
Managing SSL Certificates for Linux Internet Server

Test Report of Kousec Server Certificate Manager

Part 2. Configuring SSL on Server

Copyright 2009,2010 Kousec Software, Inc. All rights reserved.
All company names and product names are trademarks of their respective holders.

Series Index

http://www.kousec.com/prod_cm.html

Part 1. Planning for Acquiring and Deploying Certificates

PDF: http://www.kousec.com/tj/tj_review_S1_en.pdf

Part 2. Configuring SSL on Server

PDF: http://www.kousec.com/tj/tj_review_S2_en.pdf

Part 3. Defining Certificate Specs and Obtaining Certificates

PDF: http://www.kousec.com/tj/tj_review_S3_en.pdf

Part 4. Deploying and Monitoring Certificates

PDF: http://www.kousec.com/tj/tj_review_S4_en.pdf

Part 5. Best Practices for Security and Operations

PDF: http://www.kousec.com/tj/tj_review_S5_en.pdf

Table of Contents

Configuring SSL on Server	5
Installing Linux OS and All Server Software Programs	6
Locations for Certificate-related Files on Server	6
Sharing Certificate-related Files among Server Software Programs.....	8
Configuring SSL Certificates for Server Programs.....	9
Generating Self-Signed Certificate	9
OpenLDAP (slapd)	10
Apache2 (httpd).....	12
FTP Server (vsftp)	13
SMTP Server (Postfix).....	14
POP/IMAP Server (Dovecot).....	16
Tomcat6.....	17
Testing Server Configurations	19

Configuring SSL on Server

In Part 1, we defined requirements for services to be installed on the server and associated SSL server certificates. In this part we start doing hands-on work to install and configure server software programs and to acquire and deploy server certificates.

The overall flow of activities for this evaluation is as follows.

1. **Plan for acquiring and deploying SSL certificates**
2. **Install Linux OS, install all server software programs**
3. **Configure each server software**

In this configuration, a self-signed certificate will be generated and it will be used to configure the server for SSL.
4. **On Kousec Server Certificate Manager, create certificate definitions**

Create certificate definitions based the requirements that we determined in Step 1.
5. **On Kousec Server Certificate Manager, start certificate acquisition process**

Obtain the certificate and save it in the certificate repository. In this evaluation we will use the built-in private CA to issue all certificates.
6. **On Kousec Server Certificate Manager, start certificate deployment process**

Deploy certificates stored in the certificate repository on to the servers. Specifically we will install the certificate on each service. In this step, we will replace certificate that were generated in Step 3 with certificates issued from the CA.
7. **On Kousec Server Certificate Manager, enabled monitoring of deployed certificate**

These steps will be described in multi-part series, as follows:

Part 2 : Step 2, Step 3

Part 3 : Step 4, Step 5

Part 4 : Step 6, Step 7

Part 5 : Best Practices in Operations, Conclusion

Also from now on, we will refer to Kousec Server Certificate Manager as just “CertMgr” or “Kousec CertMgr” for conciseness.

Installing Linux OS and All Server Software Programs

We will be using Ubuntu Server Edition 9.04 in this evaluation.

When installing the OS, we chose the following package “tasks”: LAMP server, Mail Server, OpenSSH server, Tomcat Java server. For the missing packages, we installed them using apt-get command, which is described in the Ubuntu Server Guide.

For the OS installation and initial SSL configuration for various server programs, we will follow Ubuntu Server Guide 9.04.

Ubuntu Server Guide 9.04 <https://help.ubuntu.com/9.04/serverguide/C/index.html>

Locations for Certificate-related Files on Server

Before we start configuring SSL for each server program, we need to decide where we will store certificate related files on the server.

On Ubuntu Server and CentOS 5.x, the standard location for storing SSL server certificates is separate from locations of the configuration files for each server program. On Ubuntu Server, certificate related files are placed in the following directories.

/etc/ssl/certs/ : certificates for the server and for CAs

/etc/ssl/private/ : private key for the server certificate

Similarly, on CentOS 5.x

/etc/pki/tls/certs/ : certificates for the server and for CAs

/etc/pki/tls/private/ : private key for the server certificate

The rationale behind the above placement policy as we imagine is that if you replace the single per-machine certificate, it will be picked up by all server programs automatically.

