Usando OpenSSL en el mundo real. Quítate el miedo a usar OpenSSL.

Toni de la Fuente Díaz
Blyx.com
6 de Julio de 2006

OpenSSL es una herramienta libre muy potente que nos ayuda a implementar los protocolos y algoritmos de SSL (Secure Socket Layer) y TLS Transport Layer Security). OpenSSL incluye una API criptográfica de propósito general y una utilidad de línea de comandos muy potente. OpenSSL posee una licencia muy parecida a la de Apache.

La información y los comandos mostrados en este documento son genéricos, será valido para cualquier SO en el que esté correctamente instalado OpenSSL (Linux, Unix, Windowze, etc). Recuerda la web del proyecto OpenSSL es http://www.openssl.org

Hay una cita muy buena que dice: “Las palabras son enanos y los ejemplos son gigantes” o algo así. Entonces vamos al lío, veamos que podemos hacer con OpenSSL.

Todas las opciones de OpenSSL:

```
$ openssl help
openssl:Error: 'help' is an invalid command.

Standard commands
asn1parse     ca          ciphers       crl           crl2pkcs7
dgst          dh          dhparam       dsa           dsaparam
enc           engine      errstr        gendh         gendsa
genrsa        nseq        ocsp          passwd        pkcs12
pkcs7         pkcs8       rand          req           rsa
rsautl        s_client    s_server      s_time        sess_id
smime         speed       spkac         verify        version
x509

Message Digest commands (see the 'dgst' command for more details)
md2           md4         md5           mdc2          rmd160
sha           sha1

Cipher commands (see the 'enc' command for more details)
aes-128-cbc   aes-128-ecb aes-192-cbc   aes-192-ecb   aes-256-cbc
aes-256-ecb   base64      bf            bf-cbc        bf-cfb
bf-ecb        bf-ofb      cast          cast-cbc      cast5-cbc
cast5-cfb     cast5-ecb   cast5-ofb     des           des-cbc
des-cfb       des-ecb     des-ede       des-ede-cbc   des-ede-cfb
des-ede-ofb   des-ede3    des-ede3-cbc  des-ede3-cfb  des-ede3-ofb
des-ofb       des3        desx          idea          idea-cbc
idea-cfb      idea-ecb    idea-ofb      rc2           rc2-40-cbc
rc2-64-cbc    rc2-cbc     rc2-cfb       rc2-ecb       rc2-ofb
rc4           rc4-40      rc5           rc5-cbc       rc5-cfb
rc5-ecb       rc5-ofb
```

Vemos tres ayudas: Standard Commands, Message Digests y Cipher commands. Para ver la de estas tres secciones de forma independiente puedes usar los comandos:

```
$ openssl list-standard-commands
```

```
$ openssl list-message-digest-commands
$ openssl list-cipher-commands
```

Comprobación y verificación de código:

Hoy en día, gracias a Internet, podemos intercambiar gran cantidad de datos. Algo muy importante en cuanto a distribución de código fuente es comprobar la integridad de dicho código para confiar en la integridad de lo que nos estamos descargando. Para conseguir esto se hace una “huella digital” del archivo o archivos en cuestión. Esta huella digital es publicada junto con el archivo (tar.gz, zip, etc…) para poder comprobar su integridad.

OpenSSL soporta varios tipos de “huellas digitales” o digest algorithms, por ejemplo: MD2, MD4, MD5, SHA, SHA1, MDC2 y RIPEMD-160. Cada algoritmo puede ser invocado directamente o como opción del commando openssl dgst.

Vamos a ver unos ejemplos usados para sacar un digest MD5 del archivo /etc/secure/data:

```
$ openssl dgst -md5 /etc/secure/data
MD5(/etc/secure/data)= f268fc3e92ef844117365a3368b8fe45

$ openssl md5 /etc/secure/data
MD5(/etc/secure/data)= f268fc3e92ef844117365a3368b8fe45
```

OpenSSL puede combinarse con el comando find para hacer “huellas digitales” (fingerprints) de muchos archivos a la vez:

```
$ find /etc -type f | xargs openssl md5 > /etc/secure/md5_sigs.txt
```

El anterior comando creará un archivo MD5 hash de todos los archivos del directorio /etc. Estos finger prints deben ser almacenados como solo lectura y en un lugar seguro.

