目录

[一、 实验要求 2](#_Toc492740242)

[二、 实验方法 2](#_Toc492740243)

[三、 实验原理 2](#_Toc492740244)

[1. 客户端发出握手请求(Client Hello)，包含以下信息： 2](#_Toc492740245)

[2. 服务器回复(Server Hello)，包含以下信息： 3](#_Toc492740246)

[3. 客户端回应，包含以下步骤： 3](#_Toc492740247)

[4. 服务器回应 3](#_Toc492740248)

[HTTPS协议如何防止中间人攻击？ 4](#_Toc492740249)

[四、 实验环境 4](#_Toc492740250)

[硬件环境 4](#_Toc492740251)

[软件环境 4](#_Toc492740252)

[加密/解密算法版本 5](#_Toc492740253)

[五、 实验过程描述 5](#_Toc492740254)

[项目整体结构 5](#_Toc492740255)

[构建CA为服务器签发证书[4] 6](#_Toc492740256)

[为Tomcat服务器生成证书签发请求 8](#_Toc492740257)

[使用CA的证书为Tomcat服务器的签发证书 9](#_Toc492740258)

[生成PKCS12格式的keystore文件供tomcat服务器使用 9](#_Toc492740259)

[配置Tomcat服务器使用HTTPS协议 10](#_Toc492740260)

[配置web应用使用HTTPS协议 10](#_Toc492740261)

[SUCCESS 10](#_Toc492740262)

[注意事项 11](#_Toc492740263)

[客户端凭证书免密码登录 11](#_Toc492740264)

[配置防火墙 11](#_Toc492740265)

[生成 CA 的 truststore，导入到 tomcat 服务器中 11](#_Toc492740266)

[给客户端签发证书并将此证书导入浏览器中 13](#_Toc492740267)

[修改 web 应用的认证机制 13](#_Toc492740268)

[SUCCESS 13](#_Toc492740269)

[六、 实验结果与分析 14](#_Toc492740270)

[没有使用HTTPS之前的数据传输状况 14](#_Toc492740271)

[使用HTTPS之后的数据传输状况 15](#_Toc492740272)

[应用安全性分析 15](#_Toc492740273)

[数据库系统安全性分析 16](#_Toc492740274)

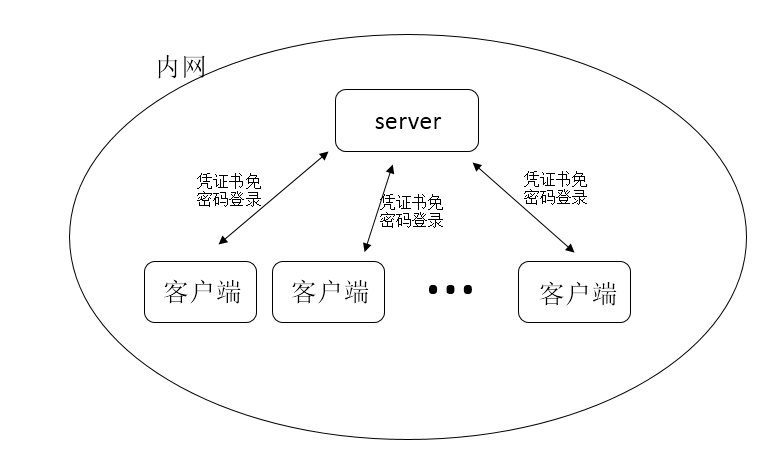
[操作系统安全性分析 16](#_Toc492740275)

[七、 参考 16](#_Toc492740276)

# 实验要求

1. 在队友的访问控制实现基础上增加HTTPS协议，实现信息的安全传输.

2. 内网用户使用证书免密码登录服务器，网络拓扑如下图：



3. 分析本web应用的数据库安全性以及系统安全性

# 实验方法

在没有使用HTTPS实现加密传输之前，从服务器到客户端以及从客户端到服务器的内容都是明文传输，因此重要信息（e.g., 帐号密码）容易被中间人截获．本实验在实现访问控制功能的前提下增加HTTPS协议，使得所有数据都能实现加密传输．并且在此基础上，使得内网用户可以使用证书免密码登录服务器

# 实验原理

HTTPS主要在原生HTTP协议的基础上增加TLS/SSL层，一开始的握手过程中使用非对称加密算法建立连接并交换相关的信息（比如TLS/SSL协议的版本和后续使用到的对称加密算法），建立连接后使用对称加密算法交换数据．其步骤如下：