However we believe that it is better to keep the certificate related files together with other configuration files for each program. By keeping the coupling of the OS and each server

program loose, we facilitate migrating one server function to another server when the server expansion is needed.

If we followed the above practice from Ubuntu and CentOS, we would have the following problems.

- It complicates the permission settings on private key file(s)
The private key must be set readable only by the intended server program. Many server programs read the private key while still running as root and then drop the root privilege. However some server programs behave otherwise. For example, Openldap reads the private in the context of the normal openldap user. In that case, the private key file must be set readable not only to root but also to the openldap user. Tomcat 6 behaves similarly, i.e., reading the private key in the context of the tomcat normal user. Therefore the private key and its directory (“private”) must be set readable to both openldap and tomcat users.
Sharing the private key and its directory among multiple server programs is very likely to trigger operational errors¹. (The key can be read while it should not, and vice versa).
- When migrating one server function to another server, private keys for other server functions can be unintentionally backed up and transferred to another server. For example, when migrating the FTP service to another server, the private key for the LDAP service could also be copied and transferred to the new server.
- If a server program is compromised, private keys for other server programs are also exposed to the attacker.

From the above reasons, we will decide certificate location for each server program as follows: **store SSL certificate and its private key along with other configuration files for the server program, give the files permissions only to the root user, and optionally give permissions to the dedicated user of the server program only if necessary.**

For example, we will create ssl/certs and ssl/private directories in /etc/apache2 for an Apache SSL certificate, and /etc/ldap for an OpenLDAP SSL certificate.

Certificate file: */etc/<server-program-name>/ssl/certs/<certificate-name>.crt*

Private key file: */etc/<server-program-name>/ssl/private/<certificate-name>.key*

Certificate chain file: */etc/<server-program-name>/ssl/certs/<certificate-name>-chain.crt*

(if the certificate is signed by an intermediate CA)

Trusted CAs file: */etc/<server-program-name>/ssl/certs/ trusted-ca.crt*

¹ On Ubuntu Server, they create a group called “ssl-cert”, set the group of the “private” directory and private key file to “ssl-cert”, and add openldap and tomcat users to the “ssl-cert” group.

(In many cases, only when the server program supports client certificate authentication)

Note on Locations for Files: AppArmor is running by default on the Ubuntu Server. If an AppArmor profile for the server program is enabled, the server program can only read and write to the files and directories specified by the AppArmor profile. When determining files locations, you need to make sure the settings of AppArmor are appropriate. You can list enabled profiles with the command “apparmor status”. On Ubuntu Server 9.04, a profile is applied to slapd (OpenLDAP).

The following table summarizes locations of files, their ownership and access permission for each server program or service.

When you use CertMgr (explained later on), you do not need to enter these locations manually as CertMgr will extract necessary information from server program's configuration files.

Table 1 Locations of Certificate related files and security settings

Service	Location of private key Location of certificate	owner:group of private key	Permission on private key
OpenLDAP	/etc/ldap/ssl/private/ldap.example.com.key /etc/ldap/ssl/certs/ldap.example.com.crt	root:openldap	0640
Apache2	/etc/apache2/ssl/private/www.example.com.key /etc/apache2/ssl/certs/www.example.com.crt	root:root	0600
FTP Server	/etc/vsftpd/ssl/private/www2.example.com.key /etc/vsftpd/ssl/certs/www2.example.com.crt	root:root	0600
Postfix	/etc/postfix/ssl/private/smtp.example.com.key /etc/postfix/ssl/certs/smtp.example.com.crt	root:root	0600
Dovecot	/etc/dovecot/ssl/private/pop.example.com.key /etc/dovecot/ssl/certs/pop.example.com.crt	root:root	0600
Tomcat6	/etc/tomcat6/ssl/private/www2.example.com.jks	root:tomcat6	0640

Sharing Certificate-related Files among Server Software Programs

In Ubuntu Server by default, a single self-signed certificate is physically shared by multiple

server programs. This is probably done so as to reduce work for certificate renewal. However, this also means that you must stop services that are sharing this certificate when replacing it with a new one.