Cifrando (encriptando) datos:

Cifrar es el proceso de convertir datos (generalmente texto plano) a un formato alternativo (texto cifrado) que sea diferente al original. El proceso de cifrado de datos generalmente requiere una clave y usa una serie de algoritmos para realizar la transformación de texto plano a texto cifrado.

Los algoritmos de **clave simétrica** (algoritmos de clave compartida) usan la misma clave para cifrar y descifrar datos. Los algoritmos de **clave pública** (algoritmos de clave asimétrica) usan diferentes claves para el cifrado y el descifrado. Los algoritmos de clave pública toman su nombre por una de las claves que se utilizan, la clave pública, que puede ser distribuida a otras personas, los datos que son cifrados con una clave pública **sólo** pueden ser descifrados con la clave privada asociada.

OpenSSL soporta varios algoritmos de clave simétrica como: DES, 3DES, IDEA, Blowfish y AES. Cada algoritmo de clave simétrica puede ser invocado desde la línea de comandos pasando el nombre del algoritmo en el comando

openssl. Veamos un ejemplo para cifrar un archivo llamado passwd con Blowfish:

```
$ openssl bf -e -in /etc/secure/passwd -out /etc/secure/passwd.enc.bf
enter bf-cbc encryption password:
Verifying - enter bf-cbc encryption password



The encrypted version of passwd will be placed in /etc/secure/passwd.enc.bf.
The following example utilizes 3DES and the enc command to encrypt the file
sensitive_data. The encrypted contents are placed in
/etc/secure/sensitive_data.enc.3des:

$ openssl enc -e -3des -in /etc/secure/sensitive_data \
  -out /etc/secure/sensitive_data.enc.3des
enter bf-cbc encryption password:
Verifying - enter bf-cbc encryption password:
```

En ocasiones se envían archivos cifrados por mail para intercambiar datos de forma segura, esto puede ser un problema a la hora de incluir estos datos en un mail, para resolver este problema podemos usar el estándar base64 que es soportado por OpenSSL y representa el binario cifrado por texto en ASCII. Por ejemplo, para cifrar con AES-128 y codificar como base 64 podemos usaríamos el siguiente comando:

```
$ openssl enc -base64 -e -aes128 -in /etc/secure/data \
  -out /etc/secure/data.enc.aes128.b64
enter bf-cbc encryption password:
Verifying - enter bf-cbc encryption password:
```

Si el transporte utilizado para envío de datos cifrados no es seguro es altamente recomendable crear una firma digital o una huella digital del archivo cifrado para asegurarnos de que no haya sido modificado durante la transferencia.

Descifrando datos:

Para descifrar datos que han sido cifrados previamente usaremos la opción –d tras el algoritmo de cifrado en el comando openssl. El siguiente ejemplo muestra como descifrar el archivo /etc/secure/sensitive_data.enc.3des que fue cifrado con el algoritmo 3DES:

```
$ openssl enc -des3 -d -in /etc/secure/sensitive_data.enc.3des \
  -out /etc/secure/sensitive_data
```

Este comando dejará el fichero descifrado en el archivo /etc/secure/sensitive_data. Si quieres ver el contenido del archive en el terminal puedes usar el siguiente comando:

```
$ openssl enc -des3 -d -in /etc/secure/sensitive_data.enc.des3 | more
```

Generar contraseñas:

Podemos usar OpenSSL para generar contraseñas a través del comando passwd. Esta opción puede ser usada para automatizar el aprovisionamiento

de usuarios o la actualización de contraseñas. El siguiente ejemplo muestra como generar el MD5 de la contraseña “blah”:

```
$ echo blah | openssl passwd -stdin -1
```

La opción "-1" indica que usaremos MD5 como algoritmo y la opción “-stdin” indica que le pasamos la contraseña a través de la entrada estándar. Si tu sistema operativo no soporta MD5 las contraseñas puede ser creadas con CRYPT:

```
$ echo blah | openssl passwd -stdin -crypt -salt GH
```

Este comando generará una contraseña crypt de “blah” con “GH” como salt.

