Spring Security Kerberos - Reference Documentation

1.0.1.RELEASE

Janne Valkealahti Pivotal

Table of Contents

Preface

This reference documentations contains following parts.

Part I, “Introduction” introduction to this reference documentation

Part II, “Spring and Spring Security Kerberos” describes the usage of Spring Security Kerberos

Part III, “Spring Security Kerberos Samples” describes the usage of Spring Security Kerberos Samples

Part IV, “Appendices” generic support material

Part I. Introduction

Spring Security Kerberos is an extension of Spring Security for application developers to Kerberos concepts with Spring.

1. Requirements

Spring Security Kerberos 1.0.1.RELEASE is built and tested with JDK 7, Spring Security 3.2.7.RELEASE and Spring Framework 4.1.6.RELEASE.

Part II. Spring and Spring Security Kerberos

This part of the reference documentation explains the core functionality that Spring Security Kerberos provides to any Spring based application.

Chapter 2, *Authentication Provider* describes the authentication provider support.

Chapter 3, *Spnego Negotiate* describes the spnego negotiate support.

Chapter 4, *Using KerberosRestTemplate* describes the RestTemplate support.

2. Authentication Provider

Provider configuration using JavaConfig.

```
@Configuration
@EnableWebMvcSecurity
public class AuthProviderConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/", "/home").permitAll()
                .anyRequest().authenticated()
                .and()
            .formLogin()
                .loginPage("/login").permitAll()
                .and()
            .logout()
                .permitAll();
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth
            .authenticationProvider(kerberosAuthenticationProvider());
    }

    @Bean
    public KerberosAuthenticationProvider kerberosAuthenticationProvider() {
        KerberosAuthenticationProvider provider =
          new KerberosAuthenticationProvider();
        SunJaasKerberosClient client = new SunJaasKerberosClient();
        client.setDebug(true);
        provider.setKerberosClient(client);
        provider.setUserDetailsService(dummyUserDetailsService());
        return provider;
    }

    @Bean
    public DummyUserDetailsService dummyUserDetailsService() {
        return new DummyUserDetailsService();
    }

}
```

Provider configuration using xml.

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sec="http://www.springframework.org/schema/security"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="
    http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-
context-3.2.xsd
    http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-
beans-3.2.xsd
    http://www.springframework.org/schema/security http://www.springframework.org/schema/security/
spring-security.xsd">

  <sec:http entry-point-ref="spnegoEntryPoint" use-expressions="true">
    <sec:intercept-url pattern="/" access="permitAll" />
    <sec:intercept-url pattern="/home" access="permitAll" />
    <sec:intercept-url pattern="/**" access="authenticated"/>
  </sec:http>

  <sec:authentication-manager alias="authenticationManager">
    <sec:authentication-provider ref="kerberosAuthenticationProvider"/>
  </sec:authentication-manager>

  <bean id="kerberosAuthenticationProvider"
    class="org.springframework.security.kerberos.authentication.KerberosAuthenticationProvider">
    <property name="kerberosClient">
      <bean class="org.springframework.security.kerberos.authentication.sun.SunJaasKerberosClient">
        <property name="debug" value="true"/>
      </bean>
    </property>
    <property name="userDetailsService" ref="dummyUserDetailsService"/>
  </bean>

  <bean
    class="org.springframework.security.kerberos.authentication.sun.GlobalSunJaasKerberosConfig">
    <property name="debug" value="true" />
    <property name="krbConfLocation" value="/path/to/krb5.ini"/>
  </bean>

  <bean id="dummyUserDetailsService"
    class="org.springframework.security.kerberos.docs.DummyUserDetailsService" />

  <bean id="spnegoEntryPoint"
    class="org.springframework.security.kerberos.web.authentication.SpnegoEntryPoint" >
    <constructor-arg value="/login" />
  </bean>

</beans>
```

3. Spnego Negotiate

Spnego configuration using JavaConfig.

```
@Configuration
@EnableWebMvcSecurity
public class SpnegoConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .exceptionHandling()
                .authenticationEntryPoint(spnegoEntryPoint())
                .and()
            .authorizeRequests()
                .antMatchers("/", "/home").permitAll()
                .anyRequest().authenticated()
                .and()
            .formLogin()
                .loginPage("/login").permitAll()
                .and()
            .logout()
                .permitAll()
                .and()
            .addFilterBefore(
                    spnegoAuthenticationProcessingFilter(authenticationManagerBean()),
                    BasicAuthenticationFilter.class);
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth)
      throws Exception {
        auth
            .authenticationProvider(kerberosAuthenticationProvider())
            .authenticationProvider(kerberosServiceAuthenticationProvider());
    }

    @Bean
    public KerberosAuthenticationProvider kerberosAuthenticationProvider() {
        KerberosAuthenticationProvider provider =
          new KerberosAuthenticationProvider();
        SunJaasKerberosClient client = new SunJaasKerberosClient();
        client.setDebug(true);
        provider.setKerberosClient(client);
        provider.setUserDetailsService(dummyUserDetailsService());
        return provider;
    }

    @Bean
    public SpnegoEntryPoint spnegoEntryPoint() {
        return new SpnegoEntryPoint("/login");
    }

    @Bean
    public SpnegoAuthenticationProcessingFilter spnegoAuthenticationProcessingFilter(
            AuthenticationManager authenticationManager) {
        SpnegoAuthenticationProcessingFilter filter =
          new SpnegoAuthenticationProcessingFilter();
        filter.setAuthenticationManager(authenticationManager);
        return filter;
    }

    @Bean
    public KerberosServiceAuthenticationProvider kerberosServiceAuthenticationProvider() {
        KerberosServiceAuthenticationProvider provider =
          new KerberosServiceAuthenticationProvider();
        provider.setTicketValidator(sunJaasKerberosTicketValidator());
        provider.setUserDetailsService(dummyUserDetailsService());
        return provider;
    }

