Linux Network Administration

Apache2 with SSL
COMP1071 Summer 2020

HTTP vs. HTTPS

- HTTP is used to provide a protocol for client programs such as browsers to request resources (e.g. documents) from servers

- HTTPS is HTTP with SSL encryption and authentication, defaulting to port 443 - requires https protocol specified in the URL

- SSL encryption uses asymmetric keys, private and public

- SSL authentication uses a certificate, which can be signed by a third party who guarantees the certificate's validity

Apache2 SSL Module

- To use SSL with Apache, enable the ssl module using a2enmod ssl
- This enables the module for all sites that choose to use it
- To test if the module is available for use in a site file, use the <IfModule mod_ssl.c> </IfModule> stanza
- The <VirtualHost host:port> </VirtualHost> stanza should specify port 443 to use the default port

HTTPS Document Store

- The document store for an ssl-enabled website is no different from the document store for a non-ssl site
- It is specified with the same DocumentRoot directive in the site conf file
- The documents are not required to be encrypted on the server's storage device
- You can use the same document store for both http and https if you want

SSL-enabled Site File

- Similar file to non-SSL sites, can use /etc/apache2/sites-available/default-ssl for a template
- ServerName, ServerAdmin, VirtualHost, DocumentRoot, and Directory directives may all be ssl site specific
- Apache2 supports many tunable options for HTTPS, most defaults are fine for generic servers
- The site file is enabled using a2ensite just like a non-SSL site and then the server must be reloaded

SSL Directives and Dependencies

- SSLEngine must be set to on for SSL to be enabled on the virtual site
- Additionally, key and certificate files must be identified using SSLCertificateFile, SSLCertificateKeyFile directives
- Changing the certificate or key file content requires reloading the apache service

Sample HTTPS Site File

```
<IfModule mod_ssl.c>
    <VirtualHost sitename:443>
        ServerName sitename
        ServerAdmin webmaster@sitename
        DocumentRoot /sites/sitename
        <Directory /sites/sitename>
            Options Indexes
            AllowOverride None
            Require all granted
        </Directory>
        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined
        SSLEngine on
        SSLCertificateFile /etc/ssl/certs/sitename.crt
        SSLCertificateKeyFile /etc/ssl/private/sitename.key
    </VirtualHost>
</IfModule>
```

Private Key File

- A private key, commonly stored in /etc/ssl/private, is used to encrypt and decrypt SSL traffic
- The package providing ssl for apache is openssl, the command line tool used to perform ssl activities is openssl
- The private key can be generated using openssl genpkey or it can be generated at the same time as a self-signed certificate using the nodes and newkey options to the openssl req command
- The private key rules them all, keep it secret, keep it safe

Certificate File

- A certificate, commonly stored in a file in the /etc/ssl/certs directory, holds the server identity, signer information, and public key used for encryption and decryption

- Certificates can be self-signed or signed by a Certificate Authority (CA)

- Certificate signing requests can be sent to a Certificate Authority for signing, or certificates can be created locally if they are self-signed or you can use a private CA

- The openssl req command can be used to create signing requests, openssl x509 can be used to sign requests and manage certificate files

Certificate Procedure

- Create a private key

```
openssl genpkey -algorithm RSA -out /etc/ssl/private/sitename.key
```

- Create a certificate signing request

```
openssl req -new -key /etc/ssl/private/sitename.key -out ~/sitename.csr
```

- Send the csr to a Certificate Authority, they will send you back a certificate file which you store as /etc/ssl/certs/sitename.crt

- You can also run your own CA and sign your own requests

Self-signed Certificates

- Self-signed certificates are useful for encryption-only situations
- Self-signed certificates are useful for internal-only use if you install your certificates as trusted on the client hosts

```
openssl x509 -signkey /etc/ssl/private/sitename.key -in ~/sitename.csr -req -out /etc/ssl/certs/sitename.crt
```

- The openssl command can create the key and cert files in a single command for self-signed certificates

```
openssl req -newkey rsa:2048 -nodes -keyout /etc/ssl/private/sitename.key -x509 -out /etc/ssl/certs/sitename.crt -subj "/C=countrycode/ST=statecode/L=municipality/O=orgname/CN=sitename"
```

Private CA

- Private CAs are used as a way of deploying one or more certificates in a controlled fashion
- User and host certificates get signed by a private CA which is not part of the Public Key Infrastructure (PKI) provided by operating system vendors
- The private CA gets added to the trust list of hosts which are expected to use certificates from the private CA
- The private CA can sign multiple certificates and controls revocation of them
- Only the private CA certificate needs to be installed as a trusted CA certificate on the client hosts for all the certificates it has signed to be trusted by those clients

easy-rsa

- The OpenVPN developers created a set of scripts as part of their software package, to make it easier to deploy multiple certificates to VPN clients
- They later split the OpenVPN software into multiple projects, one of which (easy-rsa) had the scripts and tools to create and operate a private CA
- easy-rsa is available as a package for many distros and provides a straightforward way to generate and manage certificates for limited distribution

Generating CA Files

- After installing easy-rsa, set up your private CA (make-cadir)
- Easy-rsa versions before 3 used a vars file to set up the configuration, and put all generated files in a keys subdirectory - version 3 supports multiple pkis and places things under a pki subdirectory by default
- Use the easy-rsa build-ca command to produce the CA certificate files (ca.crt, ca.key) for the server and install them, only do this once!
- The CA private key is used to sign all certificates from the CA, make sure you keep it secure!

Generating Certificate/Key Files for the HTTPS Server

- Use the easyrsa script to build the certificate files (*servername*.crt, *servername*.key) for the https server, signed by your private CA

- Copy the https server's certificate files (*servername*.crt, *servername*.key) from the easy-rsa pki directory to the appropriate /etc/ssl subdirectories (certs, private) on the https server

- Install your private CA as trusted on the https server, and on the clients that will be using that server (copy the ca.crt file to /usr/share/ca-certificates/comp1071/ca.crt and dpkg-reconfigure ca-certificates)

Testing an SSL-enabled Site

- telnet cannot be used to test SSL-enabled sites
- curl or wget are the best choices for command line testing of https websites
- A browser can also be used, but often hides too much information to be useful
- The openssl command has a subcommand called s_client which can be used to do detailed examination of SSL connections and can give a telnet-style connection to a port that provides SSL connections

Logs

- Apache logs for ssl-enabled sites are identical to logs for non-ssl sites

- Stored in /var/log/apache2 by default

- Configurable on a per-site basis using ErrorLog and CustomLog directives