Certificados Digitales:

Un certificado digital es un “carnet de conducir” electrónico que se usa para identificar a un cliente o a un servidor. Las CAs (Certificate Authorities) son las responsables de expedir y generar los certificados digitales de forma que identifiquen y sean identificados como una autoridad fiable y de confianza para los solicitantes del certificados y los usuarios.

Un certificado contiene mucha información e incluye:

-La versión del certificado.

-Un número de serie único que identifica el certificado.

-Un atributo (Issuer) que identifica a la organización o entidad que ha generado dicho certificado.

-Rango de fechas en las que el certificado es válido.

-Un atributo (subject) que identifica el sitio para el cual el certificado ha sido generado.

-Y una firma digital usada por clientes y servidores para verificar la autenticidad del certificado y garantizar que el usuario del certificado es realmente quien dice ser.

Cuando una organización quiere solicitar un certificado a una Autoridad de Certificación (CA) necesitará enviar una petición de firma de certificado (CSR = certificate-signing request). El CSR contiene la clave pública, el common name (www.ejemplo.com) identificador único de un sitio y la información local que identifica a la organización solicitante. El siguiente ejemplo muestra como generar un CSR:

```
$ openssl req -new -outform PEM -keyform PEM -keyout secret.key \
              -out cert.csr -newkey rsa:1024

Generating a 1024 bit RSA private key
.............++++++
.............................................++++++
writing new private key to 'secret.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be
incorporated into your certificate request.
What you are about to enter is what is called a Distinguished
```

```
Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:Georgia
Locality Name (eg, city) []:Atlanta
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Example
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:www.example.com
Email Address []: sysadmin@example.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Durante la creación del CSR se crea un par de claves RSA de 1024-bit y se recoge información relacionada con el certificado a solicitar. OpenSSL solicita una pass-phrase o contraseña que será utilizada para cifrar el contenido de la clave privada. El CSR se crea en el archivo cert.csr. Este archivo contiene la clave pública, información de la localidad, ciudad, etc y el common name para identificar el nombre unico del sitio. Puedes ver el contenido del CSR con el siguiente comando:

```
$ openssl -req -in cert.csr -text
```

Una vez que has verificado el CSR ya puedes subirlo/enviarlo a tu CA favorita. La CA usará el contenido del fichero .csr junto con su clave privada para generar tu certificado digital.

Mostrando el contenido de un Certificado Digital:

Un certificado digital puede ser almacenado en muchos formatos. Los dos formatos mas comunes son PEM (Privacy Enhanced Mail) y DER (Definite Encoding Rules). OpenSSL puede mostrar el contenido de ambos formatos usando comandos x509. El siguiente ejemplo mostrará el contenido del certificado cert.crt.pem (PEM-encoded):

```
$ openssl x509 -in cert.crt.pem -inform PEM -text -noout
```

OpenSSL también permite mostrar atributos del certificado individualmente:

```
$ openssl x509 -in cert.crt.der -inform DER -noout -enddate
```

El comando anterior mostrará la fecha de caducidad del certificado cert.crt.der.

Conversión entre diferentes tipos de certificados:

Como hemos mencionado anteriormente, los certificados digitales se pueden almacenar en diferentes formatos. Esto puede causar problemas cuando un certificado necesita ser migrado entre servidores Web diferentes o distribuido entre aplicaciones. OpenSSL provee la opción x509 para convertir entre

certificados codificados como PEM- y DER-. El siguiente ejemplo convertirá un certificado PEM a formato DER:

```
$ openssl x509 -in cert.crt.pem -inform PEM -out cert.crt.der -outform DER
```

Una vez que el certificado ha sido convertido, será creado el archivo cert.crt.der. El siguiente ejemplo convierte DER a formato PEM:

```
$ openssl x509 -in cert.crt.der -inform DER -out cert.crt.pem -outform PEM
```

El comando de OpenSSL pkcs12 puede ser usado para importar y exportar certificados almacenados en una base de datos PKCS#12. El siguiente ejemplo exportará un certificado con el alias Server-Cert de una base de datos PKCS#12:

```
$ openssl pkcs12 -export Server-Cert -in cert.db.pkcs12 -out cert.crt.p12
```

Una vez que el certificado ha sido exportado, éste puede ser convertido a PEM o DER con el siguiente comando:

```
$ openssl pkcs12 -in cert.crt.p12 -out cert.crt.pem
```

Monitorizando la conectividad de de un Servidor Web Seguro con OpenSSL:

SCRIPT 1 (ver Anexo) muestra como OpenSSL puede ser usado para comprobar si un servidor Web-SSL está respondiendo a nuevas conexiones. La opción openssl s_client es invocada y se le pasa una petición HTTP GET al servidor Web después de crearse la conexión SSL. Si el servidor falla se registra un mensaje vía syslog y se envía un mail a la cuenta configurada en dicho script. Este script puede ejecutarse periódicamente a través del cron.

Comprobación de expiración de certificados:

Los certificados digitales pueden ser desarrollados para autenticar clients y garantizar la identidad de un servidor. Cuando se genera un certificado digital se le asigna una fecha de caducidad, la cual garantiza una vida finita de dicho certificado. Cuando se establece una conexión SSL tanto un cliente como un servidor comprueba la caducidad de cada certificado. Cuando caduca el certificado de un servidor web al cliente le aparece un mensaje de aviso que puede crear confusión. Disponer de un sistema que nos avise de la caducidad de nuestros certificados es una buena solución para evitar problemas. Vamos a ver la el script ssl-cert-check.

SCRIPT 2 (ver Anexo) (ssl-cert-check) es un shell script que usa GNU date y el commando openssl s_client. ssl-cert-check diferentes operaciones y puede ser configurado para enviar un mail cuando un certificado está a punto de caducar. ssl-cert-check muestra la ayuda cuando es ejecutado sin opciones:

```
$ ./ssl-cert-check
Usage: ./ssl-cert-check {{ [ -b ] && [ -f cert_file ] } || { \
  [ -s common_name ] && [ -p port] }} [ -e email ] \
```

```
  [ -x expir_days ] [ -q ] [ -a ] [ -h ]
  -a                : Send a warning message through email
  -b                : Print the expiration date for all
                      certificates in cert_file (batch mode)
  -e email address  : Email address to send expiration notices
  -f cert file      : File with a list of common names and ports
                      (eg., blatch.com 443)
  -h                : Print this screen
  -p port           : Port to connect to (interactive mode)
  -s commmon name   : Server to connect to (interactive mode)
  -q                : Don't print anything on the console
  -x days           : Certificate expiration interval
                      (eg. if cert_date < days)
```

El siguiente certificado puede ser usado para ver la caducidad del certificado de del www.daemons.net que está en el puerto 443:

```
 $ ./ssl-cert-check -s www.daemons.net -p 443

Host                       Status    Expires         Days Left
www.daemons.net:443        Valid     May 24 2005     363
```

También se puede ejecutar en batch mode, así podemos comprobar un listado completo de dominios proporcionando también los puertos:

```
$ cat ssldomains
www.spotch.com 443
mail.daemons.net 995
www.daemons.net 443

$ ./ssl-cert-check -b -f ssldomains

Host                       Status    Expires         Days Left
www.spotch.com:443         Down      ?                ?
mail.daemons.net:995       Expired   Oct 30 2002     -574
www.daemons.net:443        Valid     May 24 2005      363
```

ssl-cert-check tiene la capacidad de enviar un mail de alerta o alarma cuando el certificado está a punto de expirer. Así que nos permite automatizar, por ejemplo, con en el crontab del sistema operativo:

```
$ ./ssl-cert-check -b -f ssldomains -q -a -x 90 -e matty@daemons.net
```

El comando anterior comprueba cada dominio contenido en el archive ssldomains y envia un mail a matty@daemons.net si un certificado caduca en los proximos 90 dias o menos.

Conclusión:

En este artículo, hemos comentado algunos ejemplos de cómo usar OpenSSL para securizar datos y para gestionar certificados digitales. También hemos proporcionados dos scripts del ejemplo que se pueden utilizar para comprobar la caducidad de un certificado y el estado de un servidor web SSL. Todos los ejemplos citados deben ser probados antes de usarlos con entornos en producción.

Agradecimientos:

Agradezco a los desarrolladores de OpenSSL y a toda la gente que han puesto tiempo y energía en el diseño y la puesta en práctica de los algoritmos y de los protocolos criptográficos comentados en este artículo.

Referencias

Basado en el artículo (traducción libre):
http://www.samag.com/documents/s=9390/sam0409b/0409b.htm

Webs:

OpenSSL: http://www.openssl.org
ssl-cert-check: http://www.daemons.net/software
TLS 1.0 RFC: http://www.ietf.org/html.charters/tls-charter.html

Libros:

Schneier, Bruce. 1996. **Applied Cryptography**. John Wiley & Sons.
Stallings, William. 2003. **Cryptography and Network Security: Principles and Practice.** Prentice Hall.
Thomas, Stephen. 2000. **SSL & TLS Essentials: Securing the Web**. John Wiley & Sons.
Viega, John; Matt Messier; and Pravir Chandra. **Network Security with OpenSSL: Cryptography for Secure Communications**. O'Reilly & Associates.

Nota sobre el autor del artículo original:

Ryan Matteson has been a UNIX systems administrator for eight years. He specializes in network security, Web technologies, Storage Area Networks, high-availability systems, Linux, and the Solaris operating systems. Questions and comments about this article can be addressed to: matty91@bellsouth.net.

ANEXO:

SCRIPT 1 ***web-server-status.sh***

```
#!/bin/sh

PATH=/bin:/usr/local/bin:/usr/local/ssl/bin ; export PATH

# The following variable defines which host to connect to
HOST="www.example.com"

# The following variable defines the port to connect to on HOST
PORT="443"

#  Where to send E-mail with results
ADMIN="pageme@mydomain.net"

TMP="$HOME/connect.$$"
```

```
umask 077
touch ${TMP}

openssl s_client -quiet -connect ${HOST}:${PORT} > ${TMP} 2>&1 << EOF
GET / HTTP/1.0


EOF

if egrep "Server:" ${TMP} > /dev/null
then
        :
else
        logger -p daemon.notice "Failed to connect to ${HOST} on Port
${PORT}"
    echo "Failed to initiate SSL connection to ${HOST} on ${PORT}" \
        | /bin/mail -s "$0: Failed to connect to secure server \
    on ${HOST}:${PORT}" ${ADMIN}
fi

rm -f ${TMP}
```

SCRIPT 2 ***ssl-cert-check.sh***

```
#!/bin/bash
#
# Program: SSL Certificate Check <ssl-cert-check>
#
# Author: Ryan Matteson <matty91@bellsouth.net>
#
# Current Version: 1.2
#
# Revision History:
#   Version 1.2
#      Added checks for each binary required
#      Added checks for connection timeouts
#
#   Versions 1.1
#      Added checks for GNU date
#      Added a "-h" option
#      Cleaned up the documentation
#
#  Version 1.0
#      Initial Release
#
# Last Updated: 5-23-2004
#
# Purpose:
#  ssl-cert-check checks to see if a digital certificate in X.509
format
#  has expired. ssl-cert-check can be run in interactive and batch
mode,
#  and provides facilities to alarm if a certificate is about to
expire.
#
# License:
```

