BRIAN KIROUAC

secure email for mobile devices

Brian Kirouac is the CTO and a principal security consultant for Security Horizon, Inc., and a faculty member of the University of Advancing Technology. He has an MS in both computer science and management, as well as a few certifications, including the CISSP, PCI-QSA, ISRM, and ISAM. He is a contributor to *The Security Journal* and a co-author of *IT Security Interviews Exposed: Secrets to Landing Your Next Information Security Job.*

bkirouac@securityhorizon.com

FOR SMALL BUSINESSES, SETTING UP mobile devices to send and receive email is fairly easy. Most smartphones can guide the user through the configuration process. The default configurations use plaintext (unencrypted) connections. Configuring these devices to connect in a secure manner takes more work. If you wish to do this on the cheap and use self-signed certificates, there are more steps required. This article attempts to guide you through the steps to configure a server to provide, and a mobile device to utilize, secure email services *cheaply*.

For many years I have been a staunch Linux user. Over the past few years I've slowly started to become an Apple fanboy. Started with an iPod, then an iTouch for my daughter. My first MacBook purchase was two years ago. My latest purchase was the iPad. I love the combination of toy and work device (I am writing this article on my iPad as I fly from COS to IAD). My wife loves the toy part.

Like most of us, I have both work and personal email accounts. For almost 20 years I have used ssh tunnels to secure my email connections, both for work and home. This has worked flawlessly but only for laptop or desktop use. For my Blackberry I had to open the IMAP port for connections from the Blackberry Internet Service (BIS) servers. I wasn't overly happy with this but it did serve a purpose. To get email on my iPad and my employees' iPhones something different had to be done.

BIS connections come from a limited-size, known set of IP addresses. iOs devices connect from all over the world with no set IP address. My goal was to enable access to send and receive email on iOs devices. Being a security company and not wanting to end up on "The Wall of Sheep" at conferences, the access had to be secure, meaning strongly encrypted.

Secure encrypted email access has been around for a long time: IMAPS, SMTPS, SMTP utilizing STARTTLS. All of these use SSL to encrypt the connection. I set up my first Sendmail server with an SSL certificate close to 15 years ago. This is nothing new.

Most SMTP servers do not care if you use a self-signed certificate. Setting up IMAPS for the BIS access also allowed the use of self-signed certificates. The iOs (and Android) devices, on the other hand, *do* care. By default these devices will reject self-signed certificates, but the use of self-signed certificates with these smart devices is doable.

For my home systems, I did not want to fork out the money for a commercial certificate. My company is a small business and did not want to spend money on a certificate that would only be used for internal employees. By pure coincidence, one of my customers switched from Blackberry to Droid and wanted the same type of access to their email. So now my problem was to create secure email communications for iOs and Android devices, allowing them to both send and receive email through company-controlled servers. The only resources available are man hours—no funds will be allocated. Get it done now!

The first thing I did was to use my GoogleFu to see who else has done this. I received very disappointing results. I could find parts of the process but not one that covered the complete process. Most of the guides I found referred to using a script that came with the individual applications; none of them was a comprehensive guide.

I am going to step through the process I used to configure a server to provide the required services. These steps must be completed prior to configuring your email accounts on the smartphones. The servers run Fedora Core or CentOS. The default install for these distributions uses Sendmail for SMTP and Dovecot for IMAP. For authentication, SASL will be used as provided by cyrus-sasl. For Webmail we use Squirrelmail on Apache HTTPD (although this article will limit its discussion of SSL configuration to a gloss on Apache's and nothing further).

To follow this how-to make sure you have the following packages installed:

- openssl
- httpd
- dovecot
- sendmail
- sendmail-cf
- cyrus-sasl
- iptables

The first hurdle to cross was generating a certificate that could be used across Sendmail, Dovecot, and Apache. This one certificate will also be for the global *.company.com domain. The reason for one certificate is to make it easier on the end users, since they will have to install the certificate on each of their devices. Some of our clients are not computer-centric and do not need nor want to understand the underpinnings of the system or jump through hoops for security. They want things to "just work."

The system administrator in me desires things to work after an unattended reboot. So I also have my certificate created without a passphrase. To extend the time between repeating these steps the certificate is good for 3650 days (almost 10 years).

The Linux distributions we use come with /etc/pki. In this directory is a set of folders designed to make things easier for creating and maintaining SSL certificates. For me this just makes things more complicated than they need to be. Instead of having folders for each type of service, I prefer just one folder that holds my certificates.

Making Certificates

The first step is to find your Certifying Authority certificate. Fedora and CentOS both come with one already in /etc/pki/tls/certs. This file belongs to the ca-certificates package.