    @Bean
    public SunJaasKerberosTicketValidator sunJaasKerberosTicketValidator() {
        SunJaasKerberosTicketValidator ticketValidator =
          new SunJaasKerberosTicketValidator();
        ticketValidator.setServicePrincipal("HTTP/servicehost.example.org@EXAMPLE.ORG");
        ticketValidator.setKeyTabLocation(new FileSystemResource("/tmp/service.keytab"));
        ticketValidator.setDebug(true);
        return ticketValidator;
    }

    @Bean
```

Spnego configuration using xml.

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:sec="http://www.springframework.org/schema/security"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/security http://www.springframework.org/
schema/security/spring-security.xsd
    http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-
beans-4.1.xsd
    http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-
context-4.1.xsd">

  <sec:http entry-point-ref="spnegoEntryPoint" use-expressions="true" >
    <sec:intercept-url pattern="/" access="permitAll" />
    <sec:intercept-url pattern="/home" access="permitAll" />
    <sec:intercept-url pattern="/login" access="permitAll" />
    <sec:intercept-url pattern="/**" access="authenticated"/>
    <sec:form-login login-page="/login" />
    <sec:custom-filter ref="spnegoAuthenticationProcessingFilter"
      before="BASIC_AUTH_FILTER" />
  </sec:http>

  <sec:authentication-manager alias="authenticationManager">
    <sec:authentication-provider ref="kerberosAuthenticationProvider" />
    <sec:authentication-provider ref="kerberosServiceAuthenticationProvider" />
  </sec:authentication-manager>

  <bean id="kerberosAuthenticationProvider"
    class="org.springframework.security.kerberos.authentication.KerberosAuthenticationProvider">
    <property name="userDetailsService" ref="dummyUserDetailsService"/>
    <property name="kerberosClient">
      <bean class="org.springframework.security.kerberos.authentication.sun.SunJaasKerberosClient">
        <property name="debug" value="true"/>
      </bean>
    </property>
  </bean>

  <bean id="spnegoEntryPoint"
    class="org.springframework.security.kerberos.web.authentication.SpnegoEntryPoint" >
    <constructor-arg value="/login" />
  </bean>

  <bean id="spnegoAuthenticationProcessingFilter"

  class="org.springframework.security.kerberos.web.authentication.SpnegoAuthenticationProcessingFilter">
    <property name="authenticationManager" ref="authenticationManager" />
  </bean>

  <bean id="kerberosServiceAuthenticationProvider"
    class="org.springframework.security.kerberos.authentication.KerberosServiceAuthenticationProvider">
    <property name="ticketValidator">
      <bean
        class="org.springframework.security.kerberos.authentication.sun.SunJaasKerberosTicketValidator">
        <property name="servicePrincipal" value="${app.service-principal}" />
        <property name="keyTabLocation" value="${app.keytab-location}" />
        <property name="debug" value="true" />
      </bean>
    </property>
    <property name="userDetailsService" ref="dummyUserDetailsService" />
  </bean>

  <bean id="dummyUserDetailsService"
    class="org.springframework.security.kerberos.docs.DummyUserDetailsService" />

</beans>
```

4. Using KerberosRestTemplate

If there is a need to access Kerberos protected web resources programmatically we have `KerberosRestTemplate` which extends `RestTemplate` and does necessary login actions prior to delegating to actual RestTemplate methods. You basically have few options to configure this template.

- Leave keyTabLocation and userPrincipal empty if you want to use cached ticket.
- Use keyTabLocation and userPrincipal if you want to use keytab file.
- Use loginOptions if you want to customise Krb5LoginModule options.
- Use a customised httpClient.

With ticket cache.

```
public void doWithTicketCache() {
    KerberosRestTemplate restTemplate =
            new KerberosRestTemplate();
    restTemplate.getForObject("http://neo.example.org:8080/hello", String.class);
}
```

With keytab file.

```
public void doWithKeytabFile() {
    KerberosRestTemplate restTemplate =
            new KerberosRestTemplate("/tmp/user2.keytab", "user2@EXAMPLE.ORG");
    restTemplate.getForObject("http://neo.example.org:8080/hello", String.class);
}
```

5. Authentication with LDAP Services

With most of your samples we're using `DummyUserDetailsService` because there is not necessarily need to query a real user details once kerberos authentication is successful and we can use kerberos principal info to create that dummy user. However there is a way to access kerberized LDAP services in a say way and query user details from there.

`KerberosLdapContextSource` can be used to bind into LDAP via kerberos which is at least proven to work well with Windows AD services.

```
@Value("${app.ad-server}")
private String adServer;

@Value("${app.service-principal}")
private String servicePrincipal;

@Value("${app.keytab-location}")
private String keytabLocation;

@Value("${app.ldap-search-base}")
private String ldapSearchBase;

@Value("${app.ldap-search-filter}")
private String ldapSearchFilter;

@Bean
public KerberosLdapContextSource kerberosLdapContextSource() {
    KerberosLdapContextSource contextSource = new KerberosLdapContextSource(adServer);
    SunJaasKrb5LoginConfig loginConfig = new SunJaasKrb5LoginConfig();
    loginConfig.setKeyTabLocation(new FileSystemResource(keytabLocation));
    loginConfig.setServicePrincipal(servicePrincipal);
    loginConfig.setDebug(true);
    loginConfig.setIsInitiator(true);
    contextSource.setLoginConfig(loginConfig);
    return contextSource;
}

@Bean
public LdapUserDetailsService ldapUserDetailsService() {
    FilterBasedLdapUserSearch userSearch =
            new FilterBasedLdapUserSearch(ldapSearchBase, ldapSearchFilter,
 kerberosLdapContextSource());
    LdapUserDetailsService service = new LdapUserDetailsService(userSearch);
    service.setUserDetailsMapper(new LdapUserDetailsMapper());
    return service;
}
```

> **Tip**
>
> Sample Chapter 6, *Security Server Windows Auth Sample* is currently configured to query user details from AD if authentication happen via kerberos.

Part III. Spring Security Kerberos Samples

This part of the reference documentation is introducing samples projects. Samples can be compiled manually by building main distribution from https://github.com/spring-projects/spring-security-kerberos.

