure VSJ, set:
   - `idm.keytab` to the path of the VAS HOST keytab -- for example: `/etc/opt/quest/vas/host.keytab`
   - `idm.ad.site` to the Active Directory site (if any) as determined by a `vastool info site` command
   - `idm.principalAtRealm` to `HOST/appservhost1.example.com@EXAMPLE.COM`

Configuring VSJ to use a VAS HTTP service principal

It is also possible to use `vastool` to add an account for VSJ rather than using the HOST principal. The major benefit of this approach is that it allows you to run the application server as an unprivileged user.

Note: if the host is joined to a Windows Server 2008 domain, this process requires VAS version 3.3.1.48 or higher.

Steps involved are:

1. To create the service, run the following command:

```
vastool -u <Adminuser> service create
HTTP/appservhost1.example.com
```

   where

   `<Adminuser>` is a domain user with sufficient permissions to create accounts.

   This generates output similar to the following:

```
Successfully created service
HTTP/appservhost1.example.com@EXAMPLE.COM
```

   and generates the keytab:

```
/etc/opt/quest/vas/HTTP.keytab
```

2. Update the permissions on the service keytab so that the application using the service has appropriate access to it. For example, modify the permissions on

```
/etc/opt/quest/vas/HTTP.keytab
```

   so it is readable by the process running the application server.

   Thus:

```
chown appserverowner /etc/opt/quest/vas/HTTP.keytab
```

3. When you configure VSJ, set:
   - `idm.keytab` to the path of the VAS HTTP keytab created above — for example: `/etc/opt/quest/vas/HTTP.keytab`
   - `idm.ad.site` to the Active Directory site (if any) as determined by a `vastool info site` command
   - `idm.principalAtRealm` to the account created above, in a format which follows this pattern:

     `appservhost1-HTTP@EXAMPLE.COM`

Enabling delegation for VSJ

If you want to allow operations via VSJ to use delegated credentials on behalf of clients, you will need to enable delegation operations for all relevant VSJ accounts in Active Directory.

Delegation operations require that:

1. If you want a client's request to be able to use services with delegated credentials, the "Account is sensitive and cannot be delegated" option or its equivalent must be turned off on the client's account.
2. Where Constrained Delegation or Protocol Transition operations are required, the VSJ configuration parameter `idm.allowS4U` must have a value set to `true`. (See Appendix A: Configuration Parameters)

Delegation configuration in different systems

Depending on the functional level of your domain, there are various delegation options which may be available and presented to you for configuration.

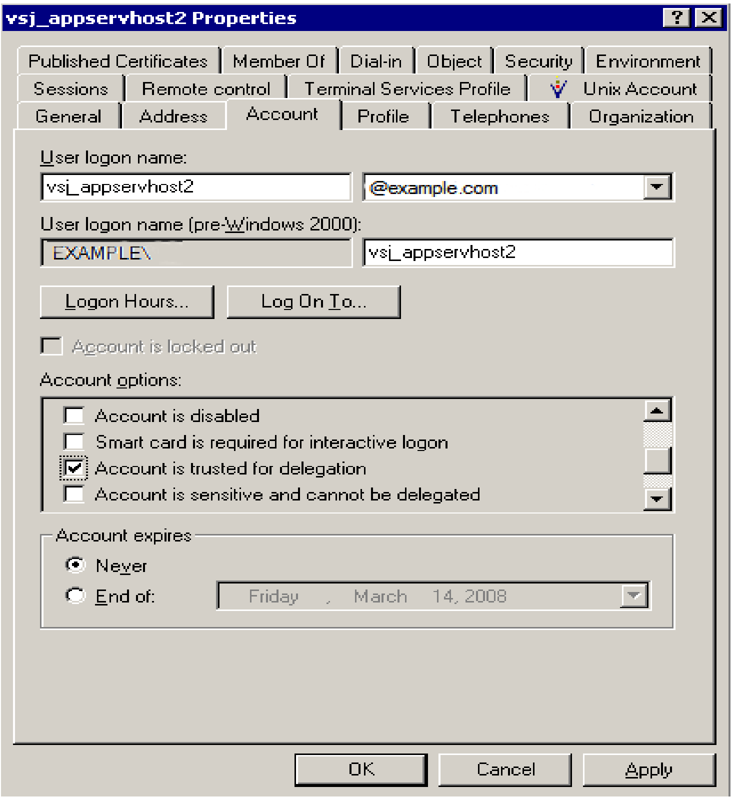
Windows 2000 Server delegation for VSJ account

Under Windows 2000 Server, a delegation option can be configured using a relatively simple toggle in the *Properties* pane of the VSJ account*.*

The "*Account is trusted for delegation*" option, if selected, means that the VSJ account can be used for *Unconstrained* delegation. Otherwise, no delegation occurs.

For access to delegation configuration in Windows 2000 Server, right click the VSJ account in the Active Directory tree, and select *Properties* — then click the *Account* tab and make selections in the *Account options* area:

Figure 3: Delegation configuration for Windows 2000 Server



User logon name:
vsj_appservhost2 @example.com

User logon name (pre-Windows 2000):
EXAMPLE\ vsj_appservhost2

□ Account is locked out

Account options:

□ Account is disabled
□ Smart card is required for interactive logon
☑ Account is trusted for delegation
□ Account is sensitive and cannot be delegated

Account expires

● Never
○ End of: Friday , March 14, 2008

Note that the "*Account is sensitive and cannot be delegated*" option in this screen has no application to the VSJ account and no effect in configuring it (as contrasted with the same setting in a client account).

Windows Server 2003 and Windows Server 2008 delegation

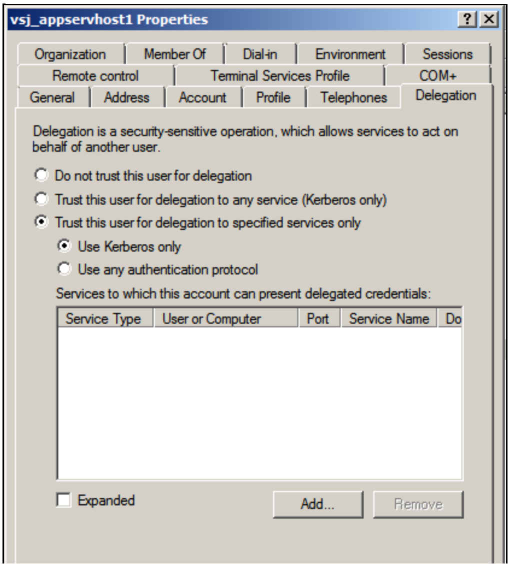
Before you can configure delegation options for these platforms, you must have set up at least one SPN mapping in the account. Option choices for delegation configuration are not available in the Windows interface until a mapping has been set.

Delegation options here are more extensive than for Windows 2000 Server. They are configured from the separate *Delegation* tab in the *Properties* pane for the VSJ account. Options here include:

- **Delegation Disallowed** ("*Do not trust this user for delegation*") — VSJ is not to be trusted for delegation at all. If you select this option, any existing (separate) delegation setting is toggled off.
- **Delegation Allowed (for all services) (**"*Trust this user for delegation to any service (Kerberos only)*"**)** — the *Unconstrained* delegation option for Kerberos operations, using a TGT.
- **Delegation to specified services only** ("*Trust this user for delegation to specified services only*") — *Constrained* delegation (*S4U2Proxy*) option involving service tickets.

  The *Constrained* delegation choice has further configuration sub-options to either permit the *Protocol transition* (*S4U2Self*) extension ("*Use any authentication protocol*"), or exclude it ("*Use Kerberos only*").

Figure 4: Configuration options in Delegation tab



Delegation is a security-sensitive operation, which allows services to act on behalf of another user.

○ Do not trust this user for delegation

○ Trust this user for delegation to any service (Kerberos only)

● Trust this user for delegation to specified services only

● Use Kerberos only

○ Use any authentication protocol

Services to which this account can present delegated credentials:

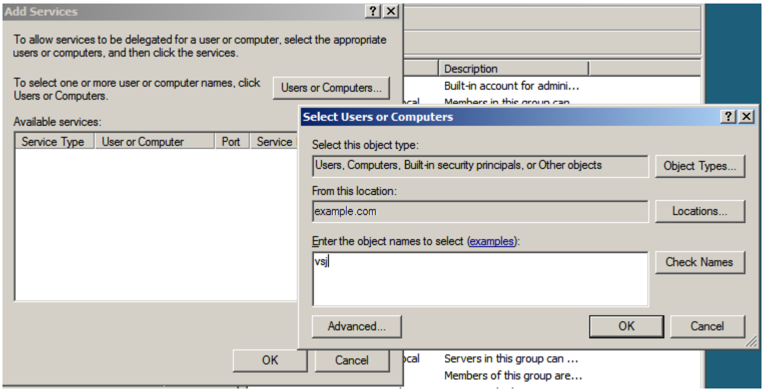
Service Type	User or Computer	Port	Service Name	Do

□ Expanded

The first two options under the *Delegation* tab work the same way as their equivalents in Windows 2000 Server, and if selected, require no further configuration action.

The *Constrained delegation* option however, requires further action: to select all the Active-Directory services and computers to which the VSJ account is permitted to delegate. To start this process, press the *Add* button. An *Add Services* window opens, holding a list box for *Available Services*. Above that, a *Users and Computers* choice leads to a further search and selection window (*Select Users or Computers*).

Figure 5: Finding and adding services for delegation



Add Services

To allow services to be delegated for a user or computer, select the appropriate users or computers, and then click the services.

To select one or more user or computer names, click Users or Computers.

Available services:

Service Type	User or Computer	Port	Service

Select Users or Computers

Select this object type: Users, Computers, Built-in security principals, or Other objects

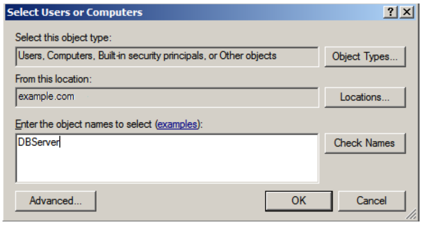
From this location: example.com

Enter the object names to select (examples): vsj

Users or Computers...
OK
Cancel
Object Types...
Locations...
Check Names
Advanced...
OK
Cancel
Description
Built-in account for admini...
Members in this group can...
Servers in this group can ...
Members of this group are...

The search panel *Check Names* button permits a "begins with..." type of name searching, while an *Advanced* button opens a sub-panel with more complex facilities for pattern search options.

Figure 6: Nominating a host with services to delegate to



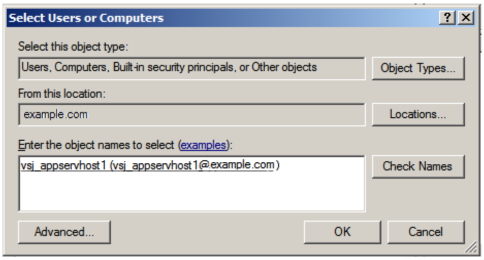
Select Users or Computers

Select this object type: Users, Computers, Built-in security principals, or Other objects

From this location: example.com

Enter the object names to select (examples): DBServer

Figure 7: Selecting a VSJ account object for its mapped services



Select Users or Computers

Select this object type:
Users, Computers, Built-in security principals, or Other objects

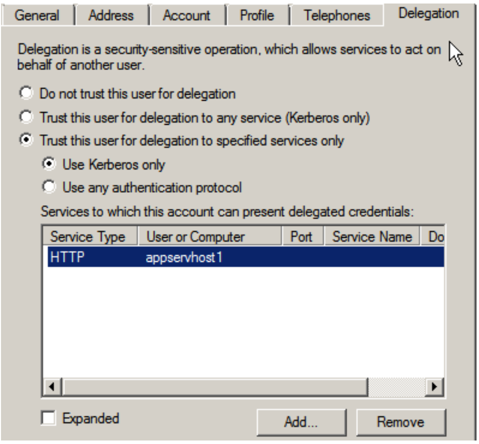
From this location:
example.com

Enter the object names to select (examples):
vsj_appservhost1 (vsj_appservhost1@example.com)

Select an object name in the *Select Users or Computers* window, and press *OK*.

You are returned to the *Add Services* window, populated with a listing of the services which are available for you to delegate to on the computer host or VSJ account chosen. Make a selection from the *Add Services* list, and press *OK*, to return to your *Delegation* tab in the *Properties* window, where your selections have been collected in a list panel.

Figure 8: Collected services list



Delegation is a security-sensitive operation, which allows services to act on behalf of another user.

○ Do not trust this user for delegation
○ Trust this user for delegation to any service (Kerberos only)
● Trust this user for delegation to specified services only
  ● Use Kerberos only
  ○ Use any authentication protocol

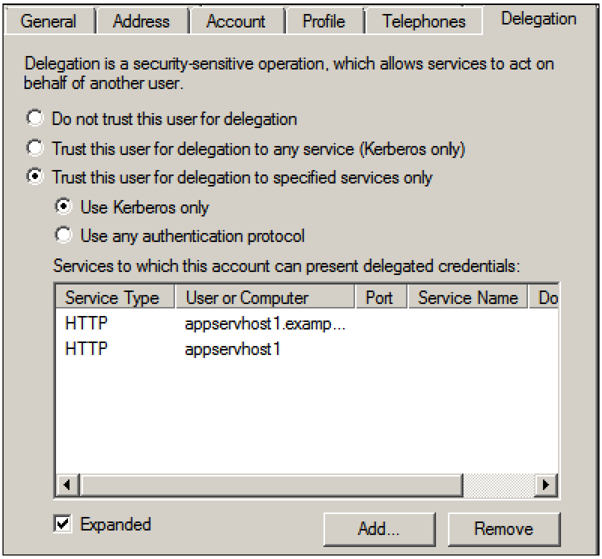
Services to which this account can present delegated credentials:

Service Type	User or Computer	Port	Service Name	Do
HTTP	appservhost1			

□ Expanded

Important Note: If you have configured more than one SPN for a service (for example, using both a fully-qualified hostname and an unqualified hostname), the list in the panel initially displays only one instance of that service, even though both are actually selected. To view all instances selected, set the* Expanded *option in this window. (See Figure 9)

Figure 9: Expanded selection of services for delegation



Delegation is a security-sensitive operation, which allows services to act on behalf of another user.

○ Do not trust this user for delegation

○ Trust this user for delegation to any service (Kerberos only)

● Trust this user for delegation to specified services only

- ● Use Kerberos only
- ○ Use any authentication protocol

Services to which this account can present delegated credentials:

Service Type	User or Computer	Port	Service Name	Do
HTTP	appservhost1.examp...			
HTTP	appservhost1			

☑ Expanded

If you need to, you can remove one or more services from this display of selected services. When satisfied that your selected list is correct, press *OK* to finalize your configuration.

Setting up a Java application server host

See the release notes (HTML) for a comprehensive list of application servers supported by VSJ. It is presumed here that you have installed one such server using standard procedures laid down for the server concerned.

If your application server and Active Directory are separated by a firewall, you need to open the following ports on the firewall to allow Kerberos to work properly:

- Ticket requests: (88/TCP, 88/UDP),
- LDAP to Directory Service: (389/TCP, 389/UDP),
- DNS: (53/TCP, 53/UDP)
- LDAP to Global Catalog (3268/TCP)

Creating keytab files

VSJ supports two mechanisms for storing key information required to authenticate users — setting a password and using keytabs. In general, use of keytabs is to be preferred for production systems, but for ease of initial setup and evaluation, the password approach is probably more convenient.

See Keytabs and passwords for more information on each method.

To create a keytab file:

First, generate keytab files for each service that requires authentication. This can be done with the tools included with the VSJ release (see Appendix B: Using the JKTools). From the `bin/` directory of your VSJ distribution, run the `jktutil` tool and take the following steps:

1. First, create keytab entries using the RC4 and DES encryption types. The commands below will create keytab entries with a key version number of 255 (a value which acts as a wildcard):

   ```
   jktutil (type '?' for help): add_entry -password -p
   vsj_appservhost1@EXAMPLE.COM -k 255 -e rc4-hmac
   Password for vsj_appservhost1@EXAMPLE.COM:
   jktutil (type '?' for help): addent -password -p
   vsj_appservhost1@EXAMPLE.COM -k 255 -e des-cbc-crc
   ```

```
Password for vsj_appservhost1@EXAMPLE.COM:
```

2. If the domain controller is running at Windows Server 2008 functional level, you also need to create keytab entries using the AES encryption type.

