Configuring Apache2 for SSL Under Ubuntu Lucid

Yet another document describing how to install an SSL cert with Apache2 and Ubuntu Lucid. Plenty of references out there on the Interwebs already, but there's always room for one more.

1. Introduction

There's recently been a big fervor over FireSheep (http://codebutler.github.com/firesheep/), an easy-to-use tool for grabbing unencrypted session cookies (so dummy-proof there's no need for a Linux version since we should be able to figure out how to do it ourselves). The purpose of the publicity is to encourage people to use SSL to encrypt EVERYTHING session-oriented. Most people encrypt logins which protects the user from exposing their password, but the cookie is often sent in plaintext meaning that people on a local wifi network can snoop the cookie and hijack the session.

While I have recently implemented a forum (/forum) which could be subject to this sort of attack, my reason for switching to SSL was actually driven by laziness. My stoker_mon (/stoker) application uses a primitive security mechanism to protect write access to my Stoker (http://www.roacksbarbque.com) and it passes your password in plaintext - to me this is a much more critical flaw as it allows anywhere/anytime access to the account.

We're also in the midst of a big stink over Google collecting open wifi data that includes passwords. Google's being made out to be the bad guy here, but the real bad guys are the ones that pass passwords in plaintext. And the criminals are the ones who passed medical records in plaintext. This time it was Google, next time it may be someone malicious who posts the data on a torrent site or sells it to an identity broker.

1.1. What is SSL

SSL is a standard encryption technology found in browsers. It provides both encryption (which I care about) as well as authentication (which I'm a little less concerned about, people who want to hijack my domain are probably going to be very disappointed with the traffic they get).

SSL is up to version 3.0, prior versions should be disallowed as every major browser supports them. SSL has also been replaced with something called TLS 1.0 which, to the layman, is functionally identical to SSL. I'm not here to explain the exact differences (today), but you can think of it as SSL 4.0. Since SSL

3.0 doesn't have any critical security flaws like 2.0 has I'm fine with offering both SSL 3.0 and TLS 1.0 and letting the browser decide which to use.

You can tell a website is using SSL/TLS because the URL starts with https, but hopefully I don't need to tell you that. Most browsers also include a little lock symbol and interacting with that lock will give you information about their certificate.

To use SSL you need to get a certificate from a known source. Again in basic terms, when you connect to a site over SSL the server gives you a certificate which says "This is the site you're connecting to, it was authorized by these guys, and it's valid until this date." For example, when I visit Google's SSL enabled search site, I currently get a certificate that says I'm visiting www.google.com, that Thawte SGC CA is the person who vouches that it's Google, and that the certificate expires just before 2012. My browser checks to see if Thawte is a trusted Certificate Authority and then verifies that the certificate really came from Thawte. All this is about verifying that Google is who they say they are, you can imagine if I could pretend to be www.yourbank.com easily I could do a lot of naughty stuff.

Once the authentication phase is complete, we can start using this certificate to encrypt data. But you don't need to authenticate to encrypt, so you can create your own certificates if you don't care about warning messages. (Un?)fortunately browsers don't tend to like this and they often complain with self-signed certificates. For a website only you use this isn't a problem, but I'd rather not have to explain why there's a big red slash through the "https" in my users' browsers.

1.2. Why Implement SSL

SSL provides some level of security. It encrypts the data from the end user's browser through to your webserver, but don't think that using https://www.google.com (https://www.google.com/) will hide your information from Google, it just hides the contents of your search from your ISP and anyone on that local wifi network. Without this people can sniff your packets and see exactly what you see, as well as anything you submit via your browser. Generally I don't really care about most of what I browse on the web, if someone else can see the same news story I'm reading good for them. But any time there's authentication I want to protect my password and my session.

1.3. Why Not Implement SSL

SSL gives a false sense of security to some people. Some people actually believed that by using the SSL version of Google they would hide their search from Google itself, that's just silly. And SSL also doesn't hide which sites you visit, only the data that gets passed back and forth. Corporate IT still has a record of all your DNS lookups and every IP address you visit, which can be converted very easily into a hostname. If you need true security you'll want to set up an SSH tunnel or, even better, install something like the Tor Project (http://www.torproject.org). But these are end-user concerns as to why you shouldn't be demanding SSL everywhere.

If you're a server owner (which you should be if you're reading this) you need to do some homework first. Do you have a small server that doesn't see much use (like mine) or a large cluster? What's your budget look like? Do you have virtual servers? How sensitive is the information on your server and how important are the identities?

For small servers the overhead of SSL encryption is minimal at best. With application heavy servers that have a high CPU utilization you may need to consider even a minimal overhead. But you may also need to consider clustering and segregating the web server from the application server. For very large clusters you'll probably want to look into SSL termination standalone devices from the likes of Citrix and F5, these use hardware encryption and can help offload your server load immensely as well as allowing for global and local load balancing and other features.