> **Important**
>
> If you run sample as is it will not work until a correct configuration is applied. See notes below for specific samples.

Chapter 6, *Security Server Windows Auth Sample* sample for Windows environment

Chapter 7, *Security Server Side Auth Sample* sample using server side authenticator

Chapter 8, *Security Server Spnego and Form Auth Sample* sample using ticket validation with spnego and form

Chapter 9, *Security Server Spnego and Form Auth Xml Sample* sample using ticket validation with spnego and form (xml config)

Chapter 10, *Security Client KerberosRestTemplate Sample* sample for KerberosRestTemplate

6. Security Server Windows Auth Sample

Goals of this sample:

- In windows environment, User will be able to logon to application with Windows Active directory Credential which has been entered during log on to windows. There should not be any ask for userid/ password credentials.
- In non-windows environment, User will be presented with a screen to provide Active directory credentials.

```
server:
    port: 8080
    app:
        ad-domain: EXAMPLE.ORG
        ad-server: ldap://WIN-EKBO0EQ7TS7.example.org/
        service-principal: HTTP/neo.example.org@EXAMPLE.ORG
        keytab-location: /tmp/tomcat.keytab
        ldap-search-base: dc=example,dc=org
        ldap-search-filter: "(| (userPrincipalName={0}) (sAMAccountName={0}))"
```

In above you can see the default configuration for this sample. You can override these settings using a normal Spring Boot tricks like using command-line options or custom `application.yml` file.

Run a server.

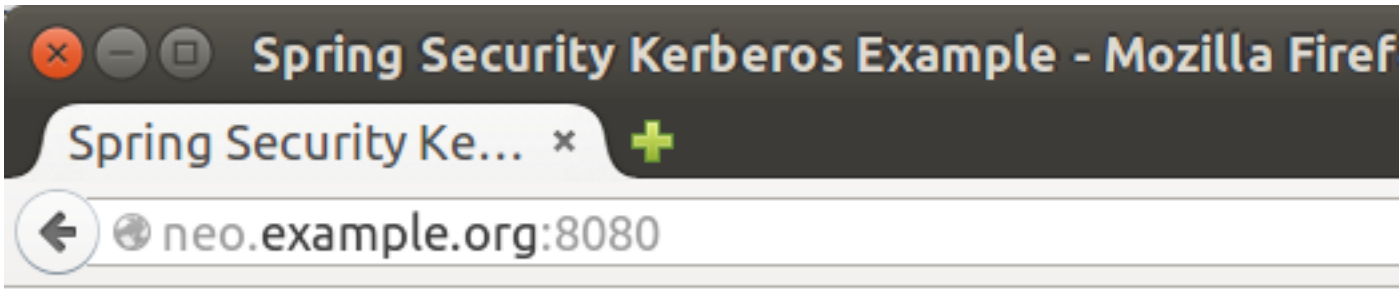
```
$ java -jar sec-server-win-auth-1.0.1.RELEASE.jar
```

> **Important**
>
> You may need to use custom kerberos config with Linux either by using `-Djava.security.krb5.conf=/path/to/krb5.ini` or `GlobalSunJaasKerberosConfig` bean.

> **Note**
>
> See Section C.2, “Setup Windows Domain Controller” for more instructions how to work with windows kerberos environment.

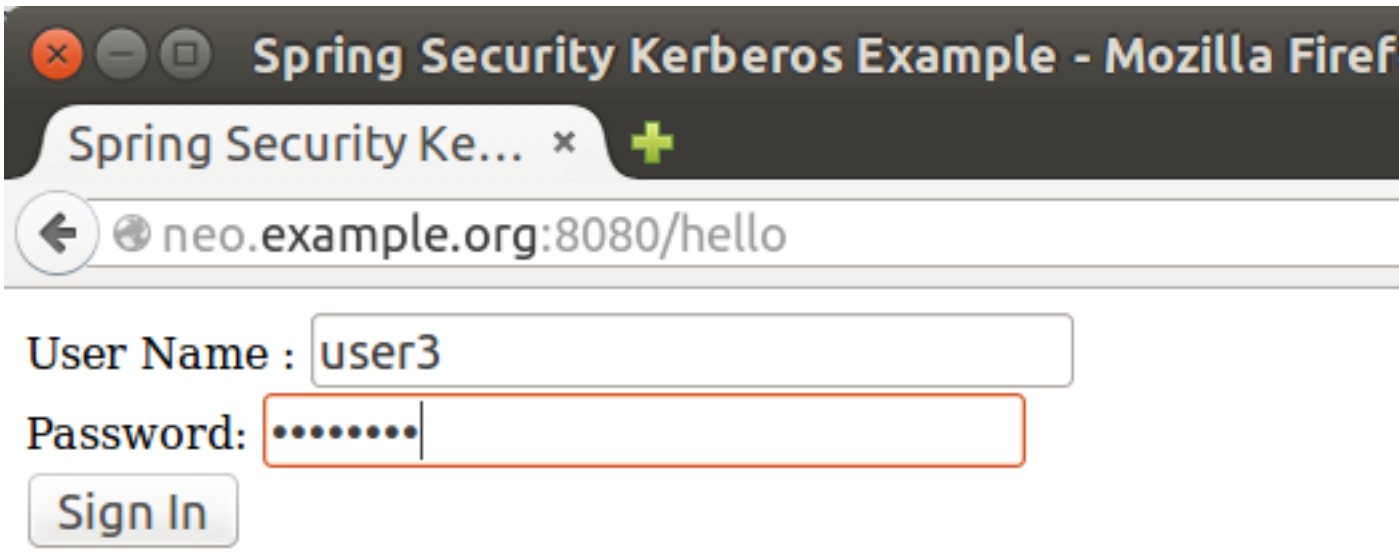
Login to `Windows 8.1` using domain credentials and access sample

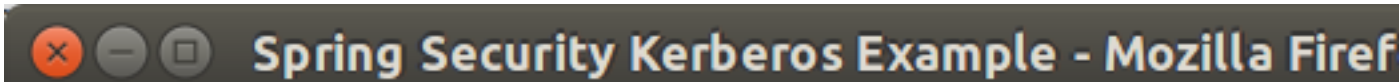
Access sample application from a non windows vm and use domain credentials manually.