   ```
   jktutil (type '?' for help): add_entry -password -p
   vsj_appservhost1@EXAMPLE.COM -k 255 -e aes256-sha1
   Password for vsj_appservhost1@EXAMPLE.COM:
   jktutil (type '?' for help): add_entry -password -p
   vsj_appservhost1@EXAMPLE.COM -k 255 -e aes128-sha1
   Password for vsj_appservhost1@EXAMPLE.COM:
   ```

3. Write the created keytab to a file and quit `jktutil`.

   ```
   jktutil (type '?' for help): write_kt
   -o vsj_appservhost1.keytab
   jktutil (type '?' for help): quit
   ```

To confirm the contents of the created keytab, run the `jklist` tool with the arguments `-e -k vsj_appservhost1.keytab`. The output should look similar to this:

```
Keytab name: FILE:vsj_appservhost1.keytab
KVNO Principal                    EncType
---- ---------------------------- -----------
255 vsj_appservhost1@EXAMPLE.COM rc4-hmac
255 vsj_appservhost1@EXAMPLE.COM des-cbc-crc
255 vsj_appservhost1@EXAMPLE.COM aes256-cts-hmac-sha1-96
255 vsj_appservhost1@EXAMPLE.COM aes128-cts-hmac-sha1-96
```

To use the keytab file with VSJ:

1. When configuring an application for VSJ you need to set the `idm.keytab` parameter (See Appendix A: Configuration Parameters).

   The `idm.keytab` parameter value should point to the keytab file you generated for this service when you set up the application server.

2. Copy the generated keytab to the location specified by the `idm.keytab` parameter.

The new keytab contains sensitive information that could be used to subvert the security of your VSJ installation. Ensure that you set appropriate access permissions on this file. Do not send it over unsecured networks unencrypted.

Setting up a client machine

Operating System

To support Windows Integrated Authentication, the client must be part of the Active Directory domain and must have one of the following operating systems installed:

- Windows 2000 Professional
- Windows 2000 Server
- Windows 2000 Advanced Server
- Windows XP Professional
- Windows Server 2003
- Windows Vista
- Windows Server 2008

These are the only operating systems that incorporate Active Directory.
If Microsoft Windows 2000 is installed on the client, Internet Explorer 5.5 or higher is also needed.

To support NTLM authentication, one of the following operating systems must be installed on the client:

- Windows 95
- Windows 98
- Windows Millennium Edition (ME)
- Windows NT
- Windows XP Home

These operating systems do not incorporate Active Directory, but do support NTLM authentication.

For more information on NTLM, see Security Issues.

Basic fallback is supported on most platforms and browsers, such as Mozilla® and Konqueror™.

Browsers and authentication

To support Windows Integrated Authentication, you need one of the following browsers:

- Internet Explorer 5.5 or higher
- Mozilla Firefox

When using a browser that is not configured for Windows Integrated Authentication, VSJ provides both an NTLM mechanism that allows NTLM SSO, and a fallback mechanism that allows a user to log in with their Kerberos password using standard HTTP basic authentication.

However, the basic feature is disabled by default for security reasons. For more details on the consequences of enabling basic fallback see Security Issues.

Setting up Internet Explorer for SSO

This section describes how to set up Internet Explorer for SSO. This includes:

- Windows Integrated Authentication
- NTLM authentication
- Troubleshooting your Internet Explorer configuration

Windows Integrated Authentication

To take full advantage of VSJ, you must enable Windows Integrated Authentication. To use Windows Integrated Authentication, clients must be running Internet Explorer 5.5 or higher on Windows 2000/2003/2008/XP. You should also ensure that the "High Encryption Pack" is installed. For details, search the site at http://www.microsoft.com.

You can confirm that the High Encryption Pack is installed by selecting *About Internet Explorer* from the Help menu in Internet Explorer.

If it is installed, "Cipher Strength: 128-bit" is shown on the *About Internet Explorer* dialog box.

You probably need to ensure that the Java EE server on which you installed VSJ is in the Intranet Zone. See Troubleshooting your Internet Explorer configuration for more information.

Windows Integrated Authentication is enabled by default on Internet Explorer 5.5. If you are using Internet Explorer 6.0 or higher you also need to do the following:

1. In Internet Explorer, on the *Tools* menu, click *Internet Options*.
2. Click the *Advanced* tab.
3. Scroll down to the *Security* section, and select the *Enable Integrated Windows Authentication (requires restart)* setting.
4. Click *OK*.
5. Restart Internet Explorer.

NTLM authentication

Windows Integrated Authentication provides a greater degree of security than NTLM authentication, so we recommend that users should attempt to use Windows Integrated Authentication unless it is unsupported by their client operating system or browser. To use NTLM for SSO, you need a version of Windows that supports NTLM challenge/response (see Operating System).

To set up Internet Explorer for NTLM authentication, configure the intranet for authentication:

1. In Internet Explorer, on the *Tools* menu, click *Internet Options*.
2. Click the *Security* tab.
3. Click the Local intranet icon and then click Custom Level....
4. Scroll down the *Security Settings* list to the User Authentication section, and select the *Automatic logon only in Intranet zone* option. This configures the intranet for SSO.
5. Click *OK*.

Troubleshooting your Internet Explorer configuration

To use Internet Explorer for Windows Integrated Authentication you must ensure that the Java EE server on which you installed VSJ is in the Intranet Zone (on the Internet Explorer Security tab). This should be the case for most default setups, but may not be if one of the following is true:

- The domain name of the Java EE server on which VSJ is installed is different from the domain name of your desktop machine.
- You have modified the *Intranet Zone* settings.

If your server is not part of the Intranet Zone then Windows Integrated Authentication will fail and a login dialog box will display.

If this occurs, perform the following steps:

1. In Internet Explorer, on the *Tools* menu, click *Internet Options*.
2. Click the *Security* tab.
3. Click the *Local intranet* icon and then click *Sites*.
4. Click *Advanced*.
5. Enter the name of your server, and then click *Add*.
6. Click *OK*.

If you are sure that your server is part of the Intranet Zone, but a login dialog box is still displayed, do the following to check that Windows Integrated Authentication is being used in the Intranet Zone.

1. In Internet Explorer, on the *Tools* menu, click *Internet Options*.
2. Click the *Security* tab.
3. Click the *Local intranet* icon, and then click *Custom Level*....
4. Scroll down the *Security* Settings to the *User Authentication* section, and then select the *Automatic logon only in Intranet Zone* option.

If Windows Integrated Authentication is still failing, you may have incorrectly set up the SPN for VSJ.

See Maintenance and Troubleshooting for other possible problems.

3

Deploying VSJ

- **Getting started with VSJ**
- **VSJ and your web applications**
- **Setting up logging**
- **Controlling access to resources**

This chapter describes how to get started with VSJ using the supplied examples, and moves on to cover how to build SSO solutions using VSJ-protected servlets and JSPs.

Getting started with VSJ

To obtain a license for VSJ

This distribution requires a VSJ license and does not include one; you must supply one yourself, as described below.

The license is packaged as a JAR file so that it can be installed in the same way as VSJ's other JAR files. The name of the license JAR file does not matter, only its contents, but older releases of VSJ (and JCSI SSO) generally called this file `jcsi_license.jar` whereas newer releases such as this one call the file `vsj-license.jar`; the two filenames are interchangeable.

If you already have a production license for VSJ, you may continue to use that.

Otherwise, evaluation licences are available by filling out the VSJ evaluation download form at:

http://www.vintela.com/vintela-single-sign-on-for-java-evaluation

You will receive an email message that includes the license jar file as an attachment.

Note: some email clients rename the attachment to `vsj-license.zip`; in this case you may have to re-rename the saved file to `vsj-license.jar`

Once you have a current license jar, add it to the `lib/` directory alongside the other VSJ libraries.

HTTP header size limits

Some Java application servers limit the maximum size of an HTTP header to a value too small for some *HTTP Negotiate* authentication headers generated by Internet Explorer/Active Directory, especially those for users with a large number of group memberships.

Refer to the *Third Party Known Issues* section in the Release Notes for details specific to your application server.

Alternatively, you can try reducing the size of the tokens the client is sending by configuring Active Directory to exclude extraneous data from the tickets it gives to the client with one or both of the following:

1. Disable delegation trust for the web server. When a service account is trusted for delegation, Active Directory includes a copy of the user's TGT in the service ticket (token). Disabling the delegation trust stops that, making the service tickets smaller. Alternatively, if it is available, you can enable Constrained delegation (S4U2Proxy) and have VSJ's support in using it.
2. Disable PAC inclusion for the HTTP service account. The PAC contains pre-computed authorization information and can also be large under certain circumstances. An update which allows you to set a flag to handle this is available. See KB 832572 for further details at http://support.microsoft.com, or search there for `NO_AUTH_REQUIRED`.

Configuring the VSJ examples

The distribution contains a number of examples demonstrating how to use VSJ.

You should test VSJ examples in your local environment before attempting to deploy a production system. Start with the '`simple`' example as a bare minimum, but ensure you also test others as appropriate to your needs.

To configure the examples, edit the sample `vsj.properties` file in the `examples/` directory.The `vsj.properties` file is used as the configuration for all examples.

For details of all VSJ parameters, see Appendix A: Configuration Parameters.

For the examples, you will need configure the following:

`idm.principalAtRealm`

This parameter should be set to the fully qualified name (including the Active Directory domain) of the VSJ service principal you set up in (see Setting up the VSJ account) -- for example, `vsj_appservhost1@EXAMPLE.COM`.

`idm.password / idm.keytab`

If you have a keytab file, use `idm.keytab` to specify the filesystem pathname of the keytab file. You may have a keytab file if you used `jktutil` to create one (see Creating keytab files) or if you are using VSJ with VAS (see VSJ setup with VAS on UNIX or Linux).

Otherwise, use `idm.password` to specify the password that you set when you created the VSJ account (see Creating a VSJ account).

See Keytabs and passwords for information on the security implications of each method.

`idm.allowUnsecured`

Specifies whether to allow authentication over HTTP (rather than requiring HTTPS).

Refer to SPNEGO/Windows Authentication for security implications and recommendations.

The default setting is false.

`idm.ad.qualifyUserPrincipal`

Fully qualify the authenticated user name returned by VSJ by appending the Active Directory domain name.

For example, by default VSJ returns the authenticated user name in the format `username`.

With this parameter set to true, VSJ returns `username@EXAMPLE.COM`

The default is false

`idm.ad.site`

The name of the local Active Directory site. Setting this parameter is good for efficiency in large networks that have multiple Active Directory Sites.

`idm.allowNTLM`

Specifies whether to allow fallback to NTLM authentication.

The default is false.

Once these parameters are configured, you can build and deploy the examples. See the `README` files in each example's individual directory for more specific details.

VSJ and your web applications

VSJ is designed to bring SSO capabilities to Java EE Web components. These supported components are Servlets and Java Server Pages (JSP) conforming to Servlet 2.3 specification. This section details the application of VSJ to both Servlets and JSPs, and the different approaches required, depending on the Servlet specification.

Deployment options

To set up web applications for SSO using VSJ, you must set up your application server so that it can find VSJ 's libraries (in the `lib/` directory), and its required thirdparty libraries (in the `thirdparty/lib` directory).

There are several options to consider when deploying VSJ:

- Deploying in a web application
- Deploying on the CLASSPATH
- Deploying in application server specific path

The recommended approach for development and testing is to deploy VSJ into the web application itself.

Deploying in a web application

If you are deploying just a single web application it is possible to deploy all the necessary libraries in the `WEB-INF/lib` directory of the Web application.

This method has the benefit of being compact. All the necessary libraries are contained in the web applications distribution or war file and no special access to the Web Container and filesystem are necessary.

A drawback of this deployment method is that this makes the `war` file large and that other applications cannot share these libraries, so to deploy a second application, the libraries must be copied into the `WEB-INF/lib` of this application too.

The libraries are loaded for each application which increases the memory footprint of the Web Container and using more resources such as file descriptors, etc.

Deploying on the CLASSPATH

Another deployment method is to set the required jars into the *CLASSPATH* variable before starting the Web Container. This allows the Web Container to take control of the specified jars without changing the JDK.

```
    C:\> set CLASSPATH=file1.jar;file2.jar;...
    C:\> startWebContainer.bat
```

This method is dependent on the Web Container that you are using and therefore may not be available.

A drawback of this method is that you must remember to set the variable before starting the Web Container. On the other hand, one benefit is the ability to switch between versions of libraries quickly by resetting the *CLASSPATH* variable and restarting the Web Container.

Some Web Containers also allow you set additional options for the JDK or JRE by including the `JAVA_OPTIONS` variable in the start command. The following example shows the VSJ Kerberos debugging being enabled under Windows.

```
    C:\> set JAVA_OPTIONS=-Djcsi.kerberos.debug=true
    C:\> startWebContainer.bat
```

Deploying in application server specific path

Some Web Containers have their own deployment methods for common library jars. For example the Apache Tomcat servlet container allows additional libraries to be placed in the common/lib directory.

Refer to your Web Container documentation for more information on the installation and deployment of common libraries and the different variables that can be used to set options.

Creating a deployment descriptor for SSO

This section describes how to build deployment descriptors to take advantage of VSJ.

Deploying SSO web components

The procedure for deploying Web components on an application server requires no programming, and is the same for both Servlets and JSPs. This is because it uses the concept of filters. These filters are used to filter incoming requests for data which they can use before forwarding the request to the intended recipient.

VSJ provides a filter called AuthFilter which is responsible for filtering authentication data from requests, and then authenticating the sender before forwarding the request onto Servlets and JSPs. The first step in adding the filter is to declare a mapping of it to the Servlet/JSP for which it authenticates:

```
<filter-mapping>
  <filter-name>authFilter</filter-name>
  <servlet-name>SimpleServlet</servlet-name>
</filter-mapping>
```

This addition to the deployment descriptor creates a new mapping of a filter called authFilter to an ordinary Servlet named SimpleServlet (the Simple example is included in the example/simple/ directory of your installation). The authFilter (as yet undefined) is to be responsible for filtering any incoming HTTP requests to the SimpleServlet. This same mapping can be done for JSPs as well:

```
<filter-mapping>
  <filter-name>authFilter</filter-name>
  <url-pattern>/test/index.jsp</url-pattern>
</filter-mapping>
```

Now that the filter mapping has been set up to intercept the HTTP requests to your Web component, you can expand the filter to perform the authentication. To define the new filter you must add the following XML fragment:

```
<filter>
  <filter-name>authFilter</filter-name>
  <filter-class>com.wedgetail.idm.sso.AuthFilter</filter-class>
</filter>
```

This addition declares a new filter called authFilter which is an instance of the VSJ filter com.wedgetail.idm.sso.AuthFilter. Now all requests to a Web component to which authFilter is mapped, pass through the VSJ AuthFilter.

Configuring the VSJ parameters

VSJ requires a number of parameters so that it can validate credentials and ensure the security of the communications. This can be done by either entering the configuration parameters and their values in the vsj.properties file (see examples/vsj.properties for an example properties file), as initialization parameters (<init-param>) or as context parameters (<context-param>).

By default, VSJ looks for a vsj.properties file on the classpath or in the WEB-INF/ directory of your web application. Alternatively, you can specify the URL location of a properties file using the idm.propertyFileURL configuration parameter. For example:

```
<filter>
```

```
    <filter-name>authFilter</filter-name>
    <filter-class>com.wedgetail.idm.sso.AuthFilter</filter-class>
    <init-param>
      <param-name>idm.propertyFileURL</param-name>
      <param-value>file:///C:/vsj.properties</param-value>
    </init-param>
  </filter>
```

Using initialization parameters is particularly good for binding specific credential validation information to individual servlets and filters since they are embedded in Web component definitions. For example, defining a principal for an authentication filter can be done as follows:

```
<filter>
   <filter-name>authFilter</filter-name>
   <filter-class>com.wedgetail.idm.sso.AuthFilter</filter-class>
   <init-param>
      <param-name>idm.principalAtRealm</param-name>
      <param-value>vsj_appservhost1@EXAMPLE.COM</param-value>
   </init-param>
</filter>
```

This binds the defined principal to one filter only.

On the other hand, context parameters are valid and visible for all Web components defined in a deployment descriptor. The following shows an example of a principal being defined in a context parameter:

```
<web-app>
  <context-param>
      <param-name>idm.principalAtRealm</param-name>
      <param-value>vsj_appservhost1@EXAMPLE.COM</param-value>
   </context-param>
</web-app>
```

See Configuration Parameters for a detailed list of all VSJ configuration parameters.