We will follow the principle that we try to separate services from the server machine. Therefore we will have a separate certificate file for each service. This is so even when two or more services use a single multi-domain certificate. With this rule, services are more independent from each other and it will be possible to renew the server certificates on per-service basis.

In this evaluation, we will use two multi-domain certificates, one for POP and IMAP and the other for FTP and WWW2 services. The POP and IMAP certificate is shared by a single server software, Dovecoat and therefore it is physically shared. The FTP and WWW2 certificate is used by separate server software, vsftpd and tomcat6 respectively. Therefore two copies of the certificate will be used. (It is also true tomcat6 and vsftpd can never physically share a certificate file because tomcat6 uses a certificate format called JKS, while vsftpd uses Openssl style certificate format).

Configuring SSL Certificates for Server Programs

Now we have decided how to place certificates on the server, we will next configure SSL for each server program. In Ubuntu Server 9.04, most configurations will be done when the software package is installed. However, since SSL and its certificate are not configured in many cases, we will configure each server software with SSL by following Ubuntu Server Guide, the official manual from Ubuntu. In this section we will list key points in configuring SSL and SSL certificates.

All commands documented are to be run by the user root. (By entering “sudo -s”, you will get a shell running as root.)

Generating Self-Signed Certificate

In the Ubuntu Server Guide, they describe a procedure in which you will generate a single self-signed certificate and have it shared by all server programs. On Ubuntu Server 9.04,

when you install at least one server program that can use SSL, a self-signed server certificate will be automatically generated in the following location.

```
/etc/ssl/certs/ssl-cert-snakeoil.pem      Server certificate
/etc/ssl/private/ssl-cert-snakeoil.key    Server private key
```

We will copy this self-signed certificate as the initial certificate when configuring SSL for each server program.

In case you need to re-create this self-signed certificate, you can follow the procedure [Creating a Self-Signed Certificate](#) in the Ubuntu Server Guide or execute the following commands (This is a more reduced set of commands).

```
> cd /tmp
> openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout server.key -out
server.crt
[Enter DN information.  Enter ldap.example.com for the Common Name.  For all others
you can just press Enter to accept default values.]
> cp server.crt /etc/ssl/certs/ssl-cert-snakeoil.pem
> cp server.key /etc/ssl/private/ssl-cert-snakeoil.key
```

OpenLDAP (slapd)

We will follow the chapter on OpenLDAP Server in the Ubuntu Server Guide for installing and configuring OpenLDAP. Since the command **apt-get install** will do the install and basic configuration, after finishing a basic testing we will configure the server program for SSL.

We will place the SSL certificate in a different location from where Ubuntu Server Guide recommends. Parameters for SSL and path to certificate files are as follows.

```
olcTLSCertificateFile /etc/ldap/ssl/certs/ldap.example.com.crt
olcTLSCertificateKeyFile /etc/ldap/ssl/private/ldap.example.com.key
olcTLSCACertificateFile /etc/ldap/ssl/certs/trusted-ca.crt
```

The following are key points to note

- Since the private key file is read by the **openldap** user, not the root user, the read permission on the private key must be given to the **openldap** user.
- In slapd, a trusted CA file must be specified even when SSL client certificate authentication is not used. When using a self-signed server certificate, that certificate will need to be stored in the file.

You can use the following commands to copy the self-signed certificate to the location for slapd use.

```
>mkdir -p /etc/ldap/ssl/certs
>mkdir /etc/ldap/ssl/private
>cd /etc/ldap/ssl
>chgrp openldap private ; chmod 0750 private
>cp /etc/ssl/certs/ssl-cert-snakeoil.pem certs/ldap.example.com.crt
>cp /etc/ssl/certs/ssl-cert-snakeoil.pem certs/trusted-ca.crt
>cp /etc/ssl/private/ssl-cert-snakeoil.key private/ldap.example.com.key
>chgrp openldap private/* ; chmod 0640 private/*
```

Then, follow the Ubuntu Server Guide to specify the paths to certificate-related files using `ldapmodify` command. The Ubuntu Server Guide shows a procedure that does not use an `ldif` file, but creating a `ldif` file will probably be easier and result in less mistyping. We list an example `ldif` file and the necessary command below.