Welcome!

Click here to see a greeting.





7. Security Server Side Auth Sample

This sample demonstrates how server is able to authenticate user against kerberos environment using his credentials passed in via a form login.

Run a server.

```
$ java -jar sec-server-client-auth-1.0.1.RELEASE.jar
```

```
server:
    port: 8080
```

8. Security Server Spnego and Form Auth Sample

This sample demonstrates how a server can be configured to accept a Spnego based negotiation from a browser while still being able to fall back to a form based authentication.

Using a `user1` principal Section C.1, “Setup MIT Kerberos”, do a kerberos login manually using credentials.

```
$ kinit user1
Password for user1@EXAMPLE.ORG:

$ klist
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: user1@EXAMPLE.ORG

Valid starting     Expires            Service principal
10/03/15 17:18:45  11/03/15 03:18:45  krbtgt/EXAMPLE.ORG@EXAMPLE.ORG
  renew until 11/03/15 17:18:40
```

or using a keytab file.

```
$ kinit -kt user2.keytab user1

$ klist
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: user2@EXAMPLE.ORG

Valid starting     Expires            Service principal
10/03/15 17:25:03  11/03/15 03:25:03  krbtgt/EXAMPLE.ORG@EXAMPLE.ORG
  renew until 11/03/15 17:25:03
```

Run a server.

```
$ java -jar sec-server-spnego-form-auth-1.0.1.RELEASE.jar
```

Now you should be able to open your browser and let it do Spnego authentication with existing ticket.

> **Note**
>
> See Appendix E, *Configure Browsers for Spnego Negotiation* for more instructions for configuring browsers to use Spnego.

```
server:
    port: 8080
app:
    service-principal: HTTP/neo.example.org@EXAMPLE.ORG
    keytab-location: /tmp/tomcat.keytab
```

9. Security Server Spnego and Form Auth Xml Sample

This is a same sample than Chapter 8, *Security Server Spnego and Form Auth Sample* but using xml based configuration instead of JavaConfig.

Run a server.

```
$ java -jar sec-server-spnego-form-auth-xml-1.0.1.RELEASE.jar
```

10. Security Client KerberosRestTemplate Sample

This is a sample using a Spring RestTemplate to access Kerberos protected resource. You can use this together with Chapter 8, *Security Server Spnego and Form Auth Sample*.

Default application is configured as shown below.

```
app:
    user-principal: user2@EXAMPLE.ORG
    keytab-location: /tmp/user2.keytab
    access-url: http://neo.example.org:8080/hello
```

Using a `user1` principal Section C.1, “Setup MIT Kerberos”, do a kerberos login manually using credentials.

```
$ java -jar sec-client-rest-template-1.0.1.RELEASE.jar --app.user-principal --app.keytab-location
```

> **Note**
>
> In above we simply set `app.user-principal` and `app.keytab-location` to empty values which disables a use of keytab file.

If operation is succesfull you should see below output with `user1@EXAMPLE.ORG`.

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
  <head>
    <title>Spring Security Kerberos Example</title>
  </head>
  <body>
    <h1>Hello user1@EXAMPLE.ORG!</h1>
  </body>
</html>
```

Or use a `user2` with a keytab file.

```
$ java -jar sec-client-rest-template-1.0.1.RELEASE.jar
```

If operation is succesfull you should see below output with `user2@EXAMPLE.ORG`.

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
  <head>
    <title>Spring Security Kerberos Example</title>
  </head>
  <body>
    <h1>Hello user2@EXAMPLE.ORG!</h1>
  </body>
</html>
```

Part IV. Appendices

Appendix A. Material Used in this Document

Dummy UserDetailsService used in samples because we don’t have a real user source.

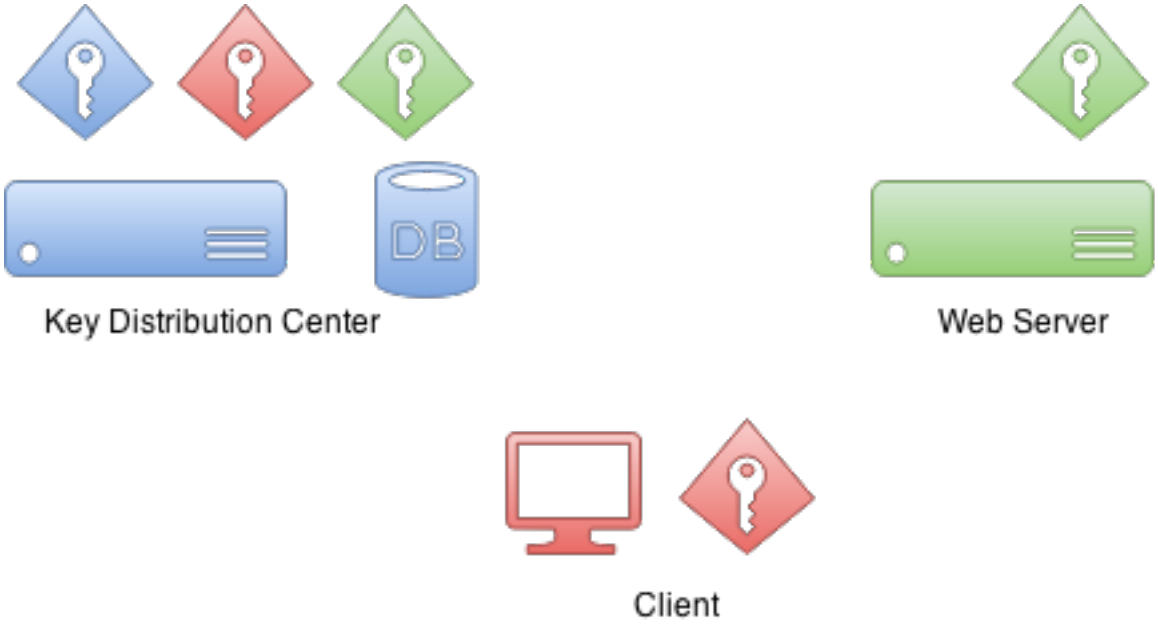
```
public class DummyUserDetailsService implements UserDetailsService {

    @Override
    public UserDetails loadUserByUsername(String username)
            throws UsernameNotFoundException {
        return new User(username, "notUsed", true, true, true, true,
                AuthorityUtils.createAuthorityList("ROLE_USER"));
    }

}
```

Appendix B. Crash Course to Kerberos

In any authentication process there are usually a three parties involved.

