Network And Security

**Abstract**

This document is for medium level or senior level developers, or any developer who wants to level up their knowledge. It is from theory to practicse.

Chapter 1 SSL, Security Certificate, HTTS

1.1 SSL

SSL (Secure Sockets Layer) is the standard security technology for establishing an encrypted link between a web server and a browser. This link ensures that all data passed between the web server and browsers remain private and integral. SSL is an industry standard and is used by millions of websites in the protection of their online transactions with their customers.To be able to create an SSL connection a web server requires an SSL Certificate.

1.2 SSL Certificate

Step 1: When you choose to activate SSL on your web server you will be prompted to complete a number of questions about the identity of your website and your company. Your web server then creates two cryptographic keys - a Private Key and a Public Key.

Step2: Then you should apply a SSL certificate from Certification Authority. The Public Key is placed into a Certificate Signing Request (CSR) - a data file also containing your details. You should then submit the CSR.

Typically an SSL Certificate will contain your domain name, your company name, your address, your city, your state and your country. It will also contain the expiration date of the Certificate and details of the Certification Authority responsible for the issuance of the Certificate. When a browser connects to a secure site it will retrieve the site's SSL Certificate and check that it has not expired, it has been issued by a Certification Authority the browser trusts, and that it is being used by the website for which it has been issued. If it fails on any one of these checks the browser will display a warning to the end user letting them know that the site is not secured by SSL.

Step3: When you use your certificate,browsers provide clients with a key indicator to let them know they are currently protected by an SSL encrypted session - the lock icon in the lower right-hand corner, clicking on the lock icon displays your SSL Certificate and the details about it. All SSL Certificates are issued to either companies or legally accountable individuals.

1.3 Wildcard certificate

There are lots of scams and bad intentions that now take place online. For people who have two or more sub-domains, they must strive to have wild card certificate. It is one of the cheapest and most famous secure certificates that are known for its advantages but one must have sufficient knowledge on how to handle it properly in order for you to reap all of its benefits.

There are different types of SSL or secure socket layer certificates available which you can choose from. The secure certificates types are company validated certificates, shared certificates, domain validated certificates, multi-domain certificates and wildcard certificates. All of these types are deliberately created for security purposes. However, they differ in terms of their functions. One of the best and the cheapest among those types is the wildcard certificate. It is well-known because of its great advantages when it comes to security aside from it less expensive price.

What is wildcard certificate? The wildcard SSL is another type of secure certificates that helps to enable the SSL encryption on your several sub domains with the use of one certificate, unless the domains are still in control by the similar organization and they are sharing same domain second-level name. Like, for example, a wildcard certificate released to SSL Company using the same name like “SSL.com” will be used to keep it secure with the following possible domains such as “login.ssl.com”, “support.ssl.com” or “payment.ssl.com” and more. A wildcard notification consists of asterisk then a period before the chosen domain name. These are wildcards to broaden the SSL encryption to its sub domains. In case of [www.ssl.com](http://www.ssl.com/) as example, the \*.ssl.com will also secure its other sub domains such as “login.ssl.com” and more.

1.4 A formal step to create SSL certificate

This is how SSL.com create SSL certificate for you.

1. You generate the csr (some tools)
2. You summit csr
3. Ssl.com validate your request by letting you reply an email or put some validation file in your domain.
4. You can your certificate from email.
5. You install your certificate on your server.

1.5 Types of SSL certificates

All [SSL certificates](https://www.ssl.com/certificates/) use similar methods to protect and validate your data, a useful way to categorize them is by validation method. Any certificate must be verified by the issuing Certificate Authority (or CA) to ensure that it is covering the correct, authorized site. This verification confirms at a minimum control of the domain. However, more steps can be taken to also confirm the existence of the requesting company or organization (for OV certificates) or to establish even more trust through extended vetting (for EV certificates).

DV: domain validated certificate

OV: Organization validated or OV certificates require more validation than DV certificates, but provide more trust.

EV: Extended validation or EV certificates provide the maximum amount of trust to visitors, and also require the most effort by the CA to validate.

1.6 SSL installation

SSL can be installed in IIS, Zimbra,Exchange, Microsoft Azure/CLOUD web app/web site.