In this same directory is a Makefile and a script called make-dummy-cert, which is part of the openssl package. Modifying the make-dummy-cert script is what I chose to create my certificates in a repeatable manner. Listing 1 contains the script after modification. Save this script as make-company-cert and run it.

```
./make-company-cert
```

This will generate three files: companyname-full.pem, companyname-key .pem, and companyname-cert.pem.

```
#!/bin/sh
umask 077
# The name of the file you wish to generate.
target="companyname"
# The answers to the questions openssl will ask.
answers() {
  # Country Name (2 letter code) [GB]:
  echo --
  # State or Province Name (full name) [Berkshire]:
  echo Colorado
  # Locality Name (e.g., city) [Newbury]:
  echo Colorado Springs
  # Organization Name (e.g., company) [My Company Ltd]:
  echo Company Name, Inc.
  # Organizational Unit Name (e.g., section) []:
  echo IT Department
  # Common Name (e.g., your name or your server's hostname) []:
  echo *.companyname.com
  # Email Address []:
  echo root@companyname.com
}

PEM1=`/bin/mktemp /tmp/openssl.XXXXXX`
PEM2=`/bin/mktemp /tmp/openssl.XXXXXX`
trap "rm -f $PEM1 $PEM2" SIGINT
answers | /usr/bin/openssl req -newkey rsa:1024 -keyout $PEM1 -nodes -x509
-days 3650 -out $PEM2 2> /dev/null
cat $PEM1 >> ${target}-key.pem
cat $PEM2 >> ${target}-cert.pem
cat $PEM1 >  ${target}-full.pem
echo ""   >> ${target}-full.pem
cat $PEM2 >> ${target}-full.pem
rm -f $PEM1 $PEM2
```

LISTING 1: MODIFIED VERSION OF MAKE _ DUMMY _ CERTS FOR CREATING SELF-SIGNED CERTIFICATES

Configuring Servers to Use SSL

Adding this certificate to Apache HTTPD was easy. By default the mod_ssl module is installed with the configuration file /etc/httpd/conf.d/ssl. Edit this file and change the following lines:

- Add the newly created certificate to the configuration file.

```
SSLCertificateFile /etc/pki/tls/certs/companyname-full.pem
```

- Comment out the other certificate lines.

```
SSLCertificateKeyFile
SSLCertificateChainFile
SSLCACertificateFile
```

The next step was to add the certificate to Dovecot. Edit the file /etc/dovecot.conf. For the protocols line choose:

```
protocols = imap imaps
```

Regular unencrypted IMAP is left for those still using the ssh tunnels, as I did not want to deal with changing everyone's configuration. We do not set up POP or POPS, so email stays on the server, where it can be backed up. If the user removes the email from the server to their laptop/desktop, we are no longer responsible for backing it up.

To configure Dovecot to use the SSL certificate, edit the following lines.

```
ssl_cert_file = /etc/pki/tls/certs/companyname-cert.pem
ssl_key_file = /etc/pki/tls/certs/companyname-key.pem
```

The next beast to slay is the line noise configuration of Sendmail. Adding the generated certificate to the configuration is easy. Go to the directory /etc/mail and edit the file sendmail.mc and add the following lines:

```
define('confCACERT_PATH', '/etc/pki/tls/certs')dnl
define('confCACERT', '/etc/pki/tls/certs/ca-bundle.crt')dnl
define('confSERVER_CERT', '/etc/pki/tls/certs/companyname-cert.pem')dnl
define('confSERVER_KEY', '/etc/pki/tls/certs/companyname-key.pem')dnl
```

To enable the different submission agents add the following lines:

```
DAEMON_OPTIONS('Port=smtp, Name=MTA')dnl
  DEFAULT Port 25
DAEMON_OPTIONS('Port=submission, Name=MSA, M=Ea')dnl
  DEFAULT Port 587
DAEMON_OPTIONS('Port=smtps, Name=TLSMTA, M=s')dnl
  DEFAULT Port 465
```

The next step is to configure Sendmail to use SASL for authenticating users before allowing them to relay email through your server. First you have to decide what type of authentication you wish to use. So that my users have just one password to forget, we use plain authentication which is tied to their host username and password. (In a possible future article, I'll discuss generating and using client certificates for authentication.) To the sendmail.mc file add the following lines:

```
define('confAUTH_OPTIONS', 'A p y')dnl
TRUST_AUTH_MECH('LOGIN PLAIN')dnl
define('confAUTH_MECHANISMS', 'LOGIN PLAIN')dnl
```

For testing purposes leave the p out of confAUTH_OPTIONS. When the p is included it will not accept plain text logins over unencrypted connections, which makes testing the new configuration a bit more complicated.