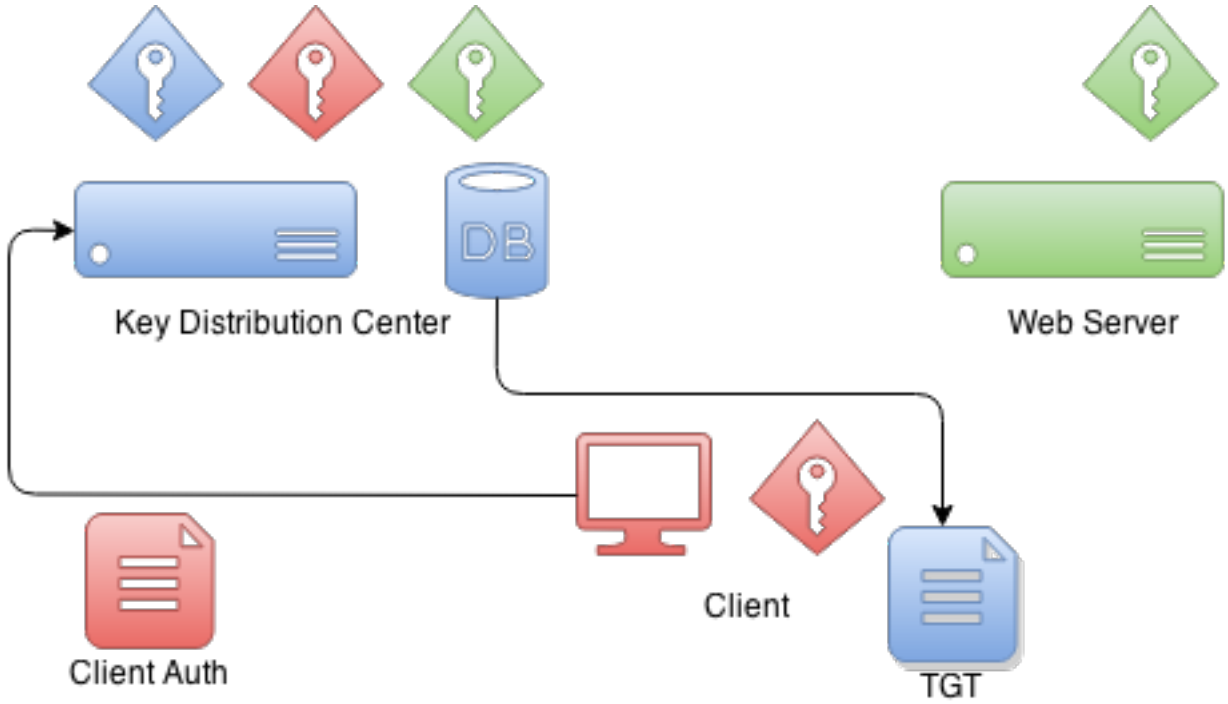


First is a `client` which sometimes is a client computer but in most of the scenarios it is the actual user sitting on a computer and trying to access resources. Then there is the `resource` user is trying to access. In this example it is a web server.

Then there is a `Key Distribution Center` or `KDC`. In a case of Windows environment this would be a `Domain Controller`. `KDC` is the one which really brings everything together and thus is the most critical component in your environment. Because of this it is also considered as a single point of failure.

Initially when `Kerberos` environment is setup and domain user principals created into a database, encryption keys are also created. These encryption keys are based on shared secrets(i.e. user password) and actual passwords are never kept in a clear text. Effectively `KDC` has its own key and other keys for domain users.

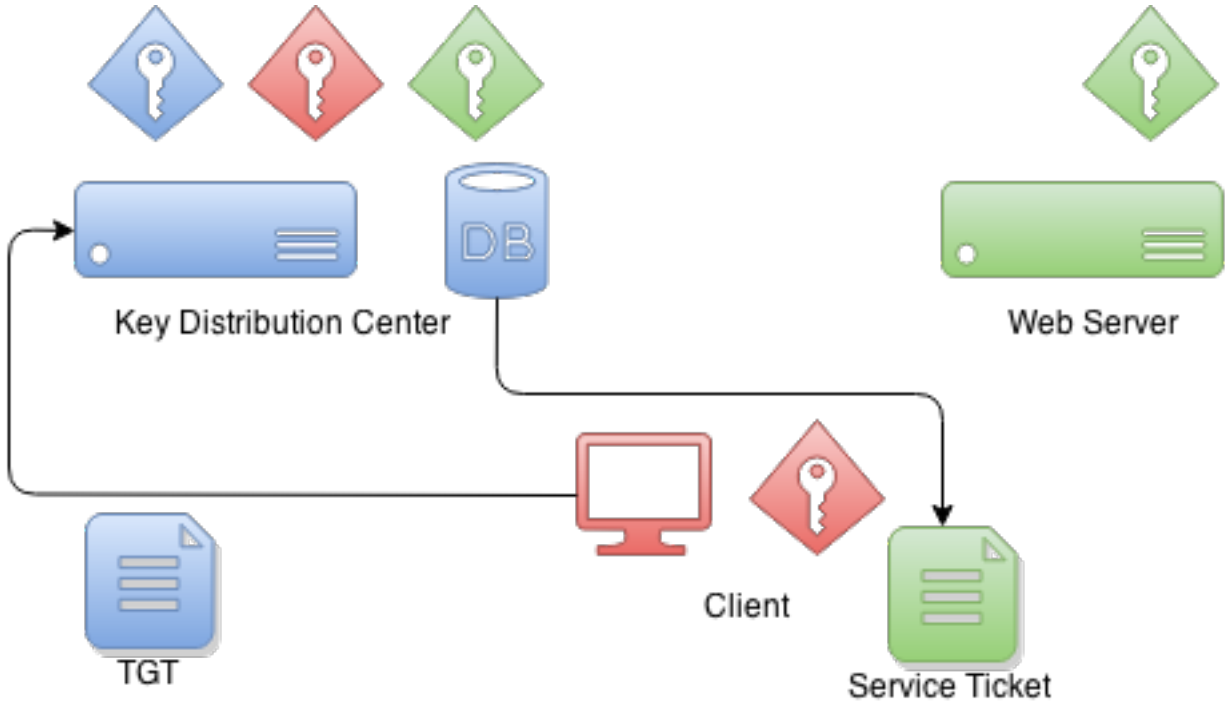
Interestingly there is no communication between a `resource` and a `KDC` during the authentication process.