Building a war file for SSO

Once you have coded your Web components and set up an SSO deployment descriptor, you can prepare the Web components for deployment. As with normal Web components, you need to create a Web Application Archive (WAR) file to maintain a standardized structure.

You can then install the VSJ-configured WAR using the appropriate deployment mechanism for your application server.

Setting up logging

VSJ includes a configurable logging package. This mechanism allows you to specify any logging mechanism that meets VSJ's requirements.

By default, VSJ uses Apache Commons Logging and relies on its autoconfiguration behavior. Apache Commons Logging delegates to some other logging package, and again VSJ relies on the autoconfiguration behavior of that logging package. The VSJ distribution provides an Apache Log4J configuration which should meet most needs.

To set up the logging mechanism, you must setup a `log4j.properties` or `log4j.xml` configuration file to be included in the `WEB-INF/classes` directory of your web application.

This log4j configuration should set the **com.dstc** and **com.wedgetail** loggers to use the log levels and appenders you desire.

```
# Set VSJ Kerberos logger priority to DEBUG and its only appender to
# FILE.
log4j.logger.com.dstc=WARN, FILE

# Set VSJ SSO logger priority to DEBUG and its appenders to CONSOLE
# and FILE.
log4j.logger.com.wedgetail=DEBUG, FILE, CONSOLE
```

You can also configure the `com.wedgetail.idm.sso.util.DefaultAuditor#login` and `com.wedgetail.idm.sso.util.DefaultAuditor#access` loggers to turn on VSJ auditing for logins and page accesses respectively:

```
# Turn on VSJ login auditing and log messages to FILE.
log4j.logger.com.wedgetail.idm.sso.util.DefaultAuditor#login=INFO,
FILE
# Turn on VSJ access auditing and log messages to FILE.
log4j.logger.com.wedgetail.idm.sso.util.DefaultAuditor#access=
INFO, FILE
```

For more information on configuring log4j default appenders or on creating new appenders, see: http://jakarta.apache.org/log4j/docs/manual.html.

As an alternative, you can set `idm.logger.name` (and optionally also `idm.logger.props`). In this case VSJ invokes and configures Log4J directly.

The parameter `idm.logger.name` is the Log4J logger name to be used. This can be any text as long as it is unique to a log file (that is, no other log file for any other web application can share the same name).

The name must also be specified in the Log4J properties file.

The parameter `idm.logger.props` is used to configure logging with a specified Log4J properties file. The value of this parameter is interpreted as follows:

- null - no logging output. This is equivalent to not specifying the `idm.logger` parameter at all.
- “BASIC” - This value indicates that basic error logging is required. All error messages will be sent to the standard output of the container. For example, if a servlet is executed under an Apache Tomcat Web server, error messages would be written to the `logs/catalina.out` file.
- Any other value (such as `error-log.properties` above) indicates the location of a Log4J properties file. This properties file is located in the `WEB-INF` directory with the deployment descriptor, and is used to configure the logger that VSJ uses for reporting errors and debug messages.

Controlling access to resources

This section shows how to use Active Directory to control access to resources in your Web application. It describes how to set up authorization using Active Directory groups, and how to write access policy files for VSJ

Authorization using Active Directory groups

VSJ supports authorization of users using Active Directory groups. This allows a deployer to specify which groups are allowed to access a given application at deployment time, and then manage membership of this group using Active Directory without redeploying the application.

This section outlines the authorization model for Servlets and JSPs in Java EE, and discusses how this maps to the authorization configuration used by VSJ.

Java EE authorization model for servlets/JSPs

Java EE provides an authorization model for servlets and JSPs that are *role*-based.

A role is a collection of users or groups of users defined by the application server container.

Membership of one or more roles is used by the container to determine whether a particular user should be allowed to access a Web resource.

Java EE provides two mechanisms for enforcing role-based access control:

1. *Declarative security.* The application deployer defines the roles and associates them with the resources that they wish to protect at deployment time.

   This is done by associating a *security constraint* with the resources you wish to protect in the application's deployment descriptor.

   For example, supposing you want to restrict access to particular resources in an order/inventory system to members of the role "StockManager".

   You could do this with the following XML fragment from the deployment descriptor:

```
<security-constraint>
  <web-resource-collection>
```

```
     <web-resource-name>InventoryReports</web-resource-name>
     <description>
     Security constraint for restricting access to
Inventory reports.
     </description>
     <url-pattern>/inventory/reports/*</url-pattern>
     <http-method>POST</http-method>
     <http-method>GET</http-method>
  </web-resource-collection>
  <auth-constraint>
     <description>
     Access only available to StockManagers
     </description>
     <role-name>StockManager</role-name>
  </auth-constraint>
  <login-config>
     <auth-method>BASIC</auth-method>
  </login-config>
</security-constraint>
```

The `<security-constraint>` tag has three main components:

- The `<web-resource-collection>` tag defines those resources you want to protect. It can include one or more URL patterns to match (specified by the `<url-pattern>` tag), and the HTTP methods to which the security constraint should apply (specified by the `<http-method>` tag, or all methods if the tag is omitted).

  Note that the URL pattern specified should exclude the servlet-context.
- The `<auth-constraint>` tag defines which roles are allowed access to the resources matched by the `<web-resource-collection>` definition. These roles are defined by application server-specific mechanisms outside of the standard deployment descriptor. For example, in BEA WebLogic these roles are defined in the weblogic.xml file using a `<security-role-assignment>` tag.
- The `<login-config>` tag defines how users are authenticated by the container to perform authorization. The example uses standard basic authentication support which uses a user name and password.

  When a user attempts to access the resources in the security constraint, the container first checks the authentication method in the `<auth-method>` tag to determine how the user is to be authenticated. Once they have authenticated successfully, the container examines the `<auth-constraint>` to see which roles are required for the user to be able to access the resource.

If the user is a member of the role, access is granted, otherwise an Access Denied error is displayed.

2. *Programmatic security.* Because the declarative security model only allows for very simple access control based on role membership, Java EE supports a programmatic enforcement mechanism that allows the application developer to implement more sophisticated rules. The basic element used is the `isUserInRole` method of the `HttpServletRequest` interface.

   This method allows the programmer to determine a user's role membership at run time and to take actions based on it.

   For example, supposing you wanted a Servlet to display different content depending on whether they were a member of the "Supplier" role or not, the following logic could be used:

```
HttpServletRequest req = ...
// Check if user is a "Supplier"
if (req.isUserInRole("Supplier")) {
  // Display one set of content
     ...
} else {
  // Display some other content
}
```

   Programmatic security provides much more flexibility in authorization decisions. Constraints can be based on information other than simple role membership (for example time of day, or the source address from which the request originated).

   The main disadvantage of programmatic security is that it requires these decisions to be hard-coded at development time. Deployers also have a slightly more difficult task, as you need to know which roles the programmer is relying on and ensure that these are configured correctly at deployment time.

   When deploying applications with programmatic security, you may still specify a security constraint in the deployment descriptor. However, you may also have to perform a role mapping to ensure that roles defined by the application server map to role names that the programmer uses. This is often used where one or more roles with different members are required, but programmers have used the same names.

   This role mapping is done using the `<security-role-ref>` tag of the servlet descriptor that is enforcing the programmatic security and linking the role to another role defined by the `<security-role>` tag.

   For example:

```
<servlet>
```

```
  <servlet-name>DisplayInventory</servlet-name>
  <servlet-class>DisplayInventoryServlet</servlet-class>
  <security-role-ref>
     <role-name>Supplier</role-name>
     <role-link>InventoryApplicationSupplier</role-link>
  </security-role_ref>
</servlet>
<security-role>
  <description>
  Suppliers who have access to the Inventory Application
  </description>
  <role-name>InventoryApplicationSupplier</role-name>
</security-role>
```

This defines a mapping between the "Supplier" role name used by the call to `isUserInRole` and the role `InventoryApplicationSupplier`. The `InventoryApplicationSupplier` role still needs to be defined using the application server-specific mechanisms described previously.

VSJ authorization

VSJ supports similar declarative and programmatic authorization models to the standard Java EE framework. Furthermore, VSJ allows you to define authorization in terms of Active Directory groups and principals, allowing centralized administration of authorization using Active Directory. VSJ also provides access to other information related to Active Directory user accounts such as their full name, last login time, number of bad password attempts since last successful login and a list of the groups to which they belong.

Compatibility with standard Java EE deployment descriptors

Because each application server container enforces its security constraints in a different and non-standardized way, it is not possible to reuse the existing security constraint mechanism in the Java EE deployment descriptor.

Instead, VSJ allows you to reuse the XML fragments in the standard deployment descriptors to specify the authorization policy in a separate XML file. This provides a compromise between tight container integration, and the ability for VSJ to support a wide range of Java application server platforms.

For this reason, when configuring authorization for VSJ you need to make sure that any existing security-constraint tags that apply to resources protected by VSJ are removed from the deployment descriptor.

How authorization works in VSJ

VSJ maintains an XML policy file detailing the authorization rules for an application. When the VSJ/Filter starts, it reads this policy file and contacts Active Directory to convert the user-friendly names used in the policy file, to the unique Security IDentifiers (SIDs) used by Active Directory. A secured LDAP connection is used to ensure that the mapping between these names and the SIDs can be relied upon.

When the browser contacts the application server, the VSJ-protected Filter/Servlet requests authentication using Windows Integrated Authentication. The end result is that the Filter/Servlet receives a Kerberos ticket that is proof of the user's authenticity.

In addition, Active Directory includes a Privilege Attribute Certificate (PAC) in the ticket. This includes information about the user including the Active Directory groups to which they belong. The PAC is also cryptographically authenticated, ensuring that the authorization information passed with the request can be relied upon.

The SSO Filter/Servlet then consults the policy to see if the constraints specified allow the user to access the requested resource. The constraints can be determined by the user's principal name, group membership, or—if programmatic security is being used—by any other useful information in the PAC. If the authorization is successful, then the user is allowed to access the resource. Otherwise, access is denied.

In most cases, the only time that the SSO Filter/Servlet needs to access security information from Active Directory is when it starts up. This greatly increases the scalability and reliability of your SSO solution.

The container may apply multiple filters to a request. Each filter may be assigned an access policy. VSJ currently recognizes only one access policy per request. There is currently no provision for combining multiple access policies.

For example, consider a request that is mapped using filters A and B. If filter A allows access to resource X, and filter B denies access to resource X, then access to resource X shall depend on whether filter A or filter B is being processed.

It is recommended that only one access policy is defined, and applied to one filter only.

Declarative security using VSJ

VSJ takes almost the same approach to declarative security as Java EE. Authorization rules are specified in XML using the same syntax as the standard `<security-constraint>` tag described above, and are placed in a separate policy file that the SSO Filter/Servlet reads at startup. In addition to specifying security constraints for the resources you wish to protect, this XML file also allows you to specify the role mappings that define which Active Directory groups and principals are members of a given role.

The following shows how the security constraints for declarative security in the order/inventory system described previously might be implemented:

```
<policy>
  <role name="StockManager">
  <include>
     <group name="StockMgrs"/>
  </include>
  </role>
  <security-constraint>
     <web-resource-collection>
        <web-resource-name>InventoryReports</web-resource-name>
        <description>
Security constraint for restricting access to Inventory reports.
        </description>
        <url-pattern>/inventory/reports/*</url-pattern>
        <http-method>POST</http-method>
        <http-method>GET</http-method>
     </web-resource-collection>
     <auth-constraint>
        <description>
        Access only available to StockManagers
        </description>
        <role-name>StockManager</role-name>
     </auth-constraint>
  </security-constraint>
</policy>
```

You will notice that the policy file reuses the same syntax for the security constraint. In fact, in most cases you can copy the XML `<security-constraint>` fragments from the deployment descriptor to the VSJ policy file with few or no changes.

The only difference between the `security-constraint` tag here and the one in the original example is that the `<login-config>` tag has been removed. This is because it is implied from the use of VSJ for authentication. For similar reasons the `<user-data-constraint>` tag is also not supported in the SSO policy file.

Another addition is the specification of the "StockManager" role (mapping directly to the "StockMgrs" Active Directory group. This replaces the role definition function that the application server was required to do, meaning that the same roles can be used for authorization when deploying to any application server. Roles can be specified in terms of Active Directory groups, users, or other roles in the policy file. The role definition supports both an `<includes>` and an `<excludes>` tag to allow fine-grained specification of roles. This allows a lot of flexibility in the specification of roles by deployers, without the necessity to create new Active Directory groups specifically for your application.

To make an application using VSJ enforce this policy, you configure the Filter/Servlet by adding the `idm.policy` parameter. For example:

```
<filter>
  <filter-name>authFilter</filter-name>
  <filter-class>com.wedgetail.idm.sso.AuthFilter</filter-class>
  <!-- Specify the policy file -->
  <init-param>
     <param-name>idm.access.policy</param-name>
     <param-value>policy.xml</param-value>
  </init-param>
  ...
</filter>
```

Programmatic security

Using programmatic security with VSJ is identical to using programmatic security in normal Java EE applications. This means that you can use applications that support the standard Java EE `isUserInRole` mechanism with VSJ without recompiling the code.

In addition, VSJ gives access to much more information about the user, allowing you to, for example, enumerate all the groups to which a user belongs. For more information see the API documentation for the `AuthUser` interface.

Active Directory Groups as Roles

VSJ supports a configuration parameter that allows you to specify an Active Directory group name wherever a role is specified (either in a security constraint or programmatically in `isUserInRole`). With this parameter turned on, there is no need to specify extra role definitions.

To enable support for Active Directory groups as roles, add the following to your SSO Filter/Servlet configuration:

```
<!-- Indicate the AD groups should be equivalent to roles -->
<init-param>
  <param-name>idm.access.groupsAsRoles</param-name>
  <param-value>true</param-value>
```

```
    </init-param>
```

The following caveats apply when using groups as roles:

- If a role with the same name as an Active Directory group is specified in the policy file, the role name is used rather than the group.

- If you enable roles as groups, then you may get name clashes with roles defined programmatically. To counter this you can map the clashing role to a different Active Directory group.

  For example, suppose you have code which calls `isUserInRole("Managers")` that is meant to refer to Inventory managers, and an Active Directory group called Managers that refers to all Managers within the current domain, then you should create a role-mapping to override the group as follows:

  ```
  <role name="Manager">
    <group name="InventoryManagers"/>
  </role>
  ```

- When using programmatic security, the mapping between Active Directory names and SIDs may not be known until the actual call to `isUserInRole`. This may incur a delay for the first user that trips over this code, while Active Directory is contacted to perform the mapping. In all other cases this mapping can be done at load time.

Active Directory Principals or Groups in other Realms/Domains

VSJ supports cross-realm authentication and authorization so that the policy can refer to principals and groups in other domains. This is done by using the *fully-qualified* name of the principal/group, for example:

```
<role name="Suppliers">
  <!-- Refer to suppliers in the Extranet realm -->
  <include>
    <group name="Suppliers@EXTERNAL.EXAMPLE.COM"/>
  </include>
</role>
```

Names that are not qualified are automatically assumed to be in the default domain specified by the `idm.realm` configuration parameter. This fact needs to be taken into account when changing the default domain.

Recommendations for managing authorization

The best way to manage authorization for VSJ is to define the security constraints for the resources that you wish to protect, and then define one or more "Domain Local" Active Directory groups to manage access to these resources. This allows centralized management of authorization without requiring redeployment of the application.

This is recommended, as the groups which are allowed access to an application are likely to be stable, whereas the membership of these groups is not. The Domain Local group must be in the server's domain, but may include groups and users from any other domain. See Active Directory groups for more details.

An alternative is to define the roles fully in the policy XML file and redeploy each time the role membership changes. Despite the obvious disadvantages, this does have the benefit of not requiring any support from the IT staff who manage Active Directory to manage the application authorization. It is only recommended in cases where this advantage is significant or the authorization rules are likely to be very static.

Writing access policy files

This section describes in detail how to write access policy files for VSJ. You should be familiar with the VSJ approach to authorization as described in VSJ authorization.

Overview of policy files

VSJ uses a standard text file, formatted in XML for the authorization policy. This policy file consists of two main components:

- *Role definitions* - list of roles and their members
- *Security constraints* - list of constraints to apply to resources being protected

This file is included in the Web Application aRchive (WAR) file, and referred to from the Web application deployment descriptor using the `idm.access.policy` parameter of the SSO Servlet/Filter configuration.

You can define a policy file for each Filter/Servlet, or define a global one for your application by setting the parameter inside the `<servlet-context>` tag of the deployment descriptor.

Preconditions for writing an XML policy file

1. Identify the resources that are to be protected.
2. Identify the roles that are to access these resources.
3. Disable any security constraints in your existing deployment descriptor.