Generate your new sendmail.cf by running the following while in the /etc/mail directory:

```
make sendmail.cf
```

Now cyrus-sasl must be configured. By default sasl wants to use it's own database of usernames and passwords. This file is /etc/sasldb2. If this file does not exist sasl will fail and your users will not be able to authenticate. To create this file I just used touch:

```
touch /etc/sasldb2
```

Adjusting the Firewall

To ensure that users can access the newly configured services, iptables needs to be configured. Edit /etc/sysconfig/iptables and add the following lines (typically just after the rule to allow ssh (port 22) access):

```
-A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 25 -j
ACCEPT
-A RH-Firewall-1-INPUT -s 192.168.1.0/24 -m state --state NEW -m tcp -p tcp
--dport 143 -j ACCEPT
-A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 443 -j
ACCEPT
-A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 465 -j
ACCEPT
-A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 587 -j
ACCEPT
-A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 993 -j
ACCEPT
```

Now run the following commands to restart the services:

```
service httpd restart
service saslauth restart
service sendmail restart
service dovecot restart
service iptables restart
```

Ensure that all of these services are configured to start automatically on reboot. (I missed this step for one of the services on the customer's machine. The customer requested we reboot his server a few days after getting everything working. I missed the saslauth autostart and the user could receive but not send email with his Droid.)

```
chkconfig httpd on
chkconfig saslauth on
chkconfig sendmail on
chkconfig dovecot on
chkconfig iptables on
```

Testing

Now to test and ensure that everything is working. Testing from an external source is suggested. Use your browser to test the HTTPS access. Your prompts and how you view the certificate for HTTPS access will vary from browser to browser.

To test Sendmail, connect to port 25. Issue an ESMTP hello with `ehlo me` and you should get a response back similar to Listing 2. The response contains the `250-STARTTLS` line that shows it allows TLS connections. The `250-AUTH LOGIN PLAIN` shows the allowed authorization mechanisms.

Note: If you included the `p` option to the `confAUTH_OPTIONS` option in your sendmail.mc you will not see the `250-AUTH`.

```
user@host# ncat 1.2.3.4 25
220 companyname.com ESMTP Sendmail 8.14.3/8.14.3; Sun, 18 Jul 2010
14:59:55 -0600
ehlo me
250-companyname.com Hello localhost [4.3.2.1], pleased to meet you
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-8BITMIME
250-SIZE
250-DSN
250-ETRN
250-AUTH LOGIN PLAIN
250-STARTTLS
250-DELIVERBY
250 HELP
221 2.0.0 companyname.com closing connection
user@host#
```

LISTING 2: AFTER CONNECTING TO YOUR SMTP PORT, YOU SHOULD SEE STARTTLS IN THE LIST OF ENHANCED STATUS CODES.

To test Dovecot, follow Listing 3. The inclusion of STARTTLS in the welcome message shows the certificate has been loaded and encrypted connections can be used.

```
user@host# ncat localhost 143
* OK [CAPABILITY IMAP4rev1 LITERAL+ SASL-IR LOGIN-REFERRALS ID
ENABLE STARTTLS AUTH=PLAIN] Dovecot ready.
^C
user@host#
```

LISTING 3: TEST DOVECOT (YOUR IMAP SERVER) BY CONNECTING TO PORT 143 AND LOOKING FOR STARTTLS AGAIN.

Now reboot your server and test again. Once you are satisfied that the server is working properly you can start working on the clients.

Client Side

For the clients to accept the self-signed certificate as valid they must install the certificate on the device or application. Since they cannot get email, the best way is for them to download via HTTP(S). Copy the certificate to a location on the Web server accessible from the mobile device and your laptop/desktops. Depending on the device or application, the file extension will be either .crt or .pem. Listing 4 shows the commands I used to move the certificate files into place.

```
mkdir /var/www/html/certs
cp /etc/pki/certs/companyname-crt.pem /var/www/html/certs/cert.pem
ln /var/www/html/certs/cert.pem /var/www/html/certs/cert.crt
```

LISTING 4: PLACING CERTIFICATES WHERE THEY CAN BE DOWNLOADED

On Safari/Firefox as well as iOs/Droid devices, open the default Web browser and connect to the URL http://www.companyname.com/certs/cert.crt. You will be presented with a dialog box to install the profile on your device/or application. On the iOs device, this will create a new Profile in your configuration. This can be viewed under Settings->General->Profiles. Your profile will be labeled as *.companyname.com.

On your smartphone/mobile devices you can now proceed to create your email accounts. During the initial process choose the SSL or TLS options as offered.

For Thunderbird to accept the certificate, you must download the certificate to the local machine. You cannot use the ".crt" URL since your browser will think this is a certificate that needs to be installed. Thus you must use the URL http://www.companyname.com/certs/cert.pem. Save this file to a good location. Then within Thunderbird open the Preferences window. Choose Advanced->Certificates->View Certificates. Use the Import… button to import your new certificate.

You should now be able to check and send email securely from anywhere in the world where you have Internet connectivity. The fear of appearing on the Wall of Sheep should also be diminished (this fear should *never* go away).

Have a happy and secure computing day!