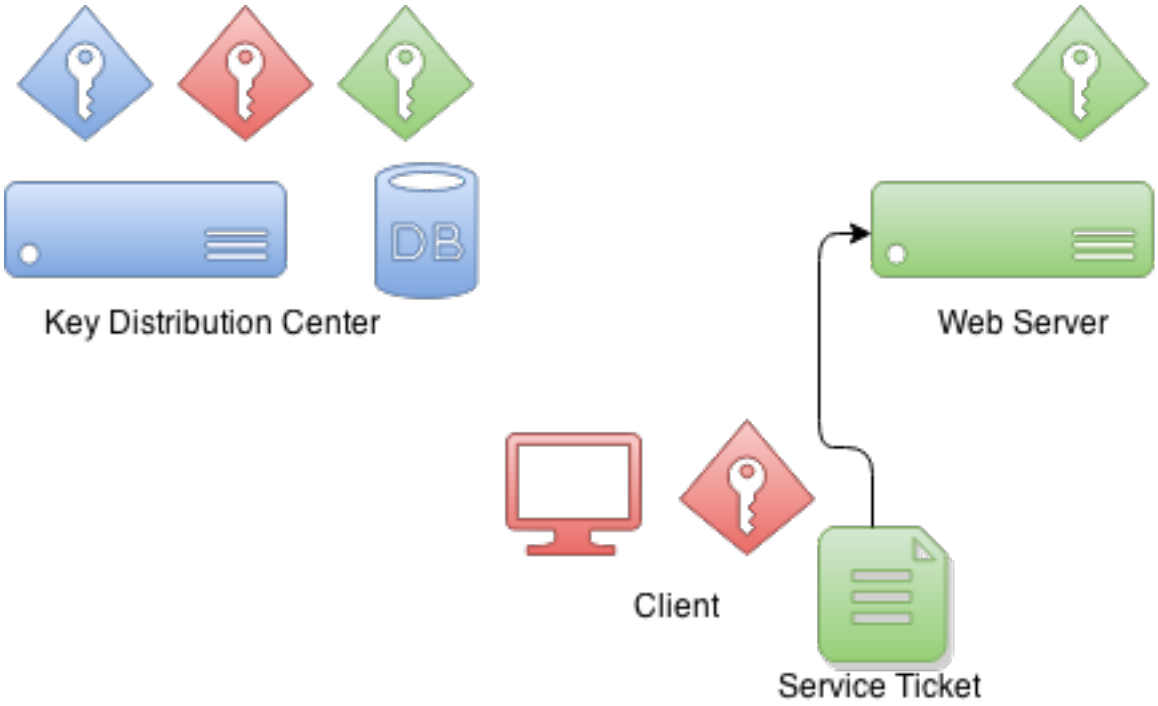
When client wants to authenticate itself with a `resource` it first needs to communicate with a KDC. `Client` will craft a special package which contains encrypted and unencrypted parts. Unencrypted part contains i.e. information about a user and encrypted part other information which is part of a protocol. `Client` will encrypt package data with its own key.

When KDC receives this authentication package from a client it checks who this `client` claims to be from an unencrypted part and based on that information it uses `client` decryption key it already have in its database. If this decryption is succesfull KDC knows that this `client` is the one it claims to be.

What KDC returns to a client is a ticket called `Ticket Granting Ticket` which is signed by a KDC's own private key. Later when `client` sends back this ticket it can try to decrypt it and if that operation is succesfull it knows that it was a ticket it itself originally signed and gave to a `client`.



When client wants to get a ticket which it can use to authenticate with a service, `TGT` is send to `KDC` which then signs a service ticket with service's own key. This a moment when a trust between `client` and `service` is created. This service ticket contains data which only `service` itself is able to decrypt.



When `client` is authenticating with a service it sends previously received service ticket to a service which then thinks that I don't know anything about this guy but he have me an authentication ticket. What `service` can do next is try to decrypt that ticket and if that operation is succesfull it knows that only other party who knows my credentials is the `KDC` and because I trust him I can also trust that this client is a one he claims to be.

Appendix C. Setup Kerberos Environments

Doing a production setup of Kerberos environment is out of scope of this document but this appendix provides some help to get you started for setting up needed components for development.