The following sections discuss each of these steps in more detail.

Identify the resources to be protected

The first step to undertake in defining an access policy is to determine which resources require protection. VSJ allows you define one or more *resource collections* which are sets of URLs that match Servlets/JSPs in your application.

Depending on the complexity of your application you may decide to have one access policy that covers the entire application, or you may wish to define finer-grained access to each resource.

It helps if you can arrange the namespace of your application so that resources belonging to different groups have different URL prefixes. For example, if you have an application that has administrative and user components, you may choose to organize resources with the prefixes `/admin` and `/user`.

Identify the roles that are to access these resources

Once you have identified the resources to protect, you need to identify the roles that are to have access to these resources.

Roles are an abstraction for grouping users under one heading relating to the tasks or permissions you wish to allow. For example, you may wish to allow administrator access to an application, access by normal customers and access by premium customers. So you could define three roles:

- Admin
- Customer
- Premium Customer

These roles are then allocated to the resources they are allowed to access. When deciding to which resources a given role should be allowed access, you should adhere to the *principle of least privilege*.

Each role should be allowed to access only those resources that they need to complete their tasks, and no more.

Disable security constraints in existing deployment descriptor

VSJ will not work if there are existing constraints defined in your deployment descriptor. This is because these constraints apply before the VSJ Servlet/Filter is run, and prevent access. However, you can copy these constraints directly from the existing deployment descriptor to the policy XML file.

Creating the policy XML file

To setup the policy XML file

1. Create the policy XML file.
2. Create the main body of the policy XML file.
3. Define security constraints.
4. Define roles.
5. Set the deployment descriptor parameters.

The following sections discuss each of these steps in more detail.

Create the policy XML file

Create the file using a standard text editor. It should be saved with the extension `.xml` in the `WEB-INF` directory of the Web application.

Create the main body of the policy XML file

Place all your policy definitions inside `<policy>` tags:

```
<policy>
  <!-- Define your policy entries inside the element -->
</policy>
```

Define security constraints

For each set of resources identified, you need to define a `<security-constraint>` that maps these resources to the roles you wish to allow access.

For example:

```
<security-constraint>
  <web-resource-collection>
     <web-resource-name>Customer files</web-resource-name>
     <description>
        Resources that may be accessed by customers
     </description>
     <url-pattern>/customer/*</url-pattern>
  </web-resource-collection>
```

```
      <auth-constraint>
         <role-name>Customer</role-name>
      </auth-constraint>
   </security-constraint>
```

You can define one or more security constraints, and they can map to multiple resources and roles.

If you previously had defined security constraints in your deployment descriptor, you can copy these directly in the policy file.

Define Roles

You have two options for defining roles:

- Set the `idm.access.groupsAsRoles` option in the SSO Servlet/Filter configuration. For each role in a security constraint you should ensure there is a group defined in Active Directory of the same name.
- Define one or more `<role>` elements that map the roles you have specified to Active Directory groups/principals.

Either option can be used, however we recommend defining the roles directly in the policy file if you are using programmatic security (see VSJ authorization). This allows group membership to be resolved by querying Active Directory at load time rather than at run time.

If you are using the `idm.access.groupsAsRoles` option, we recommend defining Domain Local groups specific to your application in the same domain as the application server.

To define a role mapping to Active Directory groups using the `role` element, do the following:

```
<role name="Customer">
   <include>
      <group name="My Application Customers"/>
   </include>
</role>
```

Group names are case-sensitive.

More examples are given for the `role` element on page 71.

Set the deployment descriptor parameters

Edit the deployment descriptor (web.xml) file using either a standard text editor, or a tool supplied by your application server vendor. Add the following parameters to your SSO Servlet/Filter configuration:

```
<init-param>
  <param-name>idm.access.policy</param-name>
  <param-value>policy.xml</param-value>
</init-param>
```

Where *policy.xml* is replaced with the name of your policy file.

Policy XML descriptor elements

This section describes the following policy XML descriptor elements:

- `role`
- `include`
- `exclude`
- `user`
- `group`
- `security-constraint`
- `web-resource-collection`
- `auth-constraint`

role

The `role` element defines a security role that may be associated with a set of resources. Membership of the role can include Active Directory groups or principals, or other roles.

If the `idm.access.groupsAsRoles` option is enabled, role definitions can be used to avoid name clashes with existing Active Directory groups.

Attributes

Attribute	Required	Description
`name`	Yes	Name of the role

Elements

Attribute	Required	Description
`<include>`	Yes	Contains a list of the groups, users or roles that are members of this role
`<exclude>`	Optional	Contains a list of the groups, users or roles that are not members of this role

Group names are case-sensitive.

Examples

1. Allow the user “Alice”, and nobody else:

```
<role name="TechniciansGroupA">
  <include>
     <user name="Alice"/>
  </include>
</role>
```

2. Allow the users “Bob” and “Carol” in the domain “ACME”, and nobody else:

```
<role name="TechniciansGroupB">
  <include>
     <user name="Bob@ACME"/>
     <user name="Carol@ACME"/>
  </include>
</role>
```

3. Allow all users at the domain “ACME” and all users at the domain “APEX”.

   For this example, we use the well-known Active Directory group "Domain Users" to represent all users in a domain.

```
<role name="TechniciansAndUnqualified">
  <include>
     <group name="Domain Users@ACME"/>
     <group name="Domain Users@APEX"/>
  </include>
</role>
```

4. Allow all users at the domain “ACME”, except for “Alice”:

```
<role name="AlmostAllTechnicians">
  <include>
     <group name="Domain Users@ACME"/>
  </include>
  <exclude>
     <user name="Alice@ACME"/>
  </exclude>
</role>
```

It is a property of Active Directory that all users belong to the "Domain Users" group.

5. Allow “Dave” and “Alice” at the domain “APEX”, and “Carol” in the “ACME” domain:

```
<role name="Unqualified">
  <include>
     <user name="Alice@APEX"/>
     <user name="Dave@APEX"/>
     <user name="Carol"/>
  </include>
</role>
```

The default domain of the role above is “ACME”.

6. Allow all technicians, except those who may be unqualified:

```
<role name="QualifiedTechnicians">
  <include>
     <role name="AllTechnicians"/>
  </include>
  <exclude>
     <role name="Unqualified"/>
  </exclude>
</role>
```

include

List of groups, users or roles that are members of a given role.

At least one user, group or role element must be present.

Elements

Element	Required	Description
`<group>`	Optional	Active Directory group to be included as a member of a given role
`<role>`	Optional	Role to be included as a member of a given role
`<user>`	Optional	Active Directory user to be included as a member of a given role

exclude

List of groups, users or roles that are excluded from being members of a given role.

At least one user, group or role element must be present.

Elements

Element	Required	Description
`<group>`	Optional	Active Directory group that is excluded from being a member of a given role
`<role>`	Optional	Role to be excluded from being a member of a given role
`<user>`	Optional	Active Directory user to be excluded from being a member of a given role

user

The `user` element defines an Active Directory user. If the username is unqualified, it is assumed to be in the same domain/realm as the Web application. If you wish to specify a user in a different domain/realm, use the syntax *`user@REALM`* to specify the user.

Attribute

Attribute	Required	Description
name	Yes	Name of the user

group

The `group` element defines an Active Directory group. If the group name is unqualified, it is assumed to be in the same domain/realm as the Web application. If you wish to specify a user in a different domain/realm, use the syntax *`group@REALM`* to specify the group.

Attributes

Attribute	Required	Description
`name`	Yes	Name of the group

security-constraint

The `security-constraint` element defines access to one or more resources by one or more roles. The syntax for this element is the same as that used in the Java EE deployment descriptor, only the `user-data-constraint` and `login-config` elements are ignored.

Elements

Element	Required	Description
`<web-resource-collection>`	Yes	Lists the resources that are to be protected by the security constraint

Element	Required	Description
`<auth-constraint>`	Optional	Lists the roles that may have access to the resources protected by the security constraint

web-resource-collection

The `web-resource-collection` element defines the resources that are protected by a given `security-constraint` element.

Elements

Element	Required	Description
`<web-resource-name>`	Yes	Name of this collection
`<description>`	Optional	Description of the resources being protected
`<url-pattern>`	Optional	One or more `url-pattern` elements may be used to indicate which resources this `security-constraint` protects. Note that the URL pattern should exclude the `servlet-context`.
`<http-method>`	Optional	Indicates which HTTP methods (for example, GET or POST) are subject to this `security-constraint`. If no method is indicated, then all methods are protected.

auth-constraint

The `auth-constraint element` is used to list those roles that are authorized to access `resources` specified in a `security-constraint`.

Elements

Element	Required	Description
`<description>`	Optional	Description of the roles that are authorized
`<role-name>`	Optional	Roles that can access resources defined in the `web-resource-collection` of this `security-constraint`. If the `idm.access.groupsAsRoles` parameter is enabled, groups can be fully qualified with their realm/domain name. See the `group` element for more details.
`<http-method>`	Optional	Indicates which HTTP methods (for example, GET or POST) are subject to this `security-constraint`. If no method is indicated, then all methods are protected.

4

VSJ Federation and ADFS

- Features of VSJ Federation
- Prerequisites for VSJ Federation
- VSJ Federation installation
- VSJ Federation deployment
- Configuring VSJ Federation
- Known Issues with Federation

This chapter describes VSJ Federation, which provides Single Sign-On for Web Applications in a Microsoft ADFS (Active Directory Federation Services) environment. This is separate from the SPNEGO/NTLM functions discussed in previous chapters. VSJ Federation is included in the VSJ distribution under the `federation/` *directory.*

Features of VSJ Federation

VSJ Federation:

- Provides integration of standard web applications, independent of container and operating system, working to augment and extend Active Directory Federation Services (ADFS). ADFS is a component in Microsoft® Windows Server 2003 (R2) and Windows Server 2008 that provides Single-Sign-On (SSO) technologies to authenticate a user to multiple web applications over the life of a single online session.
- Extends Single Sign-On experience to extranet and internet applications by allowing users to use their ADFS credentials.
- Is standards-based, using Passive Requestor profile of WS-Federation; SAML (Security Assertion Markup Language) to represent claims, and HTTP 1.1 for message transport.
- Is fully extensible, allowing custom integration for security processing of a Federation token.
- Allows application developers to take advantage of the Federation environment by supporting access to the authentication information (federation token) and the SAML claims and by supporting the standard methods `isUserInRole()` and `getRemoteUser()` of `HttpServletRequest`.
- Utilizes the Application Container framework to cache authentication information across accesses, avoiding the high network latency costs involved in re-authentication.
- Produces fully configurable audit trail and logging information.

Prerequisites for VSJ Federation

VSJ Federation requires an ADFS server (Windows Server 2003 R2 or Windows Server 2008).

This version of VSJ Federation requires the following libraries.

- Apache XML Security (included)
- Apache Commons Logging (included)
- Apache Xalan and Xerces (included)
- OpenSAML (included)

VSJ Federation installation

Obtaining a license for VSJ Federation

This distribution requires a VSJ license and does not include one; you must supply one, as described in Getting started with VSJ.

Once you have a current license jar, add it to the `federation/lib/` directory alongside the other VSJ federation libraries.

XML libraries

VSJ Federation requires third party XML libraries for parsing the federation tokens (`xalan.jar, xercesImpl.jar and xml-apis.jar`).

These are located in the `federation/thirdparty/endorsed` directory, and must be installed differently, depending on the application server and JDK version being used.

For application servers with a WEB-INF classes override mechanism

Copy the endorsed XML libraries into the same location as the rest of your VSJ libraries (for example, `WEB-INF/lib`) and use the appropriate `WEB-INF` classes override mechanism for your application server. Thus:

For BEA WebLogic:

Add the following lines to your `weblogic.xml`:

```
<container-descriptor>
  <prefer-web-inf-classes>true</prefer-web-inf-classes>
</container-descriptor>
```

For Oracle Application Server:

Add the following lines to your `orion-web.xml`:

```
<orion-web-app>
  <web-app-class-loader search-local-classes-first="true" />
</orion-web-app>
```

See the `descriptors/` directory in the `vsj-fed-hello` example for sample configuration files.

For application servers using JDK 1.4

If you are running an application server which uses JDK 1.4 (for example, IBM WebSphere Application Server 5.1), the libraries supplied with the JDK must be overridden with the supplied versions using Java's endorsed libraries mechanism.

This can be achieved by either of the following methods:

- Copying the jars to your application server's endorsed directory — for example, `C:\bea\jdk141_02\jre\lib\endorsed`

  Note: this directory may not exist and may need to be created.
- Specifying the directory containing the endorsed jar files via the "`-Djava.endorsed.dirs`" system property:

  for example:

```
-Djava.endorsed.dirs=c:\VSJ-Standard-Edition-x.x\federation\thir
dparty\endorsed
```

For application servers using JDK 1.5 or higher

Copy the endorsed XML libraries into the same location as the rest of your VSJ libraries (for example, into `WEB-INF/lib`).

VSJ Federation deployment

The VSJ Federation servlet filter is deployed alongside the web application it protects. The `vsj-federation jar` should be included in the `WEB-INF/lib` directory of the web application along with any other jars it requires. The filter is then configured through the file `WEB-INF/web.xml`.

The following tasks demonstrate the installation of VSJ Federation with example commands.

They assume that the variable `%VSJF_HOME%` is set to the federation directory of your VSJ installation.

1. If the application you are protecting is contained in a `war` or `ear` file, you must explode or unzip this file into its components.
2. Copy the VSJ Federation libraries (`lib/`) and the additional libraries VSJ Federation (`thirdparty/lib` and `thirdparty/endorsed)` as noted above to the `WEB-INF/lib` directory.

3. Configure the VSJ Federation servlet filter by adding and updating the configuration items contained in the `WEB-INF/web.xml` file.

   See Configuring VSJ Federation for the list of parameters and their possible values.

4. VSJ Federation logging is configured through either the JDK (1.4 or higher) logging system or through a `log4j` properties file.

   If using `log4j` it is recommended that you place the `log4j.properties` file in the `WEB-INF` directory.

5. By default, the current release of Federation Services on the Windows 2003 Server R2 operating system uses an HTML '`<img src=""` ' request to signal the signout operation. If an image has been configured, this is returned on successful signout. An image can be configured with the `signoutImage` parameter and the actual file can be placed in the `WEB-INF` directory.
6. Recreate your web application archive file.
7. Your web application can now be deployed. Don't forget that the application must be configured in the Active Directory Federation Service Resource domain.

Configuring VSJ Federation

The following parameters are available for configuring VSJ Federation.

Parameter	Description	Default	Req'd
`fsProxy`	The URL of the resource federation server, For example: `https://resource-fs.resource.partner/adfs/ls/clientlogon.aspx`		X
`applicationUrl`	The URL to where responses (from federation servers) should be directed. It must point to a protected area of the web application. This will match the URL configured in the Active Directory Federation Service. For example: `http://master.resource.partner:8080/vsj-federation/authenticated/`		X
`signoutImage`	Specifies the location of the signout image, requested by the default Federation Server configuration on signout.	plus_green.gif	
`fsCertificate`	Specifies the location of the resource federation server base64 encoded certificate. Required, unless `acceptTokenCert` is set to `true` (not recommended).		
`acceptTokenCert`	Specifies whether to accept certificates presented in the federation token as trusted. Defaults to false.	false	
`clockSkew`	The amount of time in minutes that is allowed for differences between computer times.	5	

Known Issues with Federation

When using JDK 1.4.2 it may not be necessary to install the xalan/xerces libraries as previously described in the Installation section. However, Sun Microsystems JDK 1.4.2_05 introduced an incompatibility with the Apache security library and this requires that these libraries are installed as described.

This product uses the Apache XML Security library which contains `log4j` configuration directives that may affect users programming their own Web Service clients. In particular the `xmlsec-1.2.1.jar` contains a resource file, `org/apache/xml/security/resource/config.xml`, which sets the default (root) category not to log.

You must configure the logging for your own `log4j` category (examine the file `WEB-INF/log4j.properties` in the example for details) or change the `<root>` configuration entry contained in the `config.xml` file (contained in the `xmlsec-1.2.1.jar`).

5

Security Issues

- **Basic recommendations**
- **Deployment risks**
- **Client issues with security**
- **SPNEGO/Windows Authentication**
- **Session IDs**
- **Active Directory permissions**
- **Basic fallback**
- **Keytabs and passwords**
- **Authorization**
- **Denial of Service**
- **Auditing**
- **NTLM authentication**

This section outlines the mechanisms in VSJ used to achieve secure operation, and outlines some areas that may need special attention. It assumes familiarity with basic security concepts, Kerberos, the HTTP protocol and Java EE application configuration.