Save the following content in a file named **certfile.ldif**.

```
dn: cn=config
changetype: modify
replace: olcTLSCertificateFile
olcTLSCertificateFile: /etc/ldap/ssl/certs/ldap.example.com.crt

dn: cn=config
changetype: modify
replace: olcTLSCertificateKeyFile
olcTLSCertificateKeyFile: /etc/ldap/ssl/private/ldap.example.com.key
```

```
dn: cn=config
changetype: modify
replace: olcTLSCACertificateFile
olcTLSCACertificateFile: /etc/ldap/ssl/certs/trusted-ca.crt
```

Command

```
> ldapmodify -x -D cn=admin,cn=config -W -f certfile.ldif
```

One thing to note is that if any one of the settings for SSL certificate-related files is wrong, the slapd daemon will not start. If the slapd daemon is not running, you cannot change the parameters using the `ldapmodify` command. If you fall in that situation, become the user **openldap**, open the file `/etc/ldap/slapd.d/cn=config.ldif` using **vi**, and delete the three lines including `olcTLSCertificateFile`. Then, you should be able to start the slapd daemon again.

Before you fall in that situation, it is better to confirm the integrity of the certificate related files using the following check.

How to check

```
> su -s /bin/bash openldap
[ Become openldap user to check file permission issues ]
> openssl verify -CAfile certs/trusted-ca.crt certs/ldap.example.com.crt
[ OK if "certs/ldap.example.com.crt: OK" is shown]
> openssl x509 -noout -modulus -in certs/ldap.example.com.crt | openssl md5
[ An MD5 hash (1) is shown]
> openssl rsa -noout -modulus -in private/ldap.example.com.key | openssl md5
[ If the MD5 hash shown here matches the MD5 hash of (1), then OK ]
```

Next, following the Ubuntu Server Guide, enable "ldaps" in `/etc/default/slapd` and restart the slapd daemon.

Apache2 (httpd)

Follow the chapter on Apache2 Web Server in the Ubuntu Server Guide to install and configure Apache2. In Ubuntu Server 9.04, when Apache2 is installed, the default SSL-enabled virtual server configuration file will be created automatically. It uses the

self-signed certificate located in /etc/ssl/certs.

We will rewrite the configuration file to use the certificate we have placed for solely for Apache2.

Cert-related files /etc/apache2/sites-available/default-ssl

The parameter values to set are as follows:

```
SSLCertificateFile    /etc/apache2/ssl/certs/www.example.com.crt
SSLCertificateKeyFile /etc/apache2/ssl/private/www.example.com.key
(SSLCertificateChainFile is not used when using a self-signed certificate)
```

You can use the following commands to copy the self-signed certificate to the location for apache2 use.

```
>mkdir -p /etc/apache2/ssl/certs
>mkdir    /etc/apache2/ssl/private
>cd /etc/apache2/ssl
>chgrp root private ; chmod 0700 private
>cp /etc/ssl/certs/ssl-cert-snakeoil.pem certs/www.example.com.crt
>cp /etc/ssl/certs/ssl-cert-snakeoil.pem certs/trusted-ca.crt
>cp /etc/ssl/private/ssl-cert-snakeoil.key private/www.example.com.key
>chgrp root private/* ; chmod 0700 private/*
```

Then, if not done yet, execute "a2enmod ssl ; a2ensite default-ssl" to enable SSL. Lastly restart Apache2 and check if it's working.

FTP Server (vsftpd)

Follow the chapter on vsftpd in the Ubuntu Server Guide to install and configure vsftpd. In Ubuntu Server 9.04, when vsftpd is installed, it will be configured to use the self-signed certificate located in /etc/ssl/certs.

We will rewrite the configuration file to use the certificate we have placed for solely for

vsftpd.