C.1 Setup MIT Kerberos

First action is to setup a new realm and a database.

```
# kdb5_util create -s -r EXAMPLE.ORG
Loading random data
Initializing database '/var/lib/krb5kdc/principal' for realm 'EXAMPLE.ORG',
master key name 'K/M@EXAMPLE.ORG'
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.
Enter KDC database master key:
Re-enter KDC database master key to verify:
```

`kadmin` command can be used to administer Kerberos environment but you can't yet use it because there are no admin users in a database.

```
root@neo:/etc/krb5kdc# kadmin
Authenticating as principal root/admin@EXAMPLE.ORG with password.
kadmin: Client not found in Kerberos database while initializing
kadmin interface
```

Lets use `kadmin.local` command to create one.

```
root@neo:/etc/krb5kdc# kadmin.local
Authenticating as principal root/admin@EXAMPLE.ORG with password.

kadmin.local:  listprincs
K/M@EXAMPLE.ORG
kadmin/admin@EXAMPLE.ORG
kadmin/changepw@EXAMPLE.ORG
kadmin/cypher@EXAMPLE.ORG
krbtgt/EXAMPLE.ORG@EXAMPLE.ORG

kadmin.local:  addprinc root/admin@EXAMPLE.ORG
WARNING: no policy specified for root/admin@EXAMPLE.ORG; defaulting to
no policy
Enter password for principal "root/admin@EXAMPLE.ORG":
Re-enter password for principal "root/admin@EXAMPLE.ORG":
Principal "root/admin@EXAMPLE.ORG" created.
```

Then enable admins by modifying `kadm5.acl` file and restart Kerberos services.

```
# cat /etc/krb5kdc/kadm5.acl
# This file Is the access control list for krb5 administration.
*/admin *
```

Now you can use `kadmin` with previously created `root/admin` principal. Lets create our first user `user1`.

```
kadmin:  addprinc user1
WARNING: no policy specified for user1@EXAMPLE.ORG; defaulting to no
policy
Enter password for principal "user1@EXAMPLE.ORG":
Re-enter password for principal "user1@EXAMPLE.ORG":
Principal "user1@EXAMPLE.ORG" created.
```

Lets create our second user `user2` and export a keytab file.

```
kadmin:  addprinc user2
WARNING: no policy specified for user2@EXAMPLE.ORG; defaulting to no
policy
Enter password for principal "user2@EXAMPLE.ORG":
Re-enter password for principal "user2@EXAMPLE.ORG":
Principal "user2@EXAMPLE.ORG" created.

kadmin:  ktadd -k /tmp/user2.keytab user2@EXAMPLE.ORG
Entry for principal user2@EXAMPLE.ORG with kvno 2, encryption type aes256-cts-hmac-sha1-96 added to
 keytab WRFILE:/tmp/user2.keytab.
Entry for principal user2@EXAMPLE.ORG with kvno 2, encryption type arcfour-hmac added to keytab WRFILE:/
tmp/user2.keytab.
Entry for principal user2@EXAMPLE.ORG with kvno 2, encryption type des3-cbc-sha1 added to keytab
 WRFILE:/tmp/user2.keytab.
Entry for principal user2@EXAMPLE.ORG with kvno 2, encryption type des-cbc-crc added to keytab WRFILE:/
tmp/user2.keytab.
```

Lets create a service ticket for tomcat and export credentials to a keytab file named `tomcat.keytab`.

```
kadmin:  addprinc -randkey HTTP/neo.example.org@EXAMPLE.ORG
WARNING: no policy specified for HTTP/neo.example.org@EXAMPLE.ORG;
defaulting to no policy
Principal "HTTP/neo.example.org@EXAMPLE.ORG" created.

kadmin:  ktadd -k /tmp/tomcat.keytab HTTP/neo.example.org@EXAMPLE.ORG
Entry for principal HTTP/neo.example.org@EXAMPLE.ORG with kvno 2, encryption type aes256-cts-hmac-
sha1-96 added to keytab WRFILE:/tmp/tomcat2.keytab.
Entry for principal HTTP/neo.example.org@EXAMPLE.ORG with kvno 2, encryption type arcfour-hmac added to
 keytab WRFILE:/tmp/tomcat2.keytab.
Entry for principal HTTP/neo.example.org@EXAMPLE.ORG with kvno 2, encryption type des3-cbc-sha1 added to
 keytab WRFILE:/tmp/tomcat2.keytab.
Entry for principal HTTP/neo.example.org@EXAMPLE.ORG with kvno 2, encryption type des-cbc-crc added to
 keytab WRFILE:/tmp/tomcat2.keytab.
```

C.2 Setup Windows Domain Controller

This was tested using `Windows Server 2012 R2`

> **Tip**
>
> Internet is full of good articles and videos how to setup Windows AD but these two are quite usefull Rackspace and Microsoft Technet.

- Normal domain controller and active directory setup was done.
- Used dns domain `example.org` and windows domain `EXAMPLE`.
- I created various domain users like `user1`, `user2`, `user3`, `tomcat` and set passwords to `Password#`.

I eventually also added all ip's of my vm's to AD's dns server for that not to cause any trouble.

```
Name: WIN-EKBO0EQ7TS7.example.org
Address: 172.16.101.135

Name: win8vm.example.org
Address: 172.16.101.136

Name: neo.example.org
Address: 172.16.101.1
```

Service Principal Name(SPN) needs to be setup with `HTTP` and a server name `neo.example.org` where tomcat servlet container is run. This is used with `tomcat` domain user and its `keytab` is then used as a service credential.

```
PS C:\> setspn -A HTTP/neo.example.org tomcat
```

I exported keytab file which is copied to linux server running tomcat.

```
PS C:\> ktpass /out c:\tomcat.keytab /mapuser tomcat@EXAMPLE.ORG /princ HTTP/
neo.example.org@EXAMPLE.ORG /pass Password# /ptype KRB5_NT_PRINCIPAL /crypto All
 Targeting domain controller: WIN-EKBO0EQ7TS7.example.org
 Using legacy password setting method
 Successfully mapped HTTP/neo.example.org to tomcat.
```

Appendix D. Troubleshooting

This appendix provides generic information about troubleshooting errors and problems.

> **Important**
>
> If you think environment and configuration is correctly setup, do double check and ask other person to check possible obvious mistakes or typos. Kerberos setup is generally very brittle and it is not always very easy to debug where the problem lies.

Cannot find key of appropriate type to decrypt.

```
GSSException: Failure unspecified at GSS-API level (Mechanism level:
Invalid argument (400) - Cannot find key of appropriate type to
decrypt AP REP - RC4 with HMAC)
```

If you see abore error indicating missing key type, this will happen with two different use cases. Firstly your JVM may not support appropriate encryption type or it is disabled in your `krb5.conf` file.

```
default_tkt_enctypes = rc4-hmac
default_tgs_enctypes = rc4-hmac
```

Second case is less obvious and hard to track because it will lead into same error. This specific `GSSException` is throws also if you simply don't have a required encryption key which then may be caused by a misconfiguration in your kerberos server or a simply typo in your principal.