## 客户端发出握手请求(Client Hello)，包含以下信息：

1） 支持的协议版本，比如TLS 1.0版。

2） 一个客户端生成的随机数(random\_1)，这个随机数既需要客户端保存又需要发送给服务器。

3）支持的加密方法，比如RSA公钥加密。

4）支持的压缩方法。

## 服务器回复(Server Hello)，包含以下信息：

1）确认使用的加密通信协议版本，比如TLS 1.0版本。如果浏览器与服务器支持的版本不一致，服务器关闭加密通信。

2）一个服务器生成的随机数(random\_2)。

3）确认使用的加密方法，比如RSA公钥加密。

4）服务器证书。

5）如果服务器需要确认客户端的身份，就会再包含一项请求，要求客户端提供”客户端证书”。比如，金融机构往往只允许认证客户连入自己的网络，就会向正式客户提供USB密钥，里面就包含了一张客户端证书。

## 客户端回应，包含以下步骤：

1）验证服务器证书的合法性，证书合法性包括：证书是否过期，发行服务器证书的 CA 是否可靠，发行者证书的公钥能否正确解开服务器证书的“发行者的数字签名”，服务器证书上的域名是否和服务器的实际域名相匹配。如果合法性验证没有通过，通讯将断开；

2）客户端使用一些加密算法(例如：RSA,Diffie-Hellman)产生一个48个字节的Key，这个Key叫PreMaster Secret。该PreMaster Secret用服务器公钥加密传送，防止被窃听。

3）编码改变通知，表示随后的信息都将用双方商定的加密方法和密钥发送。

4）客户端握手结束通知，表示客户端的握手阶段已经结束。这一项同时也是前面发送的所有内容的hash值，用来供服务器校验。

5）如果前一步，服务器要求客户端证书，客户端会在这一步发送证书及相关信息。

## 服务器回应

服务器通过上面的三个随机数(random\_1,random\_2,PreMaster Secret)，计算出本次会话的『会话密钥(session secret)』， 然后向客户端发送下面信息

1） 编码改变通知，表示随后的信息都将用双方商定的加密方法和密钥发送。

2）服务器握手结束通知，表示服务器的握手阶段已经结束。这一项同时也是前面发送的所有内容的hash值，用来供客户端校验。

至此，服务器和客户端的握手阶段全部结束，接下来，客户端与服务器进入加密通信，就完全是使用普通的HTTP协议，只不过用『会话密钥(session secret)』对内容做对称加密。

## HTTPS协议如何防止中间人攻击？

换句话说，如何保证

1）完整性

2）不可抵赖性．

有两种中间人攻击的形式

1）中间人假冒服务器给客户端发消息

2）中间人篡改握手过程中的数据，比如第二个随机数(服务器发给客户端的)。

对于第1个问题，在 2.2 中服务器向客户端发送证书的时候就可以知道了，如果中间人使用了其他证书，这个证书几乎是不可被信任的(浏览器会立刻警告用户)。对于第2个问题，在2.3客户端回应的hash值可以供服务器验证, 在2.4服务器向客户端发送的前面所发送的所有内容的hash 值其实是一个签名(这个hash值是用服务器的私钥来签名了的)。另外，服务器发向客户端的证书上有两个签名: CA的签名用以验证这个证书(的公钥)确实是某某某人的;服务器的签名，用以验证这个证书确实是由服务器签发的(完整性、不可抵赖性)。

# 实验环境

## 硬件环境

1. 内存: 8.00 GB

2. Intel® Core™ i5 – 4210M CPU(@2.60 GHz)

## 软件环境

1. 操作系统: Ubuntu 16.04

2. JDK: 8

3. Spring版本: 4.02.RELEASE

4. OpenSSL版本: OpenSSL 1.0.2g 1 Mar 2016

## 加密/解密算法版本

1. 签名算法: sha256WithRSAEncryption

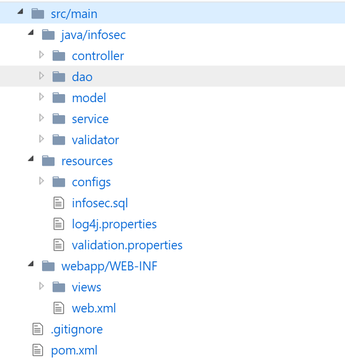
2. 公钥算法: rsa

3. 公钥长度: 2048bit

# 实验过程描述

## 项目整体结构

项目整体结构如下所示（与队友项目结构一致）:



在这个实验中只需要修改web.xml文件，给infosec web应用增加安全传输项，以及给修改tomcat服务器的多项配置使其使用HTTPS功能，并管理证书．然后再配置防火墙只允许来自内网的网络访问某个端口，然后给客户端分发证书，用户即可凭借用户证书免密码登录。下面的实验中，我们首先展示如何给tomcat服务器加上 HTTPS，然后展示如何让客户端使用证书免密码登录。

## 构建CA为服务器签发证书[4]

CA的构建和证书的签发使用openssl，CA的证书会被导入到自己的浏览器中，最终由CA签发的证书被集成到一个PKCS12格式的文件中供tomcat使用．参考[3]．

*#*

*# Filename: openssl.cnf*

*#*

*# Establish working directory*

dir = .

[ ca ]

default\_ca = CA\_default

[ CA\_default ]

serial = $dir/serial

database = $dir/index.txt

new\_certs\_dir = $dir/newcerts

certificate = $dir/cacert.pem

private\_key = $dir/private/cakey.pem

default\_days = 3650

default\_md = sha256

preserve = no

email\_in\_dn = no

nameopt = default\_ca

certopt = default\_ca

policy = policy\_match

# these are policies that are enforced by the openssl tool

[ policy\_match ]

#e.g., if set to "match", then the `stateOrProvinceName' field needed to be the

#same in the CA certificate and the request

# countryName = match

countryName = optional

stateOrProvinceName = optional

organizationName = optional

organizationalUnitName = optional

commonName = supplied

emailAddress = optional

[ req ]

default\_bits = 2048 *#size of keys*

default\_keyfile = key.pem *#name of generated keys*

default\_md = sha256 *#message digest algorithm*

string\_mask = nombstr *#permitted characters*

distinguished\_name = HP *#the owner of this cert*

[ HP ]

*# Variable name Prompt string*

*#---------------------- ----------------------------------*

0.organizationName = Organization Name (company)

organizationalUnitName = Organizational Unit Name (department, division)

emailAddress = Email Address

emailAddress\_max = 40

localityName = Locality Name (city, district)

stateOrProvinceName = State or Province Name (full name)

countryName = Country Name (2 letter code)

countryName\_min = 2

countryName\_max = 2

commonName = Common Name (hostname, IP, or your name)

commonName\_max = 64

# Default values for the above, for consistency and less typing.

# Variable name Value

#------------------------------ ------------------------------

0.organizationName\_default = walkerlalaHP

organizationalUnitName\_default = walkerlalaHPDept0

emailAddress\_default = no-reply@gmail.com

localityName\_default = Guangzhou

stateOrProvinceName\_default = Guangdong

countryName\_default = CN

commonName\_default = 127.0.0.1

[ v3\_ca ]

basicConstraints = CA:TRUE

subjectKeyIdentifier = hash

authorityKeyIdentifier = keyid:always,issuer:always

然后在当前目录是使用以下命令生成CA的私钥和证书:

openssl req -new -x509 -extensions v3\_ca -keyout private/cakey.pem **\**

-out cacert.pem -days 3650 -config ./openssl.cnf

上面的命令中，

-new -x509: 用来构建一个自签发的证书

-extensions v3\_ca: 意思是使用 配置文件中的`v3\_ca`部分的配置来构建证书

-days 3650: 意思是这个证书的有效日期是 3650天

-key: 指定CA的私钥的存放位置

-out: 指定CA的证书的存放位置

-config: 指定输入的配置文件的位置

将的得到的CA的证书cacert.pem导入到浏览器中，使得后续由CA签发的证书可以被认可。

## 为Tomcat服务器生成证书签发请求

首先为Tomcat服务器生成证书及其私钥．其配置文件如下:

*#*

*# Filename: openssl-nonca.cnf*

*#*

*# Establish working directory*

dir = .