SSL is a security protocol. Protocols describe how algorithms should be used. In this case, the SSL protocol determines variables of the encryption for both the link and the data being transmitted. Internet users have come to associate their online security with the lock icon that comes with an SSL-secured website or green address bar that comes with an extended validation SSL-secured website. SSL-secured websites also begin with https rather than http.

Here is a simple steps cited from godaddy to install the SSL in apache,

#### **To Install SSL and Intermediate Certificates**

1/ Copy your SSL certificate file and the certificate bundle file to your Apache server. You should already have a key file on the server from when you generated your certificate request.

2/ Locate the following directives in either your httpd.conf or ssl.conf file (which files you use depends on how you configured Apache). If one or more of them are currently commented out, uncomment them by removing the # character from the beginning of the line. Set the values of these directives to the absolute path and filename of the appropriate file, based on your version of Apache:

|  |  |
| --- | --- |
| **Apache Version < 2.4.8** | |
| **Directive** | **Path to Enter** |
| SSLCertificateFile | Certificate file path |
| SSLCertificateKeyFile | Key file path |
| SSLCertificateChainFile | Intermediate bundle path |
| **Apache Version 2.4.8+** | |
| **Directive** | **Path to Enter** |
| SSLCertificateFile | Certificate file path |
| SSLCertificateKeyFile | Key file path |
| SSLCACertificatePath | Intermediate bundle path |

3/ Save your configuration file and restart Apache.

4/ Restart your server. The procedure to restart Apache will depend heavily on your OS platform. On Unix-like platforms (Linux, Solaris, HP-UX, etc.) you will typically run a script to stop and restart the httpd daemon. On Windows, you will typically stop and restart the Apache service in the Services administrative console. Please consult your OS vendor's documentation or the [Apache documentation](http://httpd.apache.org/docs/).

Your SSL Certificate is installed. If you have problems, please see [Test your SSL's configuration](https://www.godaddy.com/help/test-your-ssls-configuration-6015) to help diagnose issues.

1.7 Is My Certificate SSL or TLS ?

The SSL protocol has always been used to encrypt and secure transmitted data. Each time a new and more secure version was released, only the version number was altered to reflect the change (e.g., SSLv2.0). However, when the time came to update from SSLv3.0, instead of calling the new version SSLv4.0, it was renamed TLSv1.0. We are currently on TLSv1.2.

1.8 How SSL works

1/ A browser or server attempts to connect to a Website, a.k.a. Web server, secured with SSL. The browser/server requests that the Web server identify itself.

2/ The Web server sends the browser/server a copy of its SSL certificate.

3/ The browser/server checks to see whether or not it trusts the SSL certificate. If so, it sends a message to the Web server.

4/ The Web server sends back a digitally signed acknowledgement to start an SSL encrypted session.

Encrypted data is shared between the browser/server and the Web server.

There are many benefits to using SSL Certificates. Namely, SSL customers:

* Get HTTPs which elicits a stronger Google ranking
* Create safer experiences for your customers
* Build customer trust and improve conversions
* Protect both customer and internal data
* Encrypt browser-to-server and server-to-server communication
* Increase security of your mobile and cloud apps

Apache's main configuration file is typically named httpd.conf or apache2.conf. Possible locations for this file include /etc/httpd/ or /etc/apache2/.

Set something like the following,

<VirtualHost 192.168.0.1:443>  
DocumentRoot /var/www/html2  
ServerName www.yourdomain.com  
SSLEngine on  
SSLCertificateFile /path/to/your\_domain\_name.crt  
SSLCertificateKeyFile /path/to/your\_private.key  
SSLCertificateChainFile /path/to/DigiCertCA.crt  
</VirtualHost>