Basic recommendations

- Limit the use of basic fallback where possible (disabled by default).
- Limit the lifetime of sessions, and ensure that session IDs are "unguessable".
- Ensure that the authorization rules limit users to their least privilege.
- If using basic fallback, configure Active Directory to lock out users after some specified number of failed logins.
- Do not use basic fallback where there is a high risk of Denial of Service attacks, or provide other countermeasures to prevent them.
- Enable logging to at least the WARN level.

Deployment risks

This section discusses some of the deployment risks associated with the implementation of a VSJ-based solution. These risks are not inherent to VSJ, but may impact on VSJ's service availability or result in false positive/negative authentication.

Service unavailability

If a host (for example, the domain controller indicated by a DNS SRV query) becomes unavailable, VSJ processing may be suspended until a timeout expires. Note that VSJ maintains an internal database of unavailable hosts, and subsequent requests ignore (for a time) any hosts that are known to be unavailable. If no hosts are available for a given service, VSJ indicates an error. Any subsequent VSJ operations that must communicate with the host will timeout until such time as the host becomes available.

If a service (such as DNS) becomes unavailable, VSJ processing may be suspended until a timeout expires. After this, VSJ indicates an error. Any subsequent VSJ operations that rely on the service will timeout until such time as the service becomes available.

Time synchronization

If the internal clocks of two machines or services are sufficiently out of skew, then a Kerberos ticket which is valid on one machine may not be valid on the other machine. Thus, unsynchronized time services may lead to denial of service for otherwise-valid Kerberos tickets.

Replication interruptions

VSJ supports replicated domain controllers and global catalogs, and assumes that information is replicated across the network topology in a timely and consistent manner. Failure to replicate security information (such as group membership, SIDs, etc.) accurately may result in authentication or authorization failures.

Resource security

VSJ relies on sensitive data (such as Kerberos keytabs, passwords, and Active Directory account information). Such data must be physically and logically secure. Typically, only the Active Directory administrator should have access to Active Directory configuration, and a keytab should be readable only by the principal represented by that keytab.

Client issues with security

Cookies

Unlike some Web SSO implementations, VSJ does not use cookies to store encrypted passwords. Instead, it relies on the caching of Kerberos tickets to provide SSO functions. For Microsoft Windows clients, these credentials are stored in memory, and never written to disk, so the chance of compromise is very low. However, cookies are commonly used to store session ID state (see Session IDs).

Because authentication in VSJ is bound to the session state, these cookies present a security risk if they are leaked or sent in clear across the network. This means that communication between the browser and application server should always be done via SSL to protect session IDs.

Most application servers use transient cookies for tracking session state, with the browser only keeping them in memory, without writing them to disk. This means there is a low risk of compromise of session ID information from the client.

Alternatively, you may want to develop your applications to use URL rewriting instead of cookies for session state. This has the advantage that it works for all browsers, regardless of whether they have cookies enabled or not.

Caching of passwords for basic fallback

For non-Internet Explorer clients, or clients that do not support Kerberos, VSJ provides a mechanism for authenticating via the standard basic HTTP authentication mechanism. This sends a password in clear over HTTP, which is then used by the VSJ-protected Filter/Servlet to perform a standard Kerberos authentication. Because most clients support password managers and/or caching of passwords when using basic authentication, care should be taken to ensure that this information is not leaked or sent over the network. Again, this means ensuring that the password is not saved on a network volume, and that authentication should be done over SSL (see Basic fallback). Consult your browser documentation to ensure that it is configured appropriately.

SPNEGO/Windows Authentication

VSJ uses the SPNEGO protocol to perform Windows Integrated Authentication via HTTP. This protocol uses the Kerberos GSS-API protocol to mutually authenticate clients and servers via HTTP headers. While the GSS-API protocol itself is secure, this protocol does not ensure that the content of the HTTP request is securely bound to the message. This means that a *man-in-the-middle* attack could be used to modify the HTTP content while keeping the authentication headers intact. For this reason, unless the risk is very low, you should ensure that authentication is done over SSL.

Lifetime of authentication

Because SPNEGO requires two round trips to authenticate requests, it would be too expensive to perform an authentication for each request. Instead, applications supporting SPNEGO bind the authentication to a session or a connection. Note that authentication and re-authentication is performed transparently by Internet Explorer, and the user is not required to reenter a password.

When Windows Integrated Authentication is done to IIS using Internet Explorer, the authentication is done once per connection (HTTP 1.1 allows multiple requests to be performed over the same connection). As long as a connection is kept open, the user does not have to re-authenticate.

For VSJ, authentication is bound to a Java EE session. That is, once authenticated for a particular session, the user is not required to authenticate for the lifetime of that session, or until their service ticket expires. The session has a lifetime defined by the Java EE configuration. While this means that a user has to re-authenticate for each application (and each new session), the lifetime of these authentications may be potentially longer.

In practice, this makes little difference, as the connections should be protected by SSL, meaning that there is little opportunity for an attacker to obtain a session ID or hijack a TCP connection. However, the distinction needs to be considered when thinking about the appropriate lifetime of Java EE session IDs (see Session IDs below).

Session IDs

VSJ binds the authentication of a user to a Java EE session ID. It does this by authenticating the user via SPNEGO (or the basic fallback mechanism) and then storing the user's Kerberos ticket and other associated information in the Java EE session information. For subsequent requests that use this session ID, VSJ checks that the information is still valid and lets the request through without requesting re-authentication.

This means that an attacker who is able to sniff the session ID, or obtain it by "guessing" can subvert the security of VSJ. For this reason, SSL should always be used to protect the session, and you should ensure that the session IDs used by your application server are random and large enough to ensure that they cannot be guessed by brute force.

In addition, you should ensure that the mechanism that your application server uses to ensure *persistence* of session information does not involve sending the session ID in the clear over the network. This could be the case, for example, if JDBC to a database on another machine or a network file system is being used for persistence.

Lastly, because of the sensitivity of session ID information, VSJ ensures that it does not log session IDs, but instead logs the MD5 hash of the ID. This allows events to be correlated across sessions, without the risk of the session ID being leaked (for example, if you are logging over a network, or to a network volume). See Auditing for more details on logging.

Active Directory permissions

VSJ requires read access to a number of attributes in Active Directory. This section details which attributes and permissions are required.

Access to these elements is enabled by default in Active Directory. Setting these permissions is only required if you have modified these defaults.

The following containers are examined:

- RootDSE
- Configuration
- Partitions
- User/Group

The *RootDSE* container is examined for the following attributes:

- `rootDomainNamingContext`
- `defaultNamingContext`
- `configurationNamingContext`
- `supportedSaslMechanisms`

The *Configuration* container is examined for the following attributes:

- `objectClass=crossRefContainer`

The *Partitions* container is examined for the following attributes:

- `nETBIOSName=${DOMAIN}`
- `dnsRoot`

The top-level domain container is examined for the following attributes:

- `ntMixedDomain`

User/Group Accounts are examined for the following attributes:

- `userAccountControl`
- `groupType`
- `userPrincipalName`

- servicePrincipalName
- distinguishedName
- objectClass
- cn
- sAMAccountName
- name
- lastLogon
- badPwdCount
- objectSid
- sIDHistory
- primaryGroupID

The property sets for the attributes listed above are as follows:

Attribute	Property Set
userPrincipalName	PublicInformation
servicePrincipalName	PublicInformation
distinguishedName	PublicInformation
objectClass	PublicInformation
cn	PublicInformation
sAMAccountName	GeneralInformation
member	Membership
groupType	N/A
userAccountControl	N/A
ntMixedDomain	N/A
rootDomainNamingContext	N/A
defaultNamingContext	N/A
configurationNamingContext	N/A
supportedSaslMechanisms	N/A
name	N/A
lastLogon	User-Logon

Attribute	Property Set
`badPwdCount`	User-Logon
`objectSid`	GeneralInformation
`sIDHistory`	GeneralInformation
`primaryGroupID`	GeneralInformation
`dnsRoot`	N/A
`nETBIOSName`	N/A

For the normative reference on Active Directory property sets, see:

http://msdn2.microsoft.com/en-us/library/ms683990.aspx

Basic fallback

The basic fallback mechanism is provided to allow support for non-Kerberized browser clients. It allows a simple username and password to be entered and a Kerberos login to be performed on the server. Some issues to do with caching of passwords on clients are described in Caching of passwords for basic fallback.

The basic challenge uses the Active Directory domain in the realm part of the request, meaning that for browsers that cache passwords, the user will not have to reenter the password for the lifetime of the cached password for any application that is using that Active Directory domain.

However, unless the browser is running a password manager, the cached password disappears when the user closes the browser. This means they have to reenter their password the next time they connect to an application.

Once VSJ has used the password to obtain the Kerberos ticket on the server, the password is disposed of. This limits the risk of a server compromise compromising a large number of passwords.

It should be noted that the basic fallback mechanism provides an opportunity for an attacker to try to guess passwords for Kerberos users. VSJ does not do any checking for a number of bad logins (although it is possible to obtain the number of bad logins since the last login if a request is successful programmatically).

For this reason you should consider configuring Active Directory to lock out users after a number of bad login attempts (see Denial of Service for risks associated with this approach).

Keytabs and passwords

VSJ requires that the application be associated with a user account and provided with either a password, or a `keytab` file (see Creating keytab files) stored on the server. A keytab file contains a list of one or more keys that can be shared as part of the process of Kerberos authentication. Each key is unique to a particular authenticating user or service.

`idm.password` and `com.wedgetail.idm.sso.password` are intended to be convenient for initial setup and debugging in a test environment. Once VSJ is up and running happily with the password, we recommend using a `keytab` instead.

Similarly, while the `keytab` file location is specified by a parameter in the deployment descriptor, it must be stored at an absolute location on the server, and not in the EAR/WAR file. This is because commonly there may be multiple copies of EAR/WAR files created by developers/deployers lying around with sensitive passwords in them, or they may be deployed in the clear over unsecured links.

In all cases, care must be taken to ensure that applications that should not have access to this information are either not deployed on the same server, or have their access barred by enabling the Java security manager. If you choose to do the latter, you need to configure your Java security policy file to grant various permissions.

Authorization

Do you need authorization?

Many Java EE developers are used to deploying applications and just creating a username and password for those users who need to access the application. However, with VSJ, unless you want everyone with a desktop login in your organization to access the application, you need to define some authorization groups to control access.

Securing LDAP

VSJ uses group information in the PAC of the service ticket to authorize users. However, this group information only lists the SIDs and not the names of the groups which are specified in the authorization policy.

To resolve names, VSJ needs to contact Active Directory via LDAP. In most cases it only needs to do this once at load time.

However, if you are using some of the programmatic functions to access group information, this may also happen at run time.

The information in the PAC is securely authenticated as part of the ticket. However, the mapping between SIDs and the group names they represent must be done securely.

Otherwise, an attacker could substitute their own name for SID mappings and subvert the authorization mechanism. For this reason the connections between VSJ and the Active Directory LDAP servers need to be secured.

By default, connections to Active Directory are secured using the standard SASL/GSS-API mechanism.

Using the *Principle of Least Privilege*

The *Principle of Least Privilege* is a security maxim stating that users should only be given the least amount of privilege required, and no more.

We recommend that you apply this principle when creating authorization policies using VSJ.

Allocate a role to each task and map this to one or more groups managed in Active Directory.

It is useful to use groups which are specific to your application to prevent the risk of a user inadvertently being added to a group so they may access an entirely different application.

Denial of Service

VSJ has a number of mechanisms to prevent Denial of Service (DoS) attacks. The most important is that no state information is stored on the server until the client has authenticated.

In addition, VSJ limits the amount of communication that is required with back-end servers, and in most cases authentication and authorization can be performed locally without having to contact Active Directory.

However, there are some issues that need to be considered to increase resistance to DoS attacks. These are discussed below.

Basic fallback

Unlike Windows Integrated Authentication, the basic fallback mechanism requires communication with Active Directory to be performed by the server. Because of this, it can be exploited to mount a DoS attack by saturating the number of outgoing connections. A number of application servers can also be used to "amplify" an attack on Active Directory using this feature.

For this reason, basic fallback should be disabled in situations where there is a high risk of DoS attacks, or other measures should be undertaken (such as using a firewall that drops a large number of connections, or increasing capacity).

Session state

Once authenticated, each VSJ session will store security information associated with the connection, including the user's ticket and delegated credential.

It may be possible for an attacker who obtains a legitimate user's account to saturate the memory of an application server by making a large number of new requests to an application. However, this is a fairly low risk.

Such an event would be indicated by a large number of logins from the same account in the audit logs, and could be effectively stopped by disabling the offending account.

Auditing

VSJ provides an auditing capacity with several different levels allowing effective diagnosis and recovery for security events. Setting up logging describes how to enable this logging facility.

We recommend that the logging level be set to `WARN`, which covers security sensitive events such as bad logins. If there is sufficient capacity and a low risk of a DoS attack on your logging system, you will also find `INFO` to be useful, as this logs information about successful requests.

The audit logs contain the date, source IP, URL being accessed and, if appropriate, the MD5 hash of the session ID to allow effective correlation of events.

NTLM authentication

VSJ provides support for the NTLM authentication mechanism when SPNEGO authentication is unavailable. This is of particular use for operating system / browser combinations that do not support SPNEGO (for example, Microsoft Windows 98 and Windows NT).

What is NTLM?

NT LanManager (NTLM) is a Microsoft proprietary authentication mechanism that is integrated into all of the Windows NT family of products.

Like its predecessor, LanManager, NTLM uses a challenge-response process (sometimes referred to as NTCR) to prove client identity, without ever requiring a password or even a hashed password to be sent across the network. It does this using a three-pass process consisting of:

1. **Negotiation —** Send a list of security features supported by the client.
2. **Challenge** — Send back a list of security features agreed upon by the server, as well as a challenge that only the client would know.
3. **Authentication —** Respond to the server's challenge, and also send the username and domain information for the client.

Different versions of NTLM

Historically, the Microsoft Windows family of products has supported two variants of challenge-response authentication for network logons:

- LAN Manager (LM) challenge-response
- Windows NT challenge-response (NTLM version 1)

The LM variant allows interoperability with the installed base of Windows 95, Windows 98, and Windows ME clients and servers. NTLM was designed to provide improved security connections between Windows NT clients and servers. Windows NT also supports the NTLM session security mechanism that provides for message confidentiality (encryption) and integrity (signing).

Recent improvements in computer hardware and software algorithms have made these protocols vulnerable to widely published attacks for obtaining user passwords.

To resolve these problems, Microsoft developed an enhancement, called NTLM version 2, that significantly improved both the authentication and session security mechanisms.

NTLM 2 has been available for Windows NT 4.0 since Service Pack 4 (SP4) was released, and it is supported natively in Windows 2000.

You can add NTLM 2 support to Windows 95 and Windows 98 by installing the Directory Services Client from the Windows 2000 CD-ROM. For more instructions, see:

http://support.microsoft.com/default.aspx?scid=KB;en-us;239869

We recommend that you use NTLM v2 whenever NTLM is required for authentication.

NTLM and Internet Explorer

Internet Explorer permits four options for using the NTLM authentication mechanism:

1. *Anonymous logon* — connect to a server without attempting to provide or send logon information
2. *Automatic logon only in Intranet zone* — connect to a server by using your current session username and password, but only if the server is in your Local Intranet zone
3. *Automatic logon with current username and password* — connect to a server by using your current Windows user name and password
4. *Prompt for user name and password* — connect to a server by providing a user name and password when prompted

We do not recommend using option 3, as a malicious Web site operator can trick Internet Explorer into responding to a NTLM challenge and obtaining the password by cracking the response.

Alternatively, an attacker can send an email with a link back to the attacker's Web site, which sends an NTLM authentication challenge when the user clicks on the link.

If Internet Explorer has not been securely configured, the on-site server encrypts that challenge with the user’s password hash as the key and sends it back as the response.

The attacker may then be able to crack the user’s internal domain password.

We recommend that:

- For your Intranet zone (and perhaps Trusted sites zone, if you use that zone for business partners in an extranet scenario), you set `Logon to` *Automatic logon only in Intranet zone*.
- For your Internet and Restricted sites zone, you should use *Anonymous logon* or *Prompt for user name and passwor*d.
- If you select *Anonymous logon*, Internet Explorer won't respond to authentication requests.