[ req ]

default\_bits = 2048 *#size of keys*

default\_keyfile = key.pem *#name of generated keys*

default\_md = sha256 *#message digest algorithm*

string\_mask = nombstr *#permitted characters*

distinguished\_name = HP *#the owner of this cert*

req\_extensions = v3\_req

[ HP ]

# Variable name Prompt string

#---------------------- ----------------------------------

0.organizationName = Organization Name (company)

organizationalUnitName = Organizational Unit Name (department, division)

emailAddress = Email Address

emailAddress\_max = 40

localityName = Locality Name (city, district)

stateOrProvinceName = State or Province Name (full name)

countryName = Country Name (2 letter code)

countryName\_min = 2

countryName\_max = 2

commonName = Common Name (hostname, IP, or your name)

commonName\_max = 64

# Default values for the above, for consistency and less typing.

# Variable name Value

#------------------------------ ------------------------------

0.organizationName\_default = Tomcat

organizationalUnitName\_default = TomcatDept0

emailAddress\_default = no-reply@gmail.com

localityName\_default = Guangzhou

stateOrProvinceName\_default = Guangdong

countryName\_default = CN

commonName\_default = 127.0.0.1

[ v3\_req ]

basicConstraints = CA:FALSE

subjectKeyIdentifier = hash

在当前目录下使用以下命令来为tomcat服务器生成证书和私钥:

openssl req -new -nodes -out tomcat-req.pem -config ./openssl-nonca.cnf

在当前目录下会生成证书签发请求文件 tomcat-req.pem以及相对应的私钥 key.pem．

## 使用CA的证书为Tomcat服务器的签发证书

使用以下命令为tomcat服务器签发证书

openssl ca -out tomcat-cert.pem -config ./openssl.cnf -infiles tomcat-req.pem

其中由CA签发的证书为 tomcat-cert.pem

## 生成PKCS12格式的keystore文件供tomcat服务器使用

很多人使用 JKS 格式的文件来配置tomcat服务器，因为 JKS 格式是JAVA的"standard java keystore"格式．但是由于PKCS12格式更加通用，而且在 Windows 平台上也可以使用，所以这里我们选择PKCS12格式．依然使用openssl工具来生成pkcs12格式的keystore文件:

openssl pkcs12 -export -in tomcat-cert.crt -inkey key.pem **\**

-out tomcat-pkc12-cert.p12 -name tomcat -CAfile cacert.pem **\**

-caname root -chain

这里的命令的意思是:

-in tomcat-cert.crt: 指定了已经被CA签发的tomcat服务器的证书

-inkey key.pem: 指定了与tomcat服务器的证书所对应的私钥

-out tomcat-pkcs12-cert.p12: 指定了生成的 PKCS12 格式的 keystore 文件的存放位置

-name tomcat: 指定这个PKCS12格式的keystore文件的别名-CAfile cacert.pem` 为CA的证书

(注意这里的tomcat-cert.crt文件与上一个步骤中的tomcat-cert.pem的差别只是少了几句可读性描述，打开文件就能看到．参考[3])

## 配置Tomcat服务器使用HTTPS协议

配置tomcat的配置文件 server.xml，在server配置项中增加如下信息：

**<Connector** SSLEnabled="true" acceptCount="1000" clientAuth="false"

disableUploadTimeout="true" enableLookups="false" maxThreads="25"

port="443" keystoreFile="/path/to/keystore/file" keystorePass="yourpasswd"

protocol="org.apache.coyote.http11.Http11NioProtocol" scheme="https"