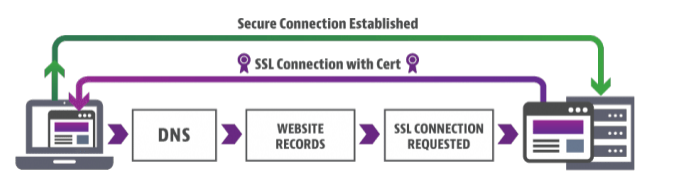
Check syntax error by

**apachectl configtest**

Restart apache,

**apachectl stop  
apachectl start**

Ref: http://www.apache.com/www/how-to-setup-an-ssl-certificate-on-apache/



1.9 What is CA

There are many CAs. Browsers like chrome or firefox will have a list of trusted CA and not trusted CA like whitelist and blacklist. Usually, client software—for example, browsers—include a set of trusted CA certificates. This makes sense, as many users need to trust their client software. A malicious or compromised client can skip any security check and still fool its users into believing otherwise. The clients of a CA are server supervisors who call for a certificate that their servers will bestow to users. Commercial CAs charge to issue certificates, and their customers anticipate the CA's certificate to be contained within the majority of web browsers, so that safe connections to the certified servers work efficiently out-of-the-box.

For example, wri got a CA and send to me and I installed in my web browser .A root CA certificate may be the base to issue multipleintermediate CA certificates with varying validation requirements.In addition to commercial CAs, some non-profits issue digital certificates to the public without charge; notable examples are [CAcert](https://en.wikipedia.org/wiki/CAcert" \o "CAcert) and [Let's Encrypt](https://en.wikipedia.org/wiki/Let's_Encrypt" \o "Let's Encrypt). There are commercial CAs, and non-profit CAs.Large organizations or government bodies may have their own PKIs ([public key infrastructure](https://en.wikipedia.org/wiki/Public_key_infrastructure" \o "Public key infrastructure)), each containing their own CAs. Any site using [self-signed certificates](https://en.wikipedia.org/wiki/Self-signed_certificate" \o "Self-signed certificate) acts as its own CA. So the default trusted certificate some are called CA certificates. Are further extended, so, for repeatedly visited servers, it is less error-prone importing and trusting the CA issued, rather than confirm a security exemption each time the server's certificate is renewed.

CA providers

Worldwide, the certificate authority business is fragmented, with national or regional providers dominating their home market. This is because many uses of digital certificates, such as for legally binding digital signatures, are linked to local law, regulations, and accreditation schemes for certificate authorities.

However, the market for globally trusted [TLS/SSL server certificates](https://en.wikipedia.org/wiki/Public_key_certificate" \l "Types_of_certificate" \o "Public key certificate) is largely held by a small number of multinational companies. This market has significant [barriers to entry](https://en.wikipedia.org/wiki/Barriers_to_entry" \o "Barriers to entry) due to the technical requirements.While not legally required, new providers may choose to undergo annual security audits (such as [WebTrust](https://en.wikipedia.org/wiki/WebTrust" \o "WebTrust)[[4]](https://en.wikipedia.org/wiki/Certificate_authority" \l "cite_note-4) for certificate authorities in North America and [ETSI](https://en.wikipedia.org/wiki/ETSI" \o "ETSI) in Europe[[5]](https://en.wikipedia.org/wiki/Certificate_authority" \l "cite_note-5)) to be included as a trusted root by a web browser or operating system. More than 180 root certificates are trusted in the [Mozilla Firefox](https://en.wikipedia.org/wiki/Mozilla_Firefox" \o "Mozilla Firefox) web browser, representing approximately eighty organizations.[[6]](https://en.wikipedia.org/wiki/Certificate_authority" \l "cite_note-6) [OS X](https://en.wikipedia.org/wiki/MacOS" \o "MacOS) trusts over 200 root certificates. On November 18, 2014, a group of companies and nonprofit organizations, including the [Electronic Frontier Foundation](https://en.wikipedia.org/wiki/Electronic_Frontier_Foundation" \o "Electronic Frontier Foundation), [Mozilla](https://en.wikipedia.org/wiki/Mozilla" \o "Mozilla), [Cisco](https://en.wikipedia.org/wiki/Cisco_Systems" \o "Cisco Systems), and [Akamai](https://en.wikipedia.org/wiki/Akamai_Technologies" \o "Akamai Technologies), announced [Let's Encrypt](https://en.wikipedia.org/wiki/Let's_Encrypt" \o "Let's Encrypt), a nonprofit certificate authority that provides free domain validated [X.509 certificates](https://en.wikipedia.org/wiki/X.509" \o "X.509) as well as software to enable installation and maintenance of certificates.[[7]](https://en.wikipedia.org/wiki/Certificate_authority" \l "cite_note-7) Let's Encrypt is operated by the newly formed [Internet Security Research Group](https://en.wikipedia.org/wiki/Internet_Security_Research_Group" \o "Internet Security Research Group), a California nonprofit recognized as tax-exempt under [Section 501(c)(3)](https://en.wikipedia.org/wiki/Section_501(c)(3)" \o "Section 501(c)(3)).[[8]](https://en.wikipedia.org/wiki/Certificate_authority" \l "cite_note-8)