```
#   This program is free software; you can redistribute it and/or
modify it
#   under the terms of the GNU General Public License as published by
the
#   Free Software Foundation; either version 2, or (at your option)
any
#   later version.
#
#   This program is distributed in the hope that it will be useful,
#   but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU General Public License for more details.
#
#   You should have received a copy of the GNU General Public License
#   along with this program; if not, write to the Free Software
#   Foundation, 59 Temple Place - Suite 330, Boston, MA 02111-1307,
USA.
#
# Requirements:
#   Requires openssl and GNU date
#
# Installation:
#   Copy the shell script to a suitable location
#
# Usage:
#   Usage: ./ssl-cert-check {{ [ -b ] && [ -f cert_file ] } || \
#     { [ -s common_name ] && [ -p port] }}
#              [ -e email ] [ -x expir_days ] [ -q ] [ -a ] [ -h ]
#     -a               : Send a warning message through email
#     -b               : Print the expiration date for all
certificates in
#                        cert_file (batch mode)
#     -e email address : Email address to send expiration notices
#     -f cert file     : File with a list of common names and ports
#                        (eg., mail.daemons.net 443)
#     -h               : Print this screen
#     -p port          : Port to connect to (interactive mode)
#     -s commmon name  : Server to connect to (interactive mode)
#     -q               : Don't print anything on the console
#     -x days          : Certificate expiration interval (eg. if
cert_date < days)
#
# Examples:
#   Print the certificate expiration dates for a list of domains
specified
#   in the file ssldomains:
#     $ ssl-cert-check -b -f ssldomains
#
#  Print the certificate expiration date for mail.daemons.net:
#     $ ssl-cert-check -s mail.daemons.net -p 995
#
#  Run ssl-cert-check in quiet mode, check for all certificates that
will
#  expire in 75-days or less. Email the expiring certs to
matty@daemons.net:
#     $ ssl-cert-check -a -b -f ssldomains -q -x 75 -e
matty@daemons.net
#

PATH=/bin:/usr/bin:/usr/local/bin:/usr/local/ssl/bin ; export PATH
```

```
# Who to page when an expired certificate is detected (cmdline: -e)
ADMIN="root"

# Number of days to give as a buffer  (cmdline: -x)
WARNDAYS=30

# If QUIET is set to TRUE, don't print anything on the console
(cmdline: -q)
QUIET="FALSE"

# Don't send emails by default (cmdline: -a)
ALARM="FALSE"

# Location of system binaries
DATE="/export/home/matty/bin/date"
MAIL="/bin/mail"
OPENSSL="/usr/local/ssl/bin/openssl"

# Place to stash temporary files
CERT_TMP="$HOME/cert.$$"
ERROR_TMP="$HOME/error.$$"

# Set the default umask to be somewhat restrictive
umask 077

# Converts a date in string format passed as $1 into
# the number of seconds since 01/01/70 00:00:00 UTC.
# Replace this with another date normalization routine if
# you do not have GNU date available.
utcseconds() {
        utcsec=`${DATE} -d "$1" +%s`
        echo "$utcsec"
}

# Calculate the number of seconds between two dates
date_diff() {
        diff=`expr $2 - $1`
        echo "${diff}"
}

# Calculate days given seconds
date_days() {
    DAYS=`expr ${1} \/ 86400`
    echo "${DAYS}"
}

# Method used to print a line with a certificates expiration status
prints() {
    if [ "${QUIET}" != "TRUE" ]
    then
        MIN_DATE=`echo $4 | awk '{ print $1, $2, $4 }'`
        printf "%-30s %-8s %-20s %-5s\n" "$1:$2" "$3" "$MIN_DATE" "$5"
    fi
}

# Print out a heading
print_heading() {
    if [ "${QUIET}" != "TRUE" ]
    then
        printf "\n%-30s %-8s %-20s %-5s\n" "Host" "Status" "Expires"
"Days Left"
```

```
    fi
}

# Provide a listing of how the tool works
useage() {
    echo "Usage: $0 {{ [ -b ] && [ -f cert_file ] } || \
      { [ -s common_name ] && [ -p port] }}"
    echo "           [ -e email ] [ -x expir_days ] [ -q ] [ -a ] [ -h
]"
    echo "  -a               : Send a warning message through email "
    echo "  -b               : Print the expiration date for all
                               certificates in cert_file (batch mode)"
    echo "  -e email address : Email address to send expiration
notices"
    echo "  -f cert file     : File with a list of common names and
ports
                               (eg., blatch.com 443)"
    echo "  -h               : Print this screen"
    echo "  -p port          : Port to connect to (interactive mode)"
    echo "  -s commmon name  : Server to connect to (interactive
mode)"
    echo "  -q               : Don't print anything on the console"
    echo "  -x days          : Certificate expiration interval
                               (eg. if cert_date < days)"
}