If you have multiple servers on the same IP address you run into another issue, you don't know which hostname the customer is looking for until after you've handed out the certificate which means you'll either need to spring for a wildcard cert (if all the hostnames are in the same domain), live with authentication errors, or rely on SNI support. SNI (Server Name Indication) is a standard means of passing the hostname prior to encryption that is supported by most modern browsers on most modern operating systems, but support is critically lacking for IE on Windows XP. Firefox supports it on all platforms, and Chrome supports it on XP if you have the client install OpenSSL manually (http://www.slproweb.com/products/Win32OpenSSL.html), fine for a controlled corporate environment but if your website is on the public Internet you'd lose about 60% of your potential users (on average, check your access logs for the user agent information to get a distribution for your own site).

If you don't have the budget for the extra overhead or use virtual servers you may want to consider other mechanisms, but the overhead question is often overstated or trivialized so I would encourage you to at least try SSL in a trial situation to see what the overhead actually is before deciding.

1.4. Alternatives to SSL

It depends on what your purpose is. If it's authentication you'll need SSL, but if you're in a business where you need authentication you're either a financial firm or a popular website and you'll probably want to hire a professional rather than reading what some random person on the Internet has to say about things.

If you want encryption of everything you're again stuck with SSL, it's the best way to prevent snooping of the datastream from the browser to the webserver.

If you want last-mile encryption you do have some options. You can request that your users utilize encrypted proxies. The Tor Project (http://torproject.com) is a great encryption mechanism that provides anonymity as well. Creating an SSH tunnel to a home computer (preferably one NOT on wireless) also helps protect users by moving the insecure last mile from a potentially unencrypted public wifi node to their home wired ISP. Services like HTTPS Everywhere (https://www.eff.org/https-everywhere) may also be worth looking into. Some Content Delivery Networks also offer an HTTPS front end for an

HTTP-based webserver. None of these protect the user from the network exit point to your webserver, but they help against the most scary part - the unsecured wifi access point.

If it's just FireSheep session hijacking you're worried about you can look into encrypted cookies or client-side authentication methods. Remember, with FireSheep the hijacked sessions will likely appear to come from the same IP address as the actual session - this is hard to track unless you can pass some cookie encryption protocols during the HTTPS login session. This is beyond the scope of this document which has proselytized far too much already.

2. Get to the Point, How do I do it Already!

Sorry, I got a little carried away there. :) The first step is to get an SSL certificate. You can create a self-signed certificate (http://httpd.apache.org/docs/2.0/ssl/ssl_faq.html#selfcert), but that's often not a great way to go when there are other viable solutions.

2.1. Obtain a Certificate

Getting an official certificate is about 100 times more expensive that creating your own. So it's free. If you need strong authentication or it's worth the money to only go through this once every two years instead of every year there are good reasons to look into more expensive offerings, but essentially your choice of certs probably comes down to price and browser support. You get a certificate from a CA, Certificate Authority, and they need to be trusted by your browser in order to work out of the box. Pretty much any CA (including yourself if you choose to do so) will work, but you'll get error messages if you don't register the CA in the browser first. If you control the end users this may not be an issue, but for the general public they see a big red screen that says the cert isn't trusted and most will go someplace else. As an example of a CA I choose not to go with, CACert has poor/no browser support and here (http://wiki.cacert.org/ImportRootCert) are the directions each user needs to follow to use the site error-free.

I ended up going with StartSSL (http://www.startssl.com/) because they offer 1 year certs for free, they support just about every major browser except for Blackberry, they're free, and because the price was right. They are far from the only game in town, but I think I mentioned earlier that they're free.

Obviously this information will be out of date the moment they change their website, so be sure to follow THEIR instructions and not mine. To sign up for a cert simply open up their website in Firefox (Chrome is not supported for their rather secure authentication mechanism) and click on the Control Panel from the left-hand menu. Then click Sign-Up and follow the basic instructions they offer.

The first thing you'll want to do is follow the Validations Wizard for your domain. If you don't own your domain you're out of luck and you'll need to contact the person who does. It will ask you to validate an email address (you should have already done this when it started) and offer to send an email to one of a

variety of addresses (webmaster@yourdomain.com, admin@yourdomain.com, the email in the whois records, etc.). Choose an email address you have access to and it should send out the email.

Once you've validated your domain, click on the Certificates Wizard. It will prompt you for a key password, it's a good idea to create a random password for this to help keep things secure (your backup of the key should be password-protected), I used a password generator (http://www.freepasswordgenerator.com/) to create a 30-character random password using letters and numbers only. Copy and paste this passphrase into whatever password manager you use (or just a text file, but change the permissions using **chmod 400** so no one else can see it). In the next step just copy and paste the contents of the box into a file called `sslcert.key`. Now type in your top-level domain. Finally, copy and paste the textbox into `sslcert.crt`.

Easy, right?

2.2. Installing the Certificate

So now what? You'll need to get the certificates onto your webserver. Remember, anyone who can read these certs can impersonate you - use mechanisms like **scp** instead of **ftp** and if you use a flash drive make sure to delete the files if not wipe the data entirely.