Big providers are,

Comodo, Symentec, Verizon, GlobalSign, DigiCert, Entrust, ...etc.

2.0 Validation weakness

Domain validation is the most commonly used way to do and these can include attacks against the DNS, TCP, or BGP protocols (which lack the cryptographic protections of TLS/SSL), or the compromise of routers. Such attacks are possible either on the network near a CA, or near the victim domain itself.

One of the most common domain validation techniques involves sending an email containing an authentication token or link to an email address that is likely to be administratively responsible for the domain. (send email that is found in whois).This could be the technical contact email address listed in the domain's [WHOIS](https://en.wikipedia.org/wiki/Whois" \o "Whois) entry, or an administrative email like admin@, administrator@, webmaster@, hostmaster@ or [postmaster@](https://en.wikipedia.org/wiki/Postmaster_(computing)" \o "Postmaster (computing)) the domain.[[13]](https://en.wikipedia.org/wiki/Certificate_authority" \l "cite_note-13)[[14]](https://en.wikipedia.org/wiki/Certificate_authority" \l "cite_note-14) Some Certificate Authorities may accept confirmation using root@,[[citation needed](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed" \o "Wikipedia:Citation needed)] info@, or support@ in the domain.[[15]](https://en.wikipedia.org/wiki/Certificate_authority" \l "cite_note-15) The theory behind domain validation is that only the legitimate owner of a domain would be able to read emails sent to these administrative addresses.  In one instance, security researchers showed that attackers could obtain certificates for webmail sites because a CA was willing to use an email address like ssladmin@domain.com for domain.com, but not all webmail systems had reserved the "ssladmin" username to prevent attackers from registering it.[[16]](https://en.wikipedia.org/wiki/Certificate_authority" \l "cite_note-16) Prior to 2011, there was no standard list of email addresses that could be used for domain validation, so it was not clear to email administrators which addresses needed to be reserved. The first version of the [CA/Browser Forum](https://en.wikipedia.org/wiki/CA/Browser_Forum" \o "CA/Browser Forum)Baseline Requirements, adopted November 2011, specified a list of such addresses. This allowed mail hosts to reserve those addresses for administrative use, though such precautions are still not universal. In January 2015, a Finnish man registered the username "hostmaster" at the Finnish version of [Microsoft Live](https://en.wikipedia.org/wiki/Microsoft_Live" \o "Microsoft Live) and was able to obtain a domain-validated certificate for live.fi, despite not being the owner of the domain name.[[17]](https://en.wikipedia.org/wiki/Certificate_authority" \l "cite_note-17)

2.1 How to secure login session with SSL and Spring framework

If you use spring framework then you will need to learn spring security. If you use other framework, then you will need to use other framework’s security feature.

Instead of using Tomcat’s HttpSession, we are actually persisting the values in Redis. Spring Session replaces theHttpSession with an implementation that is backed by Redis. When Spring Security’sSecurityContextPersistenceFilter saves the SecurityContext to the HttpSession it is then persisted into Redis.

When a new HttpSession is created, Spring Session creates a cookie named SESSION in your browser that contains the id of your session. Go ahead and view the cookies (click for help with [Chrome](https://developer.chrome.com/devtools/docs/resources" \l "cookies) or [Firefox](https://getfirebug.com/wiki/index.php/Cookies_Panel" \l "Cookies_List)).