  If you select *Prompt for user name and password*, Internet Explorer won't automatically respond to authentication requests with the user's domain credentials; instead, Internet Explorer displays a window asking the user for credentials.

6

Maintenance and Troubleshooting

- **Maintenance**
- **Troubleshooting**

This chapter discusses the common maintenance issues relating to a VSJ deployment and provides solutions to some common problems which may be experienced when configuring and deploying applications using VSJ.

Maintenance

This section discusses the maintenance issues relating to a VSJ deployment.

Logging

VSJ supports logging at different levels (see Setting up logging). For maintenance purposes, logging at `WARN` level is recommended, along with regular inspection of the generated log file. Regular inspection should alert the administrator to potential problems within VSJ.

New Users / Groups

Adding a new user or group to Active Directory should not impact upon the performance of VSJ, as long as existing policy settings remain unchanged. Once a new user or group has been added to Active Directory, that user may be authenticated by VSJ through the existing mechanisms.

Account settings

VSJ may cache information about user / group accounts for efficiency. For example, the groups that a user belongs to may be cached once that data has been obtained, and the authorization policy may be determined, based on this (cached) data.

If the group membership details of that user are updated dynamically, this may not be reflected in VSJ's cache, and subsequent authorization determinations may produce incorrect results.

Network topology changes

VSJ performs dynamic lookups when resolving services to hosts (such as finding the key distribution center for a realm, or domain controller for a domain). Modifying the underlying network topology should therefore not present a problem for VSJ, although it should be noted that a lag between the time the topology has been modified and the time that dynamic lookups reflect this new topology may cause some connection timeouts.

Troubleshooting

This section provides solutions to some common problems which may be experienced when configuring and deploying applications using VSJ.

General

Problem: When connecting to the servlet an Error page is displayed indicating an Internal Server error.

This error is due to either a configuration problem on the server, a misconfiguration of the client browser, or some other internal failure such as an incorrect response returned from a key distribution center. Depending on your application server, a more detailed message may be displayed in the error page, or you may need to look at the application server log files for the root cause. The following causes are noted below:

Cause 1: Could not get service ticket for <principal>@<REALM>

This error will be shown in the application server's logs as:

```
CryptoException: Integrity Check Fail
```

It is indicative of a keytab that has been created with an incorrect password. To resolve the problem, you should regenerate the keytab using the correct password.

Cause 2: Error 500: Filter [authFilter]: com.wedgetail.idm.sso.AuthFilter was found, but is missing another required class.

Java 2 Security has been enabled without a suitable policy file. Either disable Java 2 Security or contact us for help building a suitable security policy file.

Cause 3: [Servlet Error]-[Filter [authFilter]: could not be initialized]: com.dstc.security.kerberos.KerberosException: key type mismatch

This problem only occurs on Microsoft Windows 2003 when a service request is sent in an ENC type that is different from the service ticket returned. It is only a problem with Memory keytabs. One solution is to change the Active Directory user account for the service so that the *Use DES Only* option is checked. Alternatively, you could use a file keytab.

Active Directory

Problem: I created a new service principal, but then I received an "Integrity check failure" message.

Cause 1: Sometimes Active Directory doesn't set the keys properly for a newly created service principal until you log in to that account once.

Log in as that user, log out and then restart your application server software.

Browsers

Problem: VSJ returns the following HTTP error response codes:

```
401 (UNAUTHORIZED) -- request is not authorized

403 (FORBIDDEN) -- access to requested resource is
forbidden

500 (INTERNAL SERVER ERROR) -- internal server error.
```

Cause 1: Error responses from VSJ will typically return no content, and display on the client browser as an empty page.

If you want to display different content for such errors, or to take some other action based on such errors, add an `<error-page>` element to web.xml. For example:

```
<error-page>
<error-code>401</error-code>
<location>/errors/401.html</location>
</error-page>
```

Internet Explorer browsers

Problem: When using Internet Explorer, you are presented with a username and password dialog box rather than being automatically logged in.

Cause 1: This occurs when Internet Explorer does not recognize the hostname as being part of the Intranet Zone. You may not have configured Internet Explorer to use Windows Integrated Authentication.

For SPNEGO, check that your Internet Explorer version is 5.5 or greater.

Follow the steps in Troubleshooting your Internet Explorer configuration to ensure that Internet Explorer has been correctly configured to support SPNEGO.

Problem: The following error is encountered when authenticating against the server: [ERROR]: Provider protocol error: com.wedgetail.idm.spnego.server.SpnegoException: java.lang.SecurityException: Unsupported keysize or algorithm parameters

Cause 1: This problem is encountered when using a version of Internet Explorer that does not have the "High Encryption Pack" installed.

There are two work-arounds:

- Install the appropriate High Encryption Pack for your Internet Explorer browser (see Troubleshooting your Internet Explorer configuration for more information).
- Install the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files (`US_export_policy.jar` and `local_policy.jar`). You can download them from http://java.sun.com/j2se/1.4/download.html.

Problem: Internet Explorer 5.5 displays DNS error page

Cause 1: If the user account is disabled at any time, Kerberos can't renew credentials for the user.

Problem: Internet Explorer 6 displays a blank page

Cause 1: Windows Integrated Authentication is not enabled.

See Troubleshooting your Internet Explorer configuration.

Cause 2: You are going through a proxy that does not support session-based authentication.

Disable the proxy in your browser.

Non-Internet Explorer Browsers

Problem: When using a non-Internet Explorer browser, rather than presenting a username and password dialog box, the browser gives an error message indicating that the authentication mechanism is not supported.

Cause 1: The filter has not been configured to allow fallback authentication for non-Internet Explorer browsers.

You need to ensure that the filter is configured with the `idm.allowFallback` option set to `true`. Without this, non-Internet Explorer browsers are not supported.

Authentication

Problem: Servlet/JSP configured with AuthFilter fails to start.

This is commonly caused by a configuration problem with the `AuthFilter` filter. You need to check your application server's log file to determine the root cause. Possible log entries are:

Cause 1: ServletException: Need to set `idm.realm`.

The `idm.realm` parameter determines the Kerberos realm that will be use by the SSO solution to authenticate clients. It must be set in the deployment descriptor.

Cause 2: ConfigException: Only one of "`idm.keytab`" or "com.wedgetail.idm.sso.password" should be specified.

Only one of `idm.keytab` or `com.wedgetail.idm.sso.password` may be specified. For production systems, `idm.keytab` is to be preferred.

Cause 3: ConfigException: Must configure either `idm.keytab` or `com.wedgetail.idm.sso.password`.

The deployer must specify one or the other — with keytab preferred for production systems.

Cause 4: Config Exception: Must use a keytab if `idm.allowFallback` configured.

The `idm.allowFallback` parameter allows the deployer to specify whether the `AuthFilter` filter should allow users to authenticate using basic authentication and act as an authentication proxy to perform a Kerberos login to the key distribution center on the user's behalf. If fallback mode is enabled, then user-to-service mode must be used, and hence a keytab must be specified via the `idm.keytab` parameter, or the keytab password specified by the `com.wedgetail.idm.sso.password` system property.

Cause 5: ConfigException: <keytab> not found

The keytab specified by `idm.keytab` could not be loaded. The keytab must be at the path specified. Ensure that the file is present and that the correct path and file name is specified in the `idm.keytab` parameter.

Cause 6: ConfigException: No keytab entries for <principal>@<realm> in <keytab>

The specified keytab did not contain any keys for the principal that was configured using `idm.principal` and `idm.realm`. If `idm.principal` was not specified then entry will be of the form:

```
HTTP/<host.domain>@<realm>
```

Where `<host>` is the fully qualified name of the host on which the filter is deployed.

You can use the Kerberos `klist` command to check the keytab entries. Under UNIX this command is supplied as part of the Kerberos distribution and under Windows is supplied with JDK 1.4. Ensure that there is an entry for the specified principal.

Cause 7: ConfigException: Could not load keytab "<keytab>"

The keytab could not be loaded for some reason. Possibly this was due to corruption, or an incorrectly formatted file.

Cause 8: ConfigException: Invalid KDC host "<kdc>"

Cause 9: The hostname for the key distribution center specified by the parameter `idm.kdc` is invalid. Ensure that the correct hostname is supplied.

Cause 10: ProtocolException: Could not get service ticket for principal name [caused by: com.dstc.security.kerberos.CryptoException: Integrity check failure]

The realm specified in `idm.realm` is case-sensitive and is usually uppercase. If the case does not match, the DES keys for the service will be incorrect, as they are derived from the fully-qualified principal name.

Cause 11: Filter could not be loaded: com.dstc.security.util.licensing.InvalidLicense: Error verifying license: Cannot open resource jcsi.licensing.cert.pem

The licensing code can not find a license, either because it has not been installed, or Java 2 Security has been enabled, without a suitable policy file.

Problem: Why is Internet Explorer trying to do NTLM?

Cause 1: VSJ is designed to work with Windows Integrated Authentication using Kerberos. However, if your browser or Active Directory is not configured correctly this will fail, and Internet Explorer will attempt to fallback to NTLM authentication.

If you have NTLM disabled in VSJ, or if you are using VSJ (WebLogic Edition), then you may be presented with a username/ password dialog, and you will see the following message in the error log:

```
Could not authorize request:
com.wedgetail.idm.sso.NtlmNegotiatedException
```

Generally, fallback to NTLM will occur for one of the following reasons:

1. You are using a Microsoft Windows 95, 98 or ME browser which does not support Kerberos.
2. You are not logged in to the domain.
3. You are trying to connect from a browser on the same host as that on which the application server is running.
4. The Service Principal Name (SPN) is not correctly mapped to the service account used by VSJ.
5. Internet Explorer is not configured properly. In particular, the site you are connecting to is not considered part of the Intranet Zone. You also need to ensure that Windows Integrated Authentication is enabled.

One way to test whether it is either of the latter two possibilities is to install a packet sniffer. We recommend *Wireshark* (previously known as *Ethereal*) (http://www.wireshark.org). If SPN mapping for VSJ is not correct, Wireshark should provide the following output:

1. `HTTPresponse 401 Unauthorized`
2. `KRB_AP_REQ`
3. `KRB_AP_REP` (Indicating a Kerberos Error)
4. `HTTP request with the broken negotiate header`.

If you are certain the SPN has been correctly set up, send the Wireshark trace to support@quest.com and one of our experienced support engineers will assist you in identifying the problem.

If your browser has not been configured correctly as in 5. above, no Kerberos traffic will be visible in the Wireshark trace.

Keytabs

Problem: When using a keytab I get "Could not verify PAC in auth data"

Cause 1: When generating a keytab using `ktpass` the default key type is `DES-CBC-CRC`. There is a known problem using these types of keys.

You must specify `-crypto DES-CBC-MD5` when generating the keytab.

Problem: I am getting a "No keytab entry for decrypting. Data encrypted with key type 23..." message.

Cause 1: This error is caused when the keytab used contains only DES keys but the account is not set to *Use DES only*.

This can be fixed by either adding an RC4 key to the keytab, or setting the user account to *Use DES only*.

Network connections

Problem: When attempting to connect to a URL protected by the filter you receive an error message: 403 Forbidden This Connection Must be secured.

Cause 1: By default, VSJ requires that communications be performed securely (for example using HTTPS).

There are two reasons for this:

- Once a session is authenticated, the filter does not require authentication for subsequent requests using the same session ID. If it were, this would require an extra round-trip (or in some cases, two) to authenticate each request. However, this means an attacker who is able to sniff the session ID would be able to hijack an authenticated session. *Note: even if every request were authenticated, the SPNEGO protocol does not tie any of the content in the HTTP request to the authentication information, so an active attacker could still hijack session requests.*
- If the fallback mechanism is supported for non-Internet Explorer browsers, Kerberos usernames and passwords are sent unprotected over the network.

We strongly recommend using SSL to secure communications for these reasons. However, should you wish to use the SSO solution to authenticate clients over unprotected connections (for example, for testing, or where there is a very low risk of attackers hijacking sessions), you may set the `idm.allowUnsecured` option to `true`.

Credential delegation

Problem: Servlet authenticates but there are no delegated credentials (or deleg example displays message "Expected delegated credential but didn't get any")

Cause 1: The service account is probably not trusted for delegation

See Creating a VSJ account.

Cause 2: The user’s account may be configured so that delegation is not allowed.

Check the user's account properties in Active Directory.

Problem: Delegation to IIS fails, with a MIC check problem.

Cause 1: This is because IIS seems to send back a clone of the mechToken in the MIC field, which causes MIC-checking to fail.

Setting the system property `idm.spnego.noMICcheck` to `true` disables MIC checking.

Debugging

Problem: How do I get more debug information out of VSJ?

At the lowest level setting the Java system properties `jcsi.kerberos.debug` to the value `true` and `idm.spnego.debug` to the value `true` should produce logging to the standard error output stream.

VSJ Servlet Filter is configured on a per web application basis. This configuration is based upon `log4j` and defined in the Web application's `web.xml` deployment descriptor.

VSJ WebLogic Edition is configured through the WLS administration console by creating and adding the Default Audit Provider in the *Realms -> Providers -> Auditing* menu.

VSJ WebSphere Edition logging is configured through the WAS administration through the *Troubleshooting -> Logs and Trace -> servername -> Modify* menu.

VSJ JBoss Edition integrates with the `Log4J` logging framework used by JBoss. The Log4J configuration file can be found at `${jboss.home}/server/${jboss.server}/conf/log4j.xml`.

Appendix A:

Configuration Parameters

- **VSJ configuration parameters**
- **System properties**

This appendix includes a detailed list of configuration parameters for VSJ operations, including those involved directly in VSJ (naming convention `idm.*`*) and those related to system properties relevant to VSJ.*

VSJ configuration parameters

`idm.principalAtRealm`

This parameter should be set to the fully qualified name (including the Active Directory domain) of the VSJ service principal you have set up (see Setting up the VSJ account) — for example, `vsj_appservhost1@EXAMPLE.COM`.

Not needed if the `idm.keytab` points to a keytab exclusively containing entries for the VSJ service principal.

`idm.realm`

The Active Directory domain to be used for authentication. Its use is now deprecated in favour of the preferred `idm.principalAtRealm` parameter (see above). Not needed if the `idm.keytab` points to a keytab exclusively containing entries for the VSJ service principal.

`idm.princ`

The Kerberos service principal to use. Use of this parameter is also deprecated in favor of the preferred `idm.principalAtRealm` parameter above. Not needed if the `idm.keytab` points to a keytab exclusively containing entries for the VSJ service principal.

`idm.keytab`

The file containing the keytab that Kerberos will use for user-to-service authentication.

If unspecified, VSJ defaults to using an in-memory keytab with a password specified in the `com.wedgetail.idm.sso.password` custom property or the `idm.password` parameter.

`idm.password`

The password of the VSJ account specified by `idm.principalAtRealm`. VSJ creates an in-memory keytab using this password. **NOTE:** This parameter is required if the `idm.keytab` parameter or `com.wedgetail.idm.sso.password` property is not set.

`idm.allowS4U`

Boolean logic value (`true/false`) to toggle the use of S4U2Proxy (Constrained delegation) and S4U2Self (Protocol transition) in VSJ operations.

Default is `false`.

`idm.allowNTLM`

Boolean logic value (`true/false`) to specify whether to allow fallback to NTLM authentication. This is required if you want to use the SSO solution with pre-Windows 2000/XP Internet Explorer browsers which do not support SPNEGO.

The default is `false`.

`idm.allowFallback`

Boolean logic value (`true/false`) to specify whether to allow fallback to basic authentication. This is required if you want to use the SSO solution with non-Internet Explorer browsers.

The default is `false`.

`idm.allowUnsecured`

Boolean logic value (`true/false`) specifying whether to allow authentication over an unsecured channel.

It is **strongly recommended** that you do **not** be set this to `true` unless there is a **very** low risk of an attacker accessing the communication channel between the client and server.

The default, if not set, is `false`.

`idm.userHandledExcept`

Boolean logic value (`true/false`). Setting this parameter to `true` propagates exceptions from VSJ code to the Web server so that you can write your own error pages based upon the VSJ error that occurs.

`idm.kdc`

The Active Directory key distribution center (KDC) against which secondary credentials should be validated This can be used for basic fallback, or credential delegation.

By default, the KDC is discovered automatically and this parameter should only be used if automatic discovery fails, or if a different KDC to the one discovered automatically should be used.

`idm.fallbackCrossRealm`

Boolean logic value (`true/false`). If this parameter is set, VSJ attempts to guess the client's realm from the domain part of the hostname returned by ServletRequest.getRemoteAddr(), and the user is not required to append the realm to their username. By default this is `false`.