Cert-related files `/etc/vsftpd.conf`

The parameter values to set are as follows:

```
rsa_cert_file=/etc/vsftpd/ssl/certs/www2.example.com.crt
rsa_private_key_file=/etc/vsftpd/ssl/private/www2.example.com.key
```

In Ubuntu Server 9.04, vsftpd does not have a `/etc` directory dedicated to vsftpd. We will create such a directory, `/etc/vsftpd` to store the vsftpd SSL certificate.

You can use the following commands to copy the self-signed certificate to the location for vsftpd use.

```
>mkdir -p /etc/vsftpd/ssl/certs
>mkdir /etc/vsftpd/ssl/private
>cd /etc/vsftpd/ssl
>chgrp root private ; chmod 0700 private
>cp /etc/ssl/certs/ssl-cert-snakeoil.pem certs/www2.example.com.crt
>cp /etc/ssl/private/ssl-cert-snakeoil.key private/www2.example.com.key
>chgrp root private/* ; chmod 0700 private/*
```

Restart the vsftpd service and confirm that it is working.

SMTP Server (Postfix)

The SMTP server we are implementing needs three functionalities.

- A) Accepts outbound emails from internal PCs (User-authenticated SMTP Server)
- B) Sends outbound emails on to the Internet (SMTP Client)
- C) Receives inbound emails from the internet (Public SMTP Server)

Of these cases, A is the only case where an SSL server certificate is used very often. SSL

is used for server authentication and encryption when the SMTP server performs SMTP authentication. Here, we only describe SSL certificates regarding case A.

Specifications for the case A are as follows:

- Using SMTP authentication, authenticate users requesting to send email.
- Accepts requests on TCP/587 (SMTP submission port).
- Using TLS, encrypt communication and authenticate the server.

On Ubuntu Server 9.04, when you install Postfix, configuration files that use the default self-signed certificate will be created. Also if you follow the Ubuntu Server Guide and install the package **dovecot-postfix**, then the SMTP user authentication will be implemented by Dovecot's (POP/IMAP server) user authentication feature. (If you install **dovecot-postfix**, it will also install Dovecot itself). Also the submission port 587 is not enabled by default, so open the file `/etc/postfix/master.cf` and enable the line that contains "submission" word.

Now, let's start configuring the settings for an SSL server certificate. We will edit the config file to use the certificate dedicated for Postfix.

```
Certificate related config file /etc/postfix/main.cf
```

Required parameters are as follows:

```
smtpd_tls_cert_file=/etc/postfix/ssl/certs/smtp.example.com.crt  
smtpd_tls_key_file=/etc/postfix/ssl/private/smtp.example.com.key
```

Also comment out the following parameter in `main.cf` when you are editing the file. (This seems to be a bug in `dovecot-postfix` package in Ubuntu 9.04).

```
smtpd_tls_mandatory_ciphers = medium, high
```

You can use the following commands to copy the self-signed certificate to the location for Postfix.

```
>mkdir -p /etc/postfix/ssl/certs
>mkdir /etc/postfix/ssl/private
>cd /etc/postfix/ssl
>chgrp root private ; chmod 0700 private
>cp /etc/ssl/certs/ssl-cert-snakeoil.pem certs/smtp.example.com.crt
>cp /etc/ssl/private/ssl-cert-snakeoil.key private/smtp.example.com.key
>chgrp root private/* ; chmod 0600 private/*
```

We will not set up an SSL client certificate that would be used by Postfix when acting as an SMTP client. The primary objective for using SSL in SMTP is to encrypt and authenticate network communication when performing SMTP authentication (i.e., when acting as an SMTP server).

Lastly, restart the Postfix service and confirm that it is working.

POP/IMAP Server (Dovecot)

When you install Dovecot on in Ubuntu Server 9.04, a configuration file with the default self-signed certificate will be created. The Ubuntu Server Guide recommends installing **dovecot-postfix** package to enable SMTP-AUTH when installing Postfix. Installing that package triggers installing the main Dovecot package, so Dovecot should be already installed.

Another thing that is NOT documented in the Ubuntu Server Guide. When you install **dovecot-postfix** package, the main config file for Dovecot will be `/etc/dovecot/dovecot-postfix.conf`, not `/etc/dovecot/dovecot.conf`. So if you install Postfix along with Dovecot and want to do set up Dovecot by following the Ubuntu Server Guide, you must read `dovecot.conf` as `dovecot-postfix.conf`.