If you like, you can easily remove the session using redis-cli. For example, on a Linux based system you can type:

$ redis-cli keys '\*' | xargs redis-cli del

Alternatively, you can also delete the explicit key. Enter the following into your terminal ensuring to replace7e8383a4-082c-4ffe-a4bc-c40fd3363c5e with the value of your SESSION cookie:

$ redis-cli del spring:session:sessions:7e8383a4-082c-4ffe-a4bc-c40fd3363c5e

What is web application security?

Authorization, encryption, authentication?

Login form authentication does not provide encryption of content transmitted. It is done by TLS/SSL.

Spring Security is ultimately implemented in a web application as a javax.filter.Filter object configured in the servlet container.

The Spring Security filter should be the front-line of the web application, inspecting every request as it goes through. To automate the installation of all of these components, Spring Security supports a customWebApplicationInitializer subclass called**AbstractSecurityWebApplicationInitializer.**

package com.jl.crm.web;

import org.springframework.orm.jpa.support.OpenEntityManagerInViewFilter;

import org.springframework.security.web.context.AbstractSecurityWebApplication

  Initializer;

import org.springframework.security.web.session.HttpSessionEventPublisher;

import org.springframework.web.context.WebApplicationContext;

import org.springframework.web.filter.HiddenHttpMethodFilter;

import org.springframework.web.multipart.support.MultipartFilter;

import org.springframework.web.servlet.DispatcherServlet;

import org.springframework.web.servlet.support.AbstractDispatcherServletInitializer;

import javax.servlet.ServletContext;

public class CrmSecurityApplicationInitializer extends AbstractSecurity

             WebApplicationInitializer {

@Override

protected String getDispatcherWebApplicationContextSuffix() {

return AbstractDispatcherServletInitializer.DEFAULT\_SERVLET\_NAME;

}

@Override

protected void afterSpringSecurityFilterChain(ServletContext servletContext) {

insertFilters(servletContext,

   new HiddenHttpMethodFilter(),

   new MultipartFilter() ,

   new OpenEntityManagerInViewFilter());

}

@Override

protected boolean enableHttpSessionEventPublisher() {

return true;

}

}

Scaling web security with Spring Security Oauth

When a user authorizes the application and requested permissions, Facebook ultimately redirects the client to the requesting application and conveys in that request an accessToken. The accessToken is like a session cookie, and tells the server which client is connecting on behalf of the authenticated user. Additionally, Facebook has the ability to track which applications are installed in an application console (shown in Figure 3). The console is powerful: here, a user may centrally control (including revoke) access to any and all clients, selectively.

A username and password scheme certifies that a request is being made by a user. OAuth certifies that the request being made has the permission to do so from the user. That’s close enough to certifying the user for most applications that OAuth works with.

Integrating OAuth into our application is a snap thanks to Spring Security OAuth. Spring Security OAuth introduces the concept of a client – a logical notion composed of the authenticated user, a unique identifier, certain permissions (or scopes) that the client is permitted, and the type of OAuth connectivity supported by that client. OAuth supports varying levels of security, and can force the client to go through more hoops to increase the confidence in the security provided. We’re going to assume the common, but not exclusive, case of working with an Android mobile client. Spring Security OAuth needs information about which clients will connect.

The code below demonstrates the things added to the Spring Security Java configuration class – SecurityConfiguration – to support Spring Security OAuth. This configuration class extends from OAuth2ServerConfigurerAdapter:

@Configuration

@EnableWebSecurity

class SecurityConfiguration extends OAuth2ServerConfigurerAdapter {

private final String applicationName = ServiceConfiguration.CRM\_NAME;

@Inject private UserDetailsService userDetailsService;

@Inject private DataSource dataSource;

@Override

protected void registerAuthentication(AuthenticationManagerBuilder auth)

throws Exception {

auth.apply(new InMemoryClientDetailsServiceConfigurer())

.withClient("android-crm")

.resourceIds(applicationName)

.scopes("read", "write")

.authorities("ROLE\_USER")

.authorizedGrantTypes("authorization\_code",

"implicit", "password")

.secret("123456");

auth.userDetailsService(userDetailsService);

}

@Override

protected void configure(HttpSecurity http) throws Exception {

// ..

http.apply(new OAuth2ServerConfigurer())

.tokenStore(new JdbcTokenStore(this.dataSource))

.resourceId(applicationName);

// ..