Put the files someplace safe. Using `/etc/apache2/ssl/sslcert.key` and `/etc/apache2/ssl/sslcert.crt` is a good place. It's also good to **cd** to that directory. Now you can decrypt the cert by running **sudo copy keyname.key keyname.key.org; sudo openssl rsa -in keyname.key.org -out keyname.key**. You'll also want to download the StartSSL root certs by running **sudo wget http://www.startssl.com/certs/ca.pem; sudo wget http://www.startssl.com/certs/sub.class1.server.ca.pem**. Now run **sudo chown root *; sudo chmod 400 *** to protect them.

Congratulations! Your certs are now installed.

2.3. Configuring Apache

Now we need to configure Apache to use them. First enable `mod_ssl` by running **a2enmod ssl**. Don't worry, nothing we do here will go live until we reload Apache.

Before we continue you'll need to consider some things. First, is there any content you don't want to be SSL-enabled? Is there any you only want on SSL? In general I'd recommend making everything available over SSL and only selectively forcing content over SSL (and that's assuming you're CPU-bound on your webserver, if you're at 90% idle put everything over SSL and be done with it). The benefit of offering things like dumb static images or non-secure widgets over SSL is it allows people with SSL enabled websites to embed your content without errors. If you don't want this it may not be a concern - but if you want to embed the same image in one of your HTTPS sites you may thank me later.

You'll also want to backup every file we touch today. If you make a mistake Apache may be broken until you track down the problem.

Now you need to edit your `/etc/apache2/sites-available/default-ssl` to pretty much mirror your current `/etc/apache2/sites-available/default`. Feel free to add or remove aliases from either as you see fit, some things should be SSL enabled and some things you may not want to waste the overhead on.

Inside your <VirtualHost< definition you'll need to specify a few values. If you search for the variable names you may find some of them enabled, ensure that the following values are set:

```
SSLEngine on
SSLProtocol all -SSLv2
SSLCipherSuite ALL:!ADH:!EXPORT:!SSLv2:RC4+RSA:+HIGH:+MEDIUM
SSLCertificateFile /etc/apache2/ssl/sslcert.crt
SSLCertificateKeyFile /etc/apache2/ssl/sslcert.key
SSLCertificateChainFile /etc/apache2/ssl/sub.class1.server.ca.pem
SSLCACertificateFile /etc/apache2/ssl/ca.pem
```

The first part turns on SSL, excludes SSLv2 (which has a known vulnerability), and excludes some known-bad encryption algorithms. I'm assuming things are stored in `/etc/apache2/ssl/` and that your cert is called `sslcert.crt`, change these as you wish.

Finally, you'll want to activate the SSL website by linking it to the enabled sites using **ln -s /etc/apache2/sites-available/default-ssl /etc/apache2/sites-enabled/000-default-ssl**.

2.3.1. Setting up Apache to Only Serve SSL

I can't imagine any reason why I'd want a user to access my site using anything but SSL. This helps in that I don't need to mirror my `default` and `default-ssl` configs, I don't have to worry about what content I need to secure and I don't need to test the non-secure version of the website. Plus it's pretty darn easy to force everything over SSL.

First enable `mod_rewrite` by running **sudo a2enmod rewrite**. Now add the following lines to your `default` configuration inside the <VirtualHost> tags:

```
RewriteEngine   on
RewriteCond     %{HTTPS} !=on
RewriteRule     ^(.*)$ https://servername$1 [L,R]
```

Don't forget to change "servername" to your own hostname. Needless to say, you may want to perform this step *after* you've tested your site for SSL compatibility...

2.4. Activating the Configuration

Now you're one **sudo /etc/init.d/apache2 restart** away from enabling SSL. Before you do so, you'll want to make sure you've got port 443 open on your local firewall (**sudo ufw allow 443/tcp**) as well as port forwarding configured on your router. You'll also want to understand that this could mean a service interruption for your users - if you've got a busy website do it in the off hours. Using **reload** instead of **restart** may be a bit more graceful.

If you're prompted for a password you probably need to unlock your certificate file above. There was a mechanism whereby you could specify the password in another file, but as it was plaintext it was dropped. Realistically if someone gets root access to your unprotected certificate you're pretty much borked anyway.

Now try accessing your site normally, with http://www.yourdomain.com. That should work. Now try https://www.yourdomain.com both from inside and outside your network. You may see some messages about a mix of secure and insecure content, this means that you've explicitly referenced something (JavaScript code, an image, anything) over HTTP instead of HTTPS. In general you should try indirect references so it's easier to maintain these things, but external references you're pretty much stuck with what they offer unless you can download a local copy.

If you're used to accessing your server using an internal hostname (http://localhost or http://webserver instead of http://www.mydomain.com) you may notice that you get certificate warning messages. You may want to try to configure things so you can type in the full hostname, some routers (like my Linksys/Cisco model) behave well when a local IP sends a packet to the router's public IP address, others have problems so you may need to edit your `/etc/hosts` file with the private IP of your server. Note that this file is in `c:windowssystem32driversetchosts` on most Windows systems.

A. Discussion

Questions, comments, problems or complaints? Discuss this document on our forum here (https://www.ebower.com/forum/viewtopic.php?f=3&t=3).