secure="true" sslProtocol="SSL" **/>**

## 配置web应用使用HTTPS协议

在web应用的\<web-app>一项中增加如下信息：

**<security-constraint>**

**<web-resource-collection>**

**<web-resource-name>**securedapp**</web-resource-name>**

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

**</web-resource-collection>**

**<user-data-constraint>**

**<transport-guarantee>**CONFIDENTIAL**</transport-guarantee>**

**</user-data-constraint>**

**</security-constraint>**

## SUCCESS

最终成功为infosec web应用加上HTTPS应用：

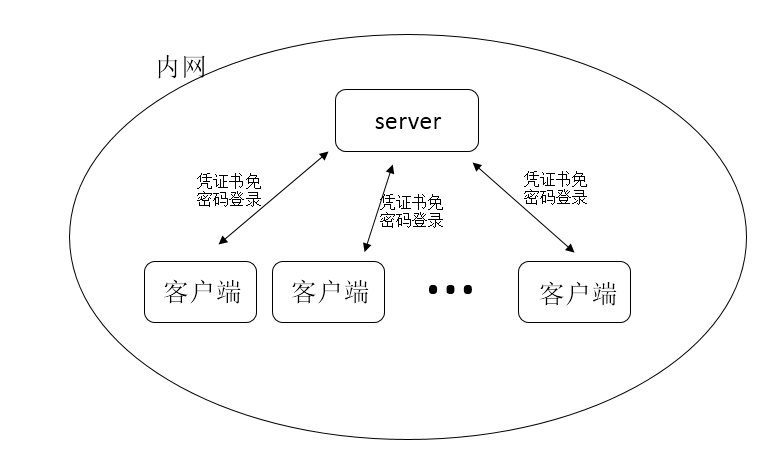


## 注意事项

由于443是小于1024的端口，只能root用户才能使用，所以要更改tomcat的启动用户为root

## 客户端凭证书免密码登录

下面展示如何配置服务器，以及更改 web 应用的代码，使得客户端可以凭证书免密码登录。网络拓扑如下图：

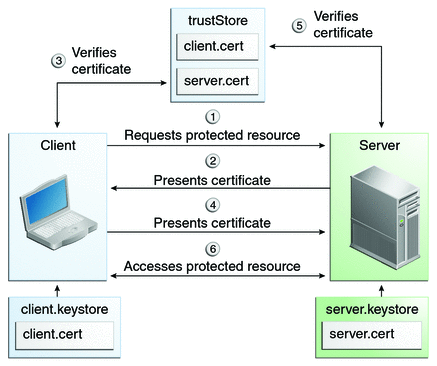


## 配置防火墙

我们首先配置防火墙，使得某个端口只允许内网访问。为了展示容易，我们这里跳过防火墙的配置。

## 生成 CA 的 truststore，导入到 tomcat 服务器中

为了使得 tomcat 服务器能够认可客户端发来的、由同一个 CA 签署的证书，我们必须要将CA的信息导入到 tomcat 服务器中：



（图片来源: <http://l-lin.github.io/2014/09/09/Auth_with_certificates_Tomcat_spring>）

具体操作如下：

$ keytool -import -alias myauthority -keystore truststore.jks \

-file cacert.pem

然后更改 tomcat 服务器的 server.xml，导入生产的 truststore，并且，为了使得内网用户可以凭借证书登录，我们要调节下面的 clientAuth 选项，打开 HTTPS 双向认证：

**<Connector** SSLEnabled="true" acceptCount="1000"

clientAuth="true"

disableUploadTimeout="true"

enableLookups="false"

maxThreads="25"

port="443"

keystoreFile="/path/to/keystore/file"

keystorePass="yourpasswd"

truststoreFile="/path/to/truststore.jks"

truststoreType="JKS"

truststorePass="yourpass2"

protocol="org.apache.coyote.http11.Http11NioProtocol" scheme="https"

secure="true"

sslProtocol="SSL" **/>**

（note: 如果不强制使用证书登录，可以设置 clientAuth=”want”）

## 给客户端签发证书并将此证书导入浏览器中

给客户端签发证书的流程和给tomcat服务器签发证书的流程一样，而且使用同一个 CA。签完之后，将证书导入浏览器中的“个人证书“中（火狐浏览器需要pks12格式的证书，因为它同时需要证书以及证书的私钥）。

## 修改 web 应用的认证机制

在负责 login 的 controller 里面，在最前面插入如下代码，查看证书是否正确，如果正确则使其成功登录：

X509ertificate certs[] = (X509ertificate[])

request.getAttribute("javax.servlet.request.X509Certificate");

String issuer = **null**;

**if**(certs != **null** && certs.length > 0) {

issuer = certs[0].getIssuerX500Principal().getName();

**if**(ISSUER\_MATCH(issuer)) {

model.attribute("msg", "login sucessfully, your issuer is:" + THE\_ISSUER);

**return** "index";

}

}

(上面的 ISSUER\_MATCH() 和 THE\_ISSUER 都是自定义的)

然后修改一下认证的逻辑使得其成功登录即可。

(note: 这里我们不需要检查证书是否正确，因为如果证书不正确，这个连接是不会成功的，即，tomcat 服务器帮我们处理的证书 认证问题。因此，到了 login controller 那里之后，我们即可知道这个客户端的证书是同一个CA 签发的)

## SUCCESS

以上步骤完成之后，客户端浏览器即可凭借服务器免密码登录（下面认证成功，告知用户他的证书的签发者(issuer)）：



# 实验结果与分析

## 没有使用HTTPS之前的数据传输状况

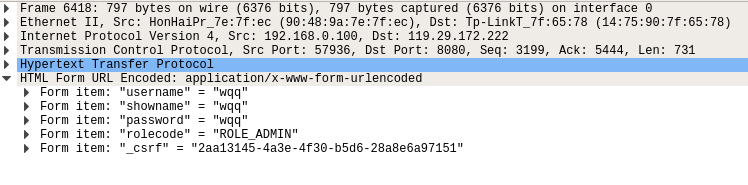
在没有使用HTTPS将数据加密传输之前，客户端POST到服务器的数据裸漏在中间人面前，我们可以获得用户的账户和密码．这里我们使用Wireshark抓包，抓取用户注册时的注册时传到服务器的数据．通过抓包，可以清楚地看到用户的账户和密码：