Now, let's start configuring the settings for an SSL server certificate. We will edit the config file to use the certificate dedicated for Dovecot.

Certificate related config file `/etc/dovecot/dovecot-postfix.conf`
(`/etc/dovecot/dovecot.conf`, if `dovecot-postfix` package is not installed)

The parameter values to set are as follows:

```
ssl_cert_file = /etc/dovecot/ssl/certs/pop.example.com.crt
ssl_key_file = /etc/dovecot/ssl/private/pop.example.com.key
```

Use the following commands to copy the self-signed certificate to the location for Dovecot use.

```
>mkdir -p /etc/dovecot/ssl/certs
>mkdir    /etc/dovecot/ssl/private
>cd /etc/dovecot/ssl
>chgrp root private ; chmod 0700 private
>cp /etc/ssl/certs/ssl-cert-snakeoil.pem certs/pop.example.com.crt
>cp /etc/ssl/private/ssl-cert-snakeoil.key private/pop.example.com.key
>chgrp root private/* ; chmod 0600 private/*
```

Lastly, restart the Dovecot service and confirm that it is working.

Tomcat6

The last server program is Tomcat 6.

On the Ubuntu Server, there are two methods of installing Tomcat. The first method is to install Tomcat on the system in the same way as other services (System-wide installation). The other method is to install the tomcat as the instance private to the developer (Using private instances).

This time, as we will be using Tomcat as a production service, we will use the System-wide installation. We will install Tomcat following the Ubuntu Server Guide.

The procedure to set up an SSL certificate for Tomcat is not documented in the Ubuntu

Server Guide, but you can easily set one up by following the Apache Tomcat manual (<http://tomcat.apache.org/tomcat-6.0-doc/ssl-howto.html>). For your information, we will describe the procedure in this document also.

Unlike other Unix-based server programs, Tomcat uses a single file-based storage called Java Key Store (JKS) to store certificates and private keys. We will place our JKS file in the following location.

```
JKS File /etc/tomcat6/ssl/private/www2.example.com.jks
```

The tomcat daemon starts as root and downgrades to the user tomcat6 privilege to read this JKS file. Therefore we need to set the permissions of this file and its containing private directory to the following.

```
owner:group root:tomcat6 permission: 0640
```

Note: Since Tomcat 6.0, it is possible to use OpenSSL as the SSL engine. We will not use this, so be careful when reading the Tomcat manual.
JSEE-based SSL Connector: we will use this one.
APR/OpenSSL-based SSL Connector: we will NOT use this one.

In the initial SSL set up, we will place a self-signed certificate in this JKS file. Then, using CertMgr, we will replace it with the formal certificate later on.

Generate a self-signed certificate using the following commands

```
>mkdir -p /etc/tomcat6/ssl/private  
>keytool -genkey -alias tomcat -keyalg RSA -keystore  
/etc/tomcat6/ssl/private/www2.example.com.jks
```

As for passwords, two passwords (keystore password and key password) will be requested. In Tomcat, these two passwords must match, so enter the same password. The default password in Tomcat is “changeit”, so we will use that.

As for information about DN, we will modify them on CertMgr when we generate or obtain an

official certificate. So we can just accept default values.

Lastly, set permissions on the JKS files:

```
>cd /etc/tomcat6/ssl
>chgrp tomcat6 private ; chmod 0750 private
>chgrp tomcat6 private/* ; chmod 0640 private/*
```

1. Edit the SSL connector setting in /etc/tomcat6/server.xml.
 - A) The SSL Connector section is commented out. Enable it.
 - B) Add the file path to the JKS file in KeystoreFile parameter.
 - C) Add the password in keystorePass.

```
<!-- Define a SSL HTTP/1.1 Connector on port 8443
      This connector uses the JSSE configuration, when using APR, the
      connector should be using the OpenSSL style configuration
      described in the APR documentation -->
<!--
<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
           maxThreads="150" scheme="https" secure="true"
           clientAuth="false" sslProtocol="TLS"
           keystoreFile="/etc/tomcat6/ssl/private/www2.example.com.jks"
           keystorePass="changeit" />
-->
```

Lastly, restart the tomcat6 service and confirm that it is working.