}

@Bean

public PasswordEncoder passwordEncoder() {

return NoOpPasswordEncoder.getInstance();

}

@Bean

public TextEncryptor textEncryptor() {

return Encryptors.noOpText();

}

}

The first thing you’ll notice here is that the class extendsOAuth2ServerConfigurerAdapter instead of WebSecurityConfigurerAdapter. The two beans – passwordEncoder and textEncryptor – below are just no-op implementations of two APIs that Spring Security requires.

In Spring Security OAuth, the ClientDetailsService manages the information about which clients might connect, and how. To simplify setup, register anInMemoryClientDetailsServiceConfigurer in the registerAuthentication method. You could as easily plugin an implementation backed by a database.

When users run through the OAuth dance with our application, on the other hand, they’ll be presented with a page to confirm what scopes the user authorizes to the client. We must provide this page ourselves. Other contents here are skipped.

Ref: <https://www.safaribooksonline.com/blog/2013/10/08/secure-rest-services-with-spring-security/>

Chapter 2 Securing Your Tomcat App with SSL and Spring Security

1. Creating a key store

keytool -genkey -alias MyKeyAlias -keyalg RSA -keystore /Users/Roger/tmp/roger.keystore

* ****-alias**** is the unique identifier for your key.
* ****-keyalg**** is the algorithm used to generate the key. Most examples you find on the web usually cite 'RSA', but you could also use 'DSA' or 'DES'
* ****-keystore**** is an optional argument specifying the location of your key store file. If this argument is missing then the default location is your $HOME directory.

**RSA** stands for Ron Rivest (also the creator of the [RC4 algorithm](http://www.captaindebug.com/2011/08/rc4-encryption.html" \l ".UMNXcaUx_8s" \t "https://dzone.com/articles/new)), Adi Shamir and Leonard Adleman   
**DSA** stands for Digital Signature Algorithm   
**DES** stands for Data Encryption Standard

1. Update tomcat configuration

 To do this you need to find tomcat's server.xml configuration file, which is usually located in the 'conf' directory. Once you've got hold of this and if you're using tomcat, then it's a matter of uncommenting:   
…and making it look something like this:

<Connector SSLEnabled="true" keystoreFile="/Users/Roger/tmp/roger.keystore" keystorePass="password" port="8443" scheme="https" secure="true" sslProtocol="TLS"/>

If you're using Spring's tcServer, then you'll find that it already has a SSL connector that's configured something like this:

<Connector SSLEnabled="true" acceptCount="100" connectionTimeout="20000" executor="tomcatThreadPool" keyAlias="tcserver" keystoreFile="${catalina.base}/conf/tcserver.keystore" keystorePass="changeme" maxKeepAliveRequests="15" port="${bio-ssl.https.port}" protocol="org.apache.coyote.http11.Http11Protocol" redirectPort="${bio-ssl.https.port}" scheme="https" secure="true"/>

If you are using Spring Security, then there are a few more steps to getting things going. Part of the general Spring Security setup is to add the following to your web.xml file. Firstly you need to add a Spring Security application context file to the contextConfigLocation context-param:

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/spring/root-context.xml

/WEB-INF/spring/appServlet/application-security.xml

</param-value>

</context-param>

Secondly, you need to add the Spring Security filter and filter-mapping:

<filter>

<filter-name>springSecurityFilterChain</filter-name>

<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>

</filter>

<filter-mapping>

<filter-name>springSecurityFilterChain</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

Lastly, you need to create, or edit, your application-security.xml as shown in the very minimalistic example below:

<?xml version="1.0" encoding="UTF-8"?>

<beans:beans xmlns="http://www.springframework.org/schema/security"

xmlns:beans="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/security

http://www.springframework.org/schema/security/spring-security-3.1.xsd">

<http auto-config='true' >

<intercept-url pattern="/\*\*" requires-channel="https" />

</http>

<authentication-manager>

</authentication-manager>

</beans:beans>

In the example above intercept-url element has been set up intercept all URLs and force them to use the https channel.

Ref:

https://dzone.com/articles/securing-your-tomcat-app-ssl

Chapter 3 OSI and TCP IP