wireshark中抓取POST数据

C:\Users\ruanyubin\Desktop\infosec\c1.png

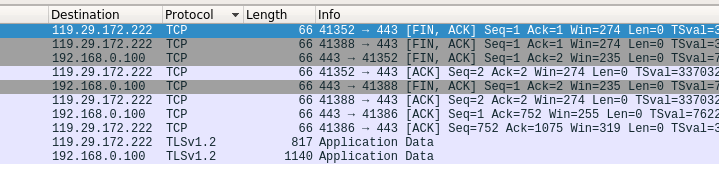
-

客户端在注册时POST到服务器的具体数据，其中的username和password就是账户密码．



## 使用HTTPS之后的数据传输状况

在使用了HTTPS协议之后，客户端POST到服务器的数据全都经过TLS层加密，因此重要的信息（帐号密码）不会被泄露．下图同样是在注册的时候进行抓包，但是已经看不见HTTP POST的数据了，所有数据交互都使用TLSv1.2协议进行，从而保证了数据安全：



## 应用安全性分析

**1. 部署tomcat服务器时不应该使用root用户权限运行tomcat**

这样做是因为，假如tomcat服务器存在缺陷而被攻击者控制，那么攻击者就可以利用root权限进行任何想做的操作了，而因为tomcat服务器是个较为大型的软件，并且多数的web应用都依赖于很多的第三方组件，所以大多数依赖于tomcat的web应用都存在被攻击者控制的危险．

（443端口的使用需要root权限，这个可以通过端口转发解决）

**2. WEB应用的文件管理**

摘一段来自Tomcat docs的话^[5]^

File permissions should also be suitably restricted. Taking the Tomcat instances at the ASF as an example (where auto-deployment is disabled and web applications are deployed as exploded directories), **the standard configuration is to have all Tomcat files owned by root with group Tomcat and whilst owner has read/write privileges, group only has read and world has no permissions. The exceptions are the logs, temp and work directory that are owned by the Tomcat user rather than root**. This means that even if an attacker compromises the Tomcat process, they can't change the Tomcat configuration, deploy new web applications or modify existing web applications. The Tomcat process runs with a umask of 007 to maintain these permissions.

重要的部分已经用黑体加粗．意思tomcat服务器目录下所有的文件夹(除了logs, temp和work之外)，都应该是属于root用户和tomcat用户组的，因为tomcat服务器不以root用户身份运行，所以就不能修改tomcat应用的配置，不能部署新的应用或者修改现有的web应用．

**3. 最小权限原则[6]**

其实上述的方法都符合一个最基本的原则：最小权限原则．最小权限原则的意思是要求计算环境中的特定个模块如进程、用户或者计算机程序只能访问当下所\*\*必需\*\*的信息或者资源．赋予每一个合法动作最小的权限，就是为了保护数据以及功能避免受到错误或者恶意行为的破坏。

## 数据库系统安全性分析

1. 使用PreparedStatement防止SQL注入

2. 数据库禁止远程连接

## 操作系统安全性分析

1. 除了常用的端口(22, 80, 8080, 443,22)外，不开放其他端口．

# 参考

[1]详解HTTPS是如何确保安全的

http://www.wxtlife.com/2016/03/27/%E8%AF%A6%E8%A7%A3https%E6%98%AF%E5%A6%82%E4%BD%95%E7%A1%AE%E4%BF%9D%E5%AE%89%E5%85%A8%E7%9A%84%EF%BC%9F/

[2] HTTPS详解

http://honglu.me/2016/01/13/HTTPS%E8%AF%A6%E8%A7%A3/)

[3] Tomcat doc, SSL-HOW-TO, /doc/ssl-howto.html

[4] SSL for tomcat in Debian

https://debian-administration.org/article/284/Creating\_and\_Using\_a\_self\_signed\_\_SSL\_Certificates\_in\_debian)

[5] Tomcat doc, SECURITY-HOWTO, /tomcat-8.0-doc/security-howto.html

[6]最小权限原则

https://zh.wikipedia.org/wiki/%E6%9C%80%E5%B0%8F%E6%9D%83%E9%99%90%E5%8E%9F%E5%88%99), Wikipedia