`idm.supportMultipleSPN`

Boolean logic value (`true/false`). Specifies whether to support multiple service principal names. If this option is set to `true`, the server uses the service name in the ticket to determine which key to use for decryption. The default is `false`. (See Creating a VSJ account.)

`idm.ad.site`

The name of the Active Directory site in which this server should be placed.

`idm.ad.login`

The username used to access Active Directory user information via simple authentication.

If specified, it should be the fully-qualified user created for the service principal as described in Creating a VSJ account (for example, user@EXAMPLE.COM).

`idm.ad.security`

This parameter specifies how connections to LDAP servers are secured. Possible options are `sasl` and `simple`.

`idm.access.policy`

Specifies the file from which access policies for authorization will be read. If unspecified, access must be handled programmatically.

idm.access.groupsAsRoles

Boolean logic value (true/false). If a given role is not associated with any users, it treats the role as an Active Directory group. By default this is false.

idm.ad.userPrincipalAttribute

This parameter can be set to the name of an Active Directory attribute.

If set, the value of this attribute will be made available to web applications rather than the user name that was authenticated. This could be used to allow a user to login normally with username and password but then be known to the web application by perhaps an employee id, which is stored and managed in Active Directory.

This property takes precedence over idm.ad.qualifyUserPrincipal but yields precedence to idm.ad.userPrincipalFormatterClass.

idm.ad.qualifyUserPrincipal

Boolean logic value (true/false). Set this to fully qualify the authenticated user name returned by VSJ that is, append the Active Directory domain name. The idm.ad.userPrincipal attribute and idm.ad.userPrincipalFormatterClass properties take precedence over this one.

This property should be set to true if you are using the VSJ User Registry in VSJ WebSphere Edition in an environment with multiple Active Directory domains.

The default, if not set, is false.

idm.ad.userPrincipalFormatterClass

The class name of a com.wedgetail.idm.sso.UserPrincipalFormatter implementation to use for formatting principal names returned by VSJ. This property takes precedence over both idm.ad.userPrincipalAttribute and idm.ad.qualifyUserPrincipal.

idm.ntlm.signing.username

The NTLM logon name of a user account that VSJ can use for NTLM signing.

`idm.ntlm.signing.domain`

The NTLM domain (not the AD domain) to which the above user account belongs.

`idm.ntlm.signing.password`

The password for the above account. See Keytabs and passwords in Security Issues.

`idm.ntlm.signing.always`

Boolean logic value (`true/false`): `true` means that VSJ should always use these signing parameters.

`false` means that VSJ should only use these signing parameters if it got an exception while trying to authenticate a user without using these signing parameters.

The default value (`true`) is recommended.

`idm.trimUnsolicitedBasic`

Boolean logic value (`true/false`) used for tuning. Optimize unnecessary HTTP Basic reauthentication. The default is `false`.

If this is `true` and a client sends us an "*Authorization: Basic*" header when we don't need it (because we have already authenticated the client) then we ignore the header.

If this is `false`, we process the header (and reauthenticate the client) even though we don't really need to.

This parameter is never harmful but only helps performance in the unusual case where a client sends Basic authentication much more often than necessary (for example, on every request).

`idm.trimUnsolicitedNTLM`

Boolean logic value (`true/false`) used for tuning. Optimize unnecessary HTTP NTLM reauthentication.

The default is `false`.

If this is `true` and a client sends us an "*Authorization: NTLM*" header (or NTLM dressed up as Negotiate) when we don't need it (because we have already authenticated the client) then we process it just enough to humour the client — because otherwise Internet Explorer won't send us the body of a POST or PUT request.

But we don't bother to perform the last, rather expensive, step of NTLM authentication: contacting a domain controller to validate the NTLM password hashes.

If this is `false` we process the header completely and reauthenticate the client even though we don't really need to.

It is generally a good idea to enable this option if `idm.allowNTLM` is enabled and a significant percentage of your HTTP NTLM requests are POST or PUT (because Internet Explorer reauthenticates on every HTTP request with a content-body, such as POST or PUT).

There are no known disadvantages to enabling this option.

`idm.trimUnsolicitedSPNEGO`

Boolean logic value (`true/false`) used for tuning. Optimize unnecessary HTTP SPNEGO reauthentication.

The default is `false`.

If this is `true` and a client sends us an SPNEGO token in an "*Authorization: Negotiate*" header when we don't need it (because we have already authenticated the client), we ignore the header.

If this is `false` we process the header (and reauthenticate the client) even though we don't really need to.

This option is only necessary if a significant percentage of your HTTP SPNEGO requests are POST or PUT — because Internet Explorer reauthenticates on every HTTP request with a content-body, such as POST or PUT —and the CPU overhead of the unnecessary SPNEGO/Kerberos reauthentication operations is becoming prohibitive.

Note: if this option is enabled, VSJ does not send an SPNEGO response token ("*NegTokenTarg*"). This is fine for Internet Explorer, but may or may not be fine for other clients.

`idm.ntlm.userCache.maxSize`

The maximum number of entries in the NTLM user cache.

`idm.ntlm.userCache.maxAge`

The maximum lifetime (in milliseconds) of entries in the NTLM user cache.

This value only needs to be large enough so that repeated authentications for the same NTLM user over a short period (for example, by HTTP clients that don't support JSESSIONID cookies) will use the cache instead of triggering repeated expensive lookups for the same information.

The default is ten minutes (600000 millisecs).

`idm.disableTicketSanityCheck`

Boolean logic value (`true/false`). Normally when VSJ starts it up it contacts Active Directory to check that `idm.principalAtRealm (`or `idm.principal` and `idm.realm)` and `idm.keytab` or `idm.password` are valid. If they aren't valid, VSJ reports the problem and gives up.

If you need to disable this check, set this boolean parameter to `true`.

Default is `false`.

`idm.propertyFileURL`

Parameter for specifying the URL of the property file.

`idm.allowServerTGT`

Boolean logic value (`true/false`). Allow code in the web application to obtain and use VSJ's own TGT. The web application may need this if, for instance, it wants to perform LDAP queries to Active Directory which go beyond the LDAP functionality that VSJ provides.

`idm.logger.name`

Specifies the unique name of the logger. This must match the name in any related Log4J properties file.

`idm.logger.props`

Specifies the file from which Log4J properties should be loaded for logging. If no properties file is specified, errors are logged to the servlet-context log by default.

System properties

These are normally to be set either:

- in start-up scripts;
- by specification in a command line (for example, using the -D flag); or
- by Java code specifically designed to set system properties.

`com.wedgetail.idm.sso.password`

The password of the Kerberos service principal. VSJ creates an in-memory keytab using this password.

Note: This property is required if `idm.keytab` or `idm.password` parameters are not set.

See Keytabs and passwords.

`jcsi.kerberos.nameservers`

A colon-separated list of one or more DNS servers that VSJ should use to look up DNS SRV records for Active Directory domain controllers. Specify each DNS server as either a hostname or as an IPv4 address.

Normally VSJ automatically discovers DNS servers by querying the JVM and/or the operating system. However, in some circumstances (such as when VSJ is running on z/OS), VSJ's auto-discovery logic comes up empty-handed, so the list of DNS servers must be specified explicitly.

Appendix B:

Using the JKTools

- Tool details
- jkinit
- jklist
- jktutil

VSJ includes several tools which enable you to create, manipulate and display Kerberos credential caches and keytab files. This section describes how to use `jkinit` *to authenticate and request a network credential,* `jklist` *to display a credential cache or keytab contents, and* `jktutil` *to create, manipulate and display keytabs*

Tool details

The tools are Java-based and will therefore run on UNIX, Linux, and z/OS as well as Microsoft Windows platforms.

The tools are compatible with files created by MIT Kerberos, Heimdal and VAS.

Each tool has help embedded so that invoking the tool with the -help parameter displays summary information about the parameters the tool supports.

Scripts for running the tools on different operating systems — Windows (`*.bat` files) and UNIX or Linux (`*.sh` files) — are provided in the `bin/` directory of your VSJ distribution.

jkinit

The **jkinit** tool is used to request and store a credential from a Microsoft Active Directory. This is located on the Domain Controller responsible for the requested domain (or realm). After a successful authentication a credential is returned which is then stored in a credential cache. The credential can then be used in later operations.

Usage:

```
jkinit {option} [principal [password]]
```

The principal for whom the ticket is issued may be specified either on the command line (in the form "name@realm"), or it may be derived from the default principal of an existing credential cache.

Authentication information must be present with the principal. This information may be in the form of a password, or as a key contained in a keytab.

The password may be specified explicitly on the command line.

A keytab may be specified using the '-k' option (see below).

If a password is not specified on the command line, and a keytab is not specified using the '-k' option, the user is prompted to enter a password.

Once the credential for the principal has been obtained, it is written to a credential cache. The credential cache file may be specified explicitly via the '-c' option (see below), or jkinit may locate the default credential cache.

If no principal has been specified on the command line, the credential cache must already exist, and must contain a default principal. If a principal has been specified on the command line, the credential cache (however specified) is created if necessary. In either case, the credential obtained from the key distribution center is added to the credential cache.

Options:

```
-c <cache_file>
```

Specifies the name of the credentials cache file. If the cache does not exist, it is created. If this option is not specified, the default credential cache is loaded, as follows:

For UNIX-based systems, the default credential cache locations are:

1. The location specified by the `$KRB5CCNAME` environment variable, if present; or
2. The location `${user.home}/krb5cc_${user.name}`, where

   `${user.home}`

   represents the user's UNIX home directory, and

   `${user.name}`

   represents the user's login name on the UNIX system; or
3. The location `/tmp/krb5cc_${uid}`, where

   `${uid}` represents the user's UNIX ID.

For Windows-based systems, the default credential cache locations are:

1. The Local Security Authority; or
2. The location `${user.home}/krb5cc_${user.name}`, where

   `${user.home}`

   represents the user's Windows home directory, and

   ${user.name}

   represents the user's login name on Windows.

`-f`

Specifies a *forwardable* ticket. Otherwise, default is not forwardable.

`-p`

Specifies a *proxiable* ticket. Default is not proxiable.

`-l <lifetime>`

Specifies lifetime (in hours) of the ticket. Otherwise, the ticket has the default lifetime as specified by the key distribution center.

`-R`

Specifies a renewable ticket. Default is not renewable.

`-A`

Specifies an addressless ticket. Otherwise, the ticket is valid for all local addresses.

`-k`

Specifies use of a keytab rather than a password.

If a keytab location is not specified via the `-t` option below, a default keytab is loaded, as follows:

For Windows-based systems, the default keytab location is

```
${user.home}\krb5.keytab
```

For UNIX-based systems, the default keytab locations are:

1. `${user.home}/krb5.keytab`
2. `/etc/krb5.keytab`

where

```
${user.home}
```

is the user's home directory.

`-t <keytab_file>`

Specifies the location of the keytab file, as opposed to the default keytab. Must be used with the `-k` option.

`-S <service_name>`

Specifies an alternative service name. Otherwise, the default service name is `krbtgt/${REALM}`, where `${REALM}` is the realm of the principal.

`-K <host name>`

Specifies the host name of the key distribution center (KDC). If not specified, the KDC is determined dynamically from the realm of the principal.

`-V, -verbose`

Specifies verbose output. This enables display of the operations performed, name of files used, and the data in the credential returned.

`-debug`

Specifies debug output. Displays the verbose output as outlined above, and further information that may be useful in debugging and locating errors.

`-help`

Shows a list of options, and exits the application.

jkinit examples

Get a TGT for the principal 'fred@EXAMPLE.COM', with the password 'test', and put that TGT into the default credential cache. Use verbose output so that the credential cache file is known:

```
$ jkinit -V fred@EXAMPLE.COM test

## Requesting ticket for service krbtgt/EXAMPLE.COM by principal
fred@EXAMPLE.COM

## Storing ticket in cache FILE:/tmp/krb5cc_1062
```

Get a TGT for the principal in the default credential cache:

```
$ jkinit -verbose

## Using credential cache FILE:/tmp/krb5cc_1062

## Requesting ticket for service krbtgt/EXAMPLE.COM by principal
fred@EXAMPLE.COM

Password for fred@EXAMPLE.COM: ****

## Storing ticket in cache FILE:/tmp/krb5cc_1062
```

Get a TGT for the principal 'fred@EXAMPLE.COM', and put that TGT into the credential cache 'fred.ccache':

```
$ jkinit -c fred.ccache -verbose fred@EXAMPLE.COM


## Using credential cache FILE:/home/fred/freddo.ccache

## Requesting ticket for service krbtgt/EXAMPLE.COM by principal
fred@EXAMPLE.COM

Password for fred@EXAMPLE.COM: ****
```

```
## Storing ticket in cache FILE:/home/fred/fred.ccache
```

Get a TGT for the principal 'barney@EXAMPLE.COM", using the keytab `/home/barney/barney.kt`***:***

```
$ jkinit -k -t /home/barney/barney.kt -verbose barney@EXAMPLE.COM

## Using credential cache FILE:/tmp/krb5cc_2000

## Requesting ticket for service krbtgt/EXAMPLE.COM by principal
barney@EXAMPLE.COM
```

jklist

The `jklist` tool is used to display the contents of credential caches and keytabs including the key encryption types, the ticket flags, principal name, or session keys held by the current user.

The following information about the credentials cache is listed:

- the name of the credentials cache
- the identity of the principal for whom the tickets in the cache are for
- information about the tickets held:
    - the principal name of the ticket;
    - the issue and expiry time of the ticket

Additional cache information may be displayed using the `-a`, `-n`, `-e`, and `-f` options.

The following information about the keytab is listed:

for each key in the keytab:

- the key version number
- the principal

Additional keytab information may be displayed using the `-K`, `-t`, and `-e` options.

Usage

```
jklist [[-c][-e][-f][-a [-n]] [-k [-t][-K]]

        [-help][-debug][-verbose] [<filename>]
```

The `<filename>` represents the name of a keytab if the `-k` option is specified, and the name of a credential cache if the `-c` option is specified.

If neither the `-c` nor the `-k` options are specified, the `-c` option is assumed as the default.

If `<filename>` is not present, the location of the credential cache or keytab is determined dynamically.

Options

The following options are supported:

`-e`

Displays the encryption type of the session key for each credential in the credential cache, or for each key in the keytab file.

`-c`

Displays the credentials of a cache. This is the default if neither -c nor -k options are specified. If no filename is specified, the cache is located as follows:

For Windows-based systems, the default keytab location is:

`${user.home}\krb5.keytab`

For UNIX-based systems, the default keytab locations are:

1. `${user.home}/krb5.keytab`
2. `/etc/krb5.keytab`

where

`${user.home}`

is the user's home directory.

`-a`

Display the addresses listed in the credential.

-n

Shows numeric IP addresses instead of reverse-resolving addresses. Only valid with `-a` option.

-f

Display the flags in the credential, with the following abbreviations:

- "F" - forwardable
- "f" - forwarded
- "P" - proxiable
- "p" - proxy
- "D" - post-dateable
- "d" - post-dated

- "R" - renewable
- "I" - initial
- "i" - invalid

`-k`

Display the keys of a keytab.

`-t`

Display timestamp for each entry in the keytab

`-K`

Display encryption key value for each entry in the keytab.

`-help`

Print help about jklist usage and exit

`-verbose`

Show verbose output.

`-debug`

Show debug output. This shall include verbose output.

jklist examples

Display the default credentials cache:

```
$ jklist


Ticket cache: FILE:/tmp/krb5cc_1062

Default principal: fred@EXAMPLE.COM


Valid starting       Expires              Service Principal

08/31/2004 12:57:35  08/31/2004 13:57:35
krbtgt/EXAMPLE.COM@EXAMPLE.COM
```

Display the credential cache `fred.ccache`***:***

```
$ jklist fred.ccache


Ticket cache: FILE:/home/fred/fred.ccache

Default principal: fred@EXAMPLE.COM
```

```
Valid starting       Expires              Service Principal

08/31/2004 14:14:02  08/31/2004 15:14:02
krbtgt/EXAMPLE.COM@EXAMPLE.COM
```

Display the credential cache `fred.ccache`***, including encryption types, ticket flags, and unresolved addresses:***

```
$ jklist -f -a -n


Ticket cache: FILE:/tmp/krb5cc_1062

Default principal: fred@EXAMPLE.COM


Valid starting       Expires              Service Principal

08/31/2004 14:14:02  08/31/2004 15:14:02
krbtgt/EXAMPLE.COM@EXAMPLE.COM

    Flags: IA

    Addresses: puffin.example.com
```

Display the default keytab:

```
$ jklist -k


Keytab name: FILE:/home/fred/krb5.keytab

KVNO Principal

---- --------------------

  255 fred@EXAMPLE.COM
```

Display the keytab `fred.kt`***:***

```
$ jklist -k fred.kt

Keytab name: FILE:freddo.kt

KVNO Principal

---- --------------------

  255 fred@EXAMPLE.COM
```

Display the default keytab, including encryption types, timestamps and key values:

```
$ jklist -k -t -K -e


Keytab name: FILE:/home/fred/krb5.keytab

KVNO Timestamp           Principal            EncType     Key

---- ------------------- -------------------- ----------- ---

  255 08/31/2004 14:17:06 fred@EXAMPLE.COM   des-cbc-crc
75B65ED67C0843B9
```

jktutil

The `jktutil` tool allows the user to create `keytab` entries specifying the principal name, encryption type and key version number. The entries can then be saved or appended to a `keytab` file. `jktutil` can also read and write keytab files, which enables merging of keytabs and their entries, and can list the current set of keys.