Using wrong kerberos configuration.

In most system all commands and libraries will search kerberos configuration either from a default locations or special locations like JDKs. It's easy to get mixed up especially if working from unix systems, which already may have default settings to work with MIT kerberos, towards Windows domains.

This is a specific example what happens with `ldapsearch` trying to query Windows AD using kerberos authentication.

```
$ ldapsearch -H ldap://WIN-EKBO0EQ7TS7.example.org -b "dc=example,dc=org"
SASL/GSSAPI authentication started
ldap_sasl_interactive_bind_s: Local error (-2)
  additional info: SASL(-1): generic failure: GSSAPI Error:
  Unspecified GSS failure.  Minor code may provide more information
  (No Kerberos credentials available)
```

Well that doesn't look good and is a simple indication that I don't have a valid kerberos tickets as shown below.

```
$ klist
klist: Credentials cache file '/tmp/krb5cc_1000' not found
```

We already have a keytab file we exported from Windows AD to be used with tomcat running on Linux. Lets try to use that to authenticate with Windows AD.

You can have a dedicated config file which usually can be used with native Linux commands and JVMs via system propertys.

```
$ cat krb5.ini
[libdefaults]
default_realm = EXAMPLE.ORG
default_keytab_name = /tmp/tomcat.keytab
forwardable=true

[realms]
EXAMPLE.ORG = {
  kdc = WIN-EKBO0EQ7TS7.example.org:88
}

[domain_realm]
example.org=EXAMPLE.ORG
.example.org=EXAMPLE.ORG
```

Lets use that config and a keytab to get initial credentials.

```
$ env KRB5_CONFIG=/path/to/krb5.ini kinit -kt tomcat.keytab HTTP/neo.example.org@EXAMPLE.ORG

$ klist
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: HTTP/neo.example.org@EXAMPLE.ORG

Valid starting     Expires            Service principal
26/03/15 09:04:37  26/03/15 19:04:37  krbtgt/EXAMPLE.ORG@EXAMPLE.ORG
  renew until 27/03/15 09:04:37
```

Lets see what happens if we now try to do a simple query against Windows AD.

```
$ ldapsearch -H ldap://WIN-EKBO0EQ7TS7.example.org -b "dc=example,dc=org"
SASL/GSSAPI authentication started
ldap_sasl_interactive_bind_s: Local error (-2)
  additional info: SASL(-1): generic failure: GSSAPI Error:
  Unspecified GSS failure.  Minor code may provide more information
  (KDC returned error string: PROCESS_TGS)
```

This may be simply because `ldapsearch` is getting confused and simply using wrong configuration. You can tell `ldapsearch` to use a different configuration via `KRB5_CONFIG` env variable just like we did with `kinit`. You can also use `KRB5_TRACE=/dev/stderr` to get more verbose output of what native libraries are doing.

```
$ env KRB5_CONFIG=/path/to/krb5.ini ldapsearch -H ldap://WIN-EKBO0EQ7TS7.example.org -b
 "dc=example,dc=org"

$ klist
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: HTTP/neo.example.org@EXAMPLE.ORG

Valid starting     Expires            Service principal
26/03/15 09:11:03  26/03/15 19:11:03  krbtgt/EXAMPLE.ORG@EXAMPLE.ORG
  renew until 27/03/15 09:11:03
  26/03/15 09:11:44  26/03/15 19:11:03
  ldap/win-ekbo0eq7ts7.example.org@EXAMPLE.ORG
    renew until 27/03/15 09:11:03
```

Above you can see what happened if query was successful by looking kerberos tickets. Now you can experiment with further query commands i.e. if you working with `KerberosLdapContextSource`.

```
$ ldapsearch -H ldap://WIN-EKBO0EQ7TS7.example.org \
-b "dc=example,dc=org" \
"(| (userPrincipalName=user2@EXAMPLE.ORG)
(sAMAccountName=user2@EXAMPLE.ORG))" \
dn

...
# test user, example.org
dn: CN=test user,DC=example,DC=org
```

Appendix E. Configure Browsers for Spnego Negotiation

E.1 Firefox

Complete following steps to ensure that your Firefox browser is enabled to perform Spnego authentication.

- Open Firefox.
- At address field, type **about:config**.
- In filter/search, type **negotiate**.
- Parameter **network.negotiate-auth.trusted-uris** may be set to default **https://** which doesn't work for you. Generally speaking this parameter has to replaced with the server address if Kerberos delegation is required.
- It is recommended to use `https` for all communication.

E.2 Chrome

With Google Chrome you generally need to set command-line parameters order to white list servers with Chrome will negotiate.

- on Windows machines (clients): Chrome shares the configuration with Internet Explorer so if all changes were applied to IE (as described in E.3), nothing has to be passed via command-line parameters.
- on Linux/Mac OS machines (clients): the command-line parameter `--auth-negotiate-delegate-whitelist` should only used if Kerberos delegation is required (otherwise do not set this parameter).
- It is recommended to use `https` for all communication.

```
--auth-server-whitelist="*.example.com"
--auth-negotiate-delegate-whitelist="*.example.com"
```

You can see which policies are enable by typing **chrome://policy/** into Chrome's address bar.

With Linux Chrome will also read policy files from `/etc/opt/chrome/policies/managed` directory.

mypolicy.json.

```
{
  "AuthServerWhitelist" : "*.example.org",
  "AuthNegotiateDelegateWhitelist" : "*.example.org",
  "DisableAuthNegotiateCnameLookup" : true,
  "EnableAuthNegotiatePort" : true
}
```

E.3 Internet Explorer

Complete following steps to ensure that your Internet Explorer browser is enabled to perform Spnego authentication.

- Open Internet Explorer.
- Click **Tools > Intenet Options > Security** tab.
- In **Local intranet** section make sure your server is trusted by i.e. adding it into a list.