Testing Server Configurations

We will now list up the TCP port numbers and their protocols through which the server programs will be providing their services. Configure the firewall(s) according to this table.

Table 2 List of Port Numbers for Services Configured

Port Number	Server Program	Protocol	Over SSL / STARTTLS	
389	OpenLDAP (slapd)	LDAP	STARTTLS	

636	OpenLDAP (slapd)	LDAP	over SSL	
80	Apache2 (httpd)	HTTP	None	
443	Apache2 (httpd)	HTTP	over SSL	
21	Vsftpd	FTP	STARTTLS	
25	Postfix	SMTP	STARTTLS	
587	Postfix	SMTP	STARTTLS	
110	Dovecot	POP3	STARTTLS	
995	Dovecot	POP3	over SSL	
143	Dovecot	IMAP	STARTTLS	
993	Dovecot	IMAP	over SSL	
8443	Tomcat6	HTTP	over SSL	

The column Over SSL/STARTTLS indicates how SSL is implemented. **over SSL** is a method to encapsulate an application protocol in SSL. Typically, SSL/TLS communication will begin immediately after an TCP connection has been established. In this method, the server program needs one TCP port dedicated for SSL.

In the **STARTTLS** method, the client connects to the standard (non-SSL-enabled) port and initiates a TLS communication by using one of application-protocol commands. It is important to be fully aware of these two methods when creating user documentation on configuring client applications and trouble-shooting.

For reference, we will also list the port numbers that are not configured in this evaluation.

Port Number	Server Program	Protocol	Over SSL / STARTTLS	
465	(Postfix)	SMTP	over SSL	
990		FTP	over SSL	

The port 465 is used for smtps (SMTP over SSL) and is supported on many email clients. Postfix also supports it and in Ubuntu Server, you can enable it by enabling “smtps” line in /etc/postfix/master.cf. It is recommended to enable this port because Postfix will support more clients and it will also make client PC configuration easier.

The port 990 is used for ftps (FTP over SSL – Implicit). Even though the vsftpd included in the Ubuntu Server 9.04 does not support this mode of ftps, it is recommended that you will be aware of this port because the term often appears in documentation of most SSL-enabled FTP clients.

About Trouble-Shooting

- How to check

Even if a service starts up successfully after configuring a SSL certificate, that does not mean the SSL certificate is configured correctly. We will perform step-by-step check as follows.

- Check if the server process listens on its TCP port.

```
netstat -untap
```

- From the local machine, use “openssl” command to the port.

Example `openssl s_client -connect host:port -showcerts [-starttls <smtp|pop3|imap|ftp>]`

If the command errors or hangs, check syslog file and logs of the server program (`/var/log/*`).

- Next, connect to the TCP port from another machine using “telnet <ip-address> <port-number>”.

If you get “Connection refused” or “Connection timedout”, then review settings of Firewall of the server.

- Common Errors

Cannot open the secret key: Review file permission of the secret key. Review AppArmor profiles.

The secret key is encrypted: Use “openssl” to unencrypt the key and place it.

The certificate and secret key do not match.

After confirming all of the above checks, we can try connecting from a real client or CertMgr. For issues that could happen in the upper layers (SSL/PKI, etc) in those tests, we will perform trouble-shooting using CertMgr.

Here is the summary on what we have done in this article.

- We installed and configured each server program by following the Ubuntu Server Guide
- We enabled SSL in each server program by with the automatically generated self-signed certificate.
- As for the location of the SSL certificate and key, we created a directory for each server program, copied the self-signed certificate there and changed the configuration file to point to the new location.

By locating SSL certificates for server programs this way, we can manage SSL certificates for each service in the state that is isolated and independent from other services and the server machine itself. This will allow for improved maintainability and extensibility of the server in future.

That is all for configuring SSL on the server.

From the next part article and on, we will use Server Certificate Manager to actually obtain and install new SSL certificates for production.