Usage

```
jktutil [-help][-verbose][-debug]
```

Options

`-verbose`

Show verbose output.

`-debug`

Show debug output (includes '-verbose').

`-help`

Show help screen and exit.

Operation

Once the jktutil application has started, the user is presented with a prompt, at which commands are entered:

```
jktutil (type '?' for help):
```

The following commands are supported by `jktutil` (note that some commands may have more than one name):

`list <filename>`

List the available entries. May use the letter `l` as an alias for `list`. Initially, there are zero entries. Entries are added by creating new entries (via the `add_entry` command), or by reading a `keytab` (via the `read_kt` command).

`clear_list`

Clear the list. May use `clear` as an alias for `clear_list`.

`read_kt <filename>`

Read keys from the specified keytab file and add them to the list. May use `rkt` as an alias for `read_kt`.

```
write_kt [-a|-o] <filename>
```

Write the entries in the list to the specified keytab file. May use `wkt` as an alias for `write_kt`.

The options for the `write_kt` command are as follows:

- -a

  Append entries to the end of the `keytab` file, if the `keytab` file already exists. This is the default option.

- -o

  Overwrite the keytab file with the entries in the list. In either case, the list remains unchanged.

```
delete_entry <slot>
```

Delete the entry at the specified slot from the list.

May use `delent` as an alias for `delete_entry`.

Entries are numbered from 1.

```
add_entry  (-key | -password) -p <principal> -k <kvno> -e <enctype>
```

Add an entry to the list.

May use `addent` as an alias for `add_entry`.

The options for the `add_entry` command are:

- `-key`

  Specify a key value via command line

- `-password`

  Specify a password via command line

- `<principal>`

  The principal, in the form 'name@realm'

- `<kvno>`

  The key version number

- `<enctype>`

  The encryption type. Supported values are:

  - des-cbc-crc
  - des3-hmac-sha1
  - des-cbc-md4
  - des-cbc-md5
  - rc4-hmac
  - aes256-sha1

- aes128-sha1

The new entry is added to the end of the current list.

`list_requests`

List the available commands. May use the ’lr’ or '`?`' as an alias for `list_requests`.

`help_command <command_name>`

Get help for the specified command name. May use '`hc`' as an alias for `help_command`.

`quit`

Quit the application. May use `exit` or `q` as an alias for `quit`.

jktutil example

The following example shows a `jktutil` session, in which a `keytab` is read, its contents listed, and a new key is added:

```
$ jktutil

jktutil (type '?' for help): l

Slot KVNO Principal

---- ---- --------------------

jktutil (type '?' for help): read_kt barney.kt

jktutil (type '?' for help): l

Slot KVNO Principal

---- ---- --------------------

1 255 barney@EXAMPLE.COM

jktutil (type '?' for help): addent -password -p fred@EXAMPLE.COM
-k 255 -e rc4-hmac

Password for fred@EXAMPLE.COM:

jktutil (type '?' for help): l

Slot KVNO Principal

---- ---- --------------------

1 255 barney@EXAMPLE.COM

2 255 fred@EXAMPLE.COM
```

```
jktutil (type '?' for help): write_kt -o barney.kt
jktutil (type '?' for help): clear
jktutil (type '?' for help): l
Slot KVNO Principal
---- ---- ---------------------
jktutil (type '?' for help): read_kt barney.kt
jktutil (type '?' for help): l
Slot KVNO Principal
---- ---- ---------------------
1 255 barney@EXAMPLE.COM
2 255 fred@EXAMPLE.COM
```

INDEX

A

acceptTokenCert 82
access
  for separately-authenticated clients 16
access policy files 20
Active Directory
  accounts for VAS 31
  configuration 27
  encryption types 29
  groups 7–9
    logon process 8
    scopes 7–8
    types 7
    VSJ 9
  LDAP integration 6
  permissions 89
  Public Key Infrastructure (PKI) support 7
  sites 9
  smartcard support 7
  VSJ 12
  Windows native credential cache 7
Active Directory Federation Services 78
adding new users or groups 100
ADFS 78
  and VSJ Federation 77
AES
  keytab creation example 42
AES encryption support 29
alias hostnames
  and SPN mappings 31
anonymous logon 97
Apache Commons Logging library 78
Apache XML security library 78
applicationUrl 82
auditing
  and logging 95
authentication 19, 88, 89, 94, 96
  and cookies 87
  basic 19, 44
  NTLM 19, 43, 95
  PKI 7
  smartcard 7
  SPNEGO 19
  Windows-integrated 43, 44, 45, 46, 88, 95
authentication risk
  passwords on network 88
automatic logon
  only in Intranet 97
  with username/password 97
automatic logon with username/password
  not recommended 97

B

basic fallback 19, 44, 86, 88, 89, 92, 95

C

cached data 100
cacheing
  and updated membership 100
challenge-response
  and LM 96
changes in network topology 100
client machine 42
  browser 43
clock skew and drift allowable 26
clockSkew 82
com.wedgetail.idm.sso.password property 117
computer objects
  and SNP mapping 11
  creation using vastool 32
configuration
  Active Directory 27
configuration parameters
  for VSJ 110
configuring VSJ Federation 82
constrained delegation 37
  and S4U2Proxy 15
  and service tickets 15
  cf unconstrained 15
  concept 13
cookies
  and SSL 87
  risk of 87
  URL rewriting as alternative 88
creating a user account
  for VSJ 27
credential cache, native
  Microsoft Windows 7

D

delegation

and "trust" 14
and protocol transition 37
constrained 15, 37
disallowing in Windows 2000 35
disallowing in Windows 2000, 2008 37
in Kerberos and VSJ 13
limits and types 14
options for Windows 2000 35
options for Windows 2003, 2008 37
setting up VSJ for 35
to limited, specified services 37
to services on another computer 38
unconstrained 15
unconstrained in Windows 2000 35
unconstrained in Windows 2000, 2008 37
using service ticket 37
using TGT 35, 37
with S4U2Proxy 37
with S4U2Self 37
Denial of Service attack 86
and VSJ 94
deployment of VSJ 47
deployment risks 86, 87
replication interruptions 87
resource security 87
service unavailability 86
time synchronization 86
DES
keytab creation example 41
DES encryption support 29
digest authentication
and protocol transition 16
and S4U2Self 16
distinguished name 6
DNS
requirement for VSJ 25
SRV resource records 25
DNS aliases
and SPN mappings 31
Domain Name Service
required for VSJ 25
DoS attacks
prevention of in VSJ 94
duplicated mappings 31
dynamic network lookups 100

E

enabling delegation 35
encryption
supported types 29
encryption types
in Active Directory 29
endorsed libraries
for JDK 1.4 80
for JDK 1.5 80
external clients
and protocol transition 16
and S4U2Self 16

F

fallback
basic 88
Federation
use of S4U2Self 16
Federation features 78
firewall
may require protocol transition 16
may require S4U2Self 16
ports to open on 41
fsCertificate 82
fsProxy 82

G

group creation
using vastool 32
groups, Active Directory 7–9

H

header size limits 48
HTTP digest authentication
and protocol transition 16
and S4U2Self 16
http header size issue 48
HTTP Negotiation
and SPNEGO 12
HTTP service principal creation
with vastool 33

I

idm.access.groupsAsRoles 113
idm.access.policy 112
idm.ad.login 112
idm.ad.qualifyuserPrincipal 113
idm.ad.security 112
idm.ad.site 112
idm.ad.userPrincipalAttribute 113
idm.ad.userPrincipalFormatterClass 113
idm.allowFallback 111
idm.allowNTLM 111
idm.allowS4U 111

idm.allowServerTGT 116
idm.allowUnsecured 111
idm.disableTicketSanityCheck 116
idm.fallbackCrossRealm 112
idm.kdc 111
idm.keytab 110
idm.logger.name 116
idm.logger.props 116
idm.ntlm.signing.always 114
idm.ntlm.signing.domain 114
idm.ntlm.signing.password 114
idm.ntlm.signing.username 113
idm.ntlm.userCache.maxAge 116
idm.ntlm.userCache.maxSize 115
idm.password 110
idm.princ (deprecated) 110
idm.principalAtRealm 110
idm.propertyFileURL 116
idm.realm (deprecated) 110
idm.supportMultipleSPN 112
idm.trimUnsolicitedBasic 114
idm.trimUnsolicitedNTLM 114
idm.trimUnsolicitedSPNEGO 115
idm.userHandledExcept 111
IE
  and NTLM 97
INFO level
  and DoS attacks 95
installation
  client machine 42
  Java application server 41
installation process example
  VSJ Federation 80
Internet Explorer
  and SPNEGO 12

J

Java JSPs 2
Java Servlets 2
jcsi.kerberos.nameservers
    property 117
jkinit tool 121
  examples of use 124
jklist tool 126
  examples of use 128
jktools 120
jktutil tool 131
  example of use 133
JSPs 2

K

Kerberos 4, 5
  and constrained delegation 13, 15
  and protocol transition 17
  and unconstrained delegation 15
  authentication 4, 5, 88
  delegation 14
  LDAP integration 6
  message exchange 4
  MIT 4, 6, 7
  Privilege Attribute Certificates
    (PAC) 6
  S4U2Self extension 17
keytab
  creation examples 41
  creation using vastool 32
  files in VAS 32
  protection of 93
  risks with 93
  role and use 93
  vs passwords 41
known issues with VSJ Federation 83

L

LDAP 6, 7, 8, 9, 93
  Active Directory integration 6
  Kerberos integration 6
libraries
  XML required for VSJ Federation 79
library requirements
  for VSJ Federation 78
license
  file format 48
  installation 48
  obtaining for VSJ 48
license deployment
  for VSJ Federation 79
LM challenge-response 96
log4j issues
  VSJ Federation 83
logging level
  recommended as WARN 95
logging maintenance 100

M

maintenance 100
maintenance issue
  cache for users, groups 100
maxHttpHeaderSize 48
maximum header size
  changing 48
memory problem
  http headers 48
Microsoft Windows

credential cache, native 7
integrated authentication 43, 44, 45, 46, 88, 95
MIT Kerberos 4, 6, 7

N

name
distinguished in LDAP 6
negotiation, challenge, authentication 96
network updates and changes 100
new users and groups 100
NTLM 96
and IE 97
authentication 19, 43, 95
described 96
different versions 96
IE options for 97
recommendations for security 97

O

OpenSAML library 78
Oracle configuration
VSJ Federation 79
overriding JDK 1.4 in WebSphere 80

P

PAC 6, 8, 93
used in NTLM 20
PAC, see *Privilege Attribute Certificate (PAC)*
parameter configuration
for VSJ 110
for VSJ Federation 82
parameters
idm.access.groupsAsRoles 113
idm.access.policy 112
idm.ad.login 112
idm.ad.qualifyuserPrincipal 113
idm.ad.security 112
idm.ad.site 112
idm.ad.userPrincipalAttribute 113
idm.ad.userPrincipalFormatterClass 113
idm.allowFallback 111
idm.allowNTLM 111
idm.allowS4U 111
idm.allowServerTGT 116
idm.allowUnsecured 111
idm.disableTicketSanityCheck 116
idm.fallbackCrossRealm 112
idm.kdc 111
idm.keytab 110
idm.ntlm.signing.always 114
idm.ntlm.signing.domain 114
idm.ntlm.signing.password 114
idm.ntlm.signing.username 113
idm.ntlm.userCache.maxAge 116
idm.ntlm.userCache.maxSize 115
idm.password 110
idm.princ 110
idm.principalAtRealm 110
idm.propertyFileURL 116
idm.realm 110
idm.supportMultipleSPN 112
idm.trimUnsolicitedBasic 114
idm.trimUnsolicitedNTLM 114
idm.trimUnsolicitedSPNEGO 115
idm.userHandledExcept 111
password risk
on network 88
permissions, Active Directory 89
PKI, see *Public Key Infrastructure (PKI)*
ports to open on firewall 41
principal names
in VSJ 11
principle of least privilege 94
privilege
principle of least 94
Privilege Attribute Certificate (PAC) 6, 8, 9, 93
prompt for username/password 97
property
com.wedgetail.idm.sso.password 117
jcsi.kerberos.nameservers 117
protocol transition
and external clients 16
and Federation 16
and HTTP digest authentication 16
and NTLM 16
and S4U2Self 16
configuring 37
Public Key Infrastructure (PKI)
Active Directory, support in 7

R

RC4
keytab creation example 41
RC4 encryption support 29

S

S4U2Proxy

and constrained delegation 15
delegation configuration 37
S4U2Self
and external clients 16
and Federation 16
and HTTP digest authentication 16
and NTLM 16
and protocol transition 16
configuration 37
SASL 94
security rules
principle of least privilege 94
Server not found in Kerebos database
SPN error 31
service account
and SPN mapping 11
service principal account
for VSJ 27
in VSJ 11
service principal name 11
service principal name mapping
for VSJ 30
services selection
on remote computer 38
on VSJ object 39
servlet filter
configuration for VSJ Federation 80
Servlets 2
session state 95
setspn
-A command 31
-F -S command (2008 only) 31
-S command (2008 only) 31
usage 31
signoutImage 82
smartcard
Active Directory, support in 7
specified services delegation 37
SPN 11
duplicates 31
SPN mapping
and DNS aliases 31
for VSJ user 30
to computer object 11
to service account 11
SPN selection
for constrained delegation 14
SPNEGO 19, 88, 89
and HTTP Negotiation 12
and IE 2
and Internet Explorer 12
and other browsers 12
fallback to NTLM 96
SRV resource records
support in DNS 25
SSL
and cookies 87
system properties
com.wedgetail.idm.sso.password 117
jcsi.kerberos.nameservers 117

T

TGT 4
used in unconstrained delegation 15
Ticket Granting Ticket 4
time synchronization service required 26
topology changes 100
troubleshooting 100, 101
trust and delegation 14

U

unconstrained delegation
and TGTs 15
cf constrained 15
configuring in Windows 2000 35
configuring in Windows 2003,2008 37
unprivileged user
creating VSJ account 33
updated membership and cacheing 100
UPN 10
URL rewriting
cookies alternative 88
user account for VSJ
creating 27
creation using vastool 32
naming convention 27
User Principal Name 10
using the VAS host with VSJ 32

V

VAS 11
and Active Directory accounts 31
keytab files 32
using HOST principal with VSJ 32
vastool
and HTTP service principal creation 33
creating accounts and keytabs 32
Vintela Authentication Services 11

VSJ
  Active Directory groups 9
  Active Directory, and 12
  advantages of vii
  and auditing 95
  and delegation 13, 35
  and denial of service attacks 94
  and Java application servers 2
  and Java EE 2
  and VAS 32
  configuration requirements 27
  deployment risks 86, 87
  how it works 17
  overview 2
  purpose vii
  SPN mapping for 30
VSJ account
  creation 27
  setup 27
VSJ configuration parameters 82, 110
VSJ Federation
  and ADFS 77
  configuring servlet filter 80
  deploying license 79
  features of 78
  installation process 80
  known issues 83
  library requirements for 78
  Oracle configuration 79
  prerequisites 78
  WebLogic configuration 79
VSJ service principal account
  setting up 27
VSJ service principal name 30

W
WARN level
  recommended for logging 95
WEB-INF deployment
  VSJ Federation 79
WebLogic configuration
  VSJ Federation 79
WebSphere
  overriding JDK 1.4 libraries 80

X
Xalan and Xerces libraries 78
Xerces and Xalan libraries 78
XML libraries
  for VSJ Federation 79