# This method takes a HOST ($1) and PORT ($2) and checks to see
# if the certificate has expired
check_cert() {

    echo "" | ${OPENSSL} s_client -connect ${1}:${2} 2> ${ERROR_TMP}
1> ${CERT_TMP}

        if grep -i  "Connection refused" ${ERROR_TMP} > /dev/null
        then
                prints ${1} ${2} "Down" "?"  "?"

        elif grep -i "gethostbyname failure" ${ERROR_TMP} > /dev/null
        then
                prints ${1} ${2} "Down" "?"  "?"

        elif grep -i "Operation timed out" ${ERROR_TMP} > /dev/null
        then
                prints ${1} ${2} "Down" "?"  "?"

        elif grep -i "ssl handshake failure" ${ERROR_TMP} > /dev/null
        then
                prints ${1} ${2} "Down" "?"  "?"

    elif grep -i "connect: Connection timed out" ${ERROR_TMP} >
/dev/null
        then
        prints ${1} ${2} "Down" "?"  "?"

        else
        # Get the expiration date with openssl
                CERTDATE=`${OPENSSL} x509 -in ${CERT_TMP} -enddate -
noout \
                  | sed 's/notAfter\=//'`
```

```
        # Convert the date to seconds, and get the diff between NOW
and
        # the expiration date
                CERTUTC=`utcseconds "${CERTDATE}"`
                CERTDIFF=`date_diff ${NOWUTC} ${CERTUTC}`

                if [ ${CERTDIFF}  -lt 0 ]
                then
            if [ "${ALARM}" == "TRUE" ]
            then
                        echo "The SSL certificate for ${1} has
expired!" \
                        | ${MAIL} -s "Certificate for ${1} has
expired!" ${ADMIN}
            fi

            DAYS=`date_days $CERTDIFF`
                        prints ${1} ${2} "Expired" "$CERTDATE" "$DAYS"

                elif [ ${CERTDIFF} -lt ${WARNSECS} ]
                then
            if [ "${ALARM}" == "TRUE" ]
            then
                        echo "The SSL certificate for ${1} will expire
on \
                        ${CERTDATE}" | ${MAIL} -s "$0: Certificate for
${1}\
                        will expire in ${WARNDAYS}-days or less"
${ADMIN}
            fi

             DAYS=`date_days $CERTDIFF`
                         prints ${1} ${2} "Expiring" "${CERTDATE}"
"${DAYS}"

                else
                        DAYS=`date_days $CERTDIFF`
                        prints ${1} ${2} "Valid" "${CERTDATE}"
"${DAYS}"

                fi
        fi
}

while getopts abe:f:hp:s:qx: option
do
        case "${option}"
        in
                a) ALARM="TRUE";;
                b) REPORT_ALL="TRUE";;
        e) ADMIN=${OPTARG};;
                f) SERVERFILE=$OPTARG;;
        h) useage
           exit 1;;
                p) PORT=$OPTARG;;
        s) HOST=$OPTARG;;
        q) QUIET="TRUE";;
        x) WARNDAYS=$OPTARG;;
                \?) useage
            exit 1;;
        esac
```

```
done


if [ ! -f ${OPENSSL} ]
then
    echo "ERROR: $OPENSSL does not exist. Please modify the \$OPENSSL
variable."
    exit 1
fi

if [ ! -f ${DATE} ]
then
        echo "ERROR: $DATE does not exist. Please modify the \$DATE
variable."
    exit 1
fi

if [ ! -f ${MAIL} ]
then
        echo "ERROR: $MAIL does not exist. Please modify the \$MAIL
variable."
    exit 1
fi

if ${DATE} -d "Wed Jan 1 00:00:00 EDT 2000" "+%s" 2>&1 | grep -i
"illegal \
  option" > /dev/null
then
    echo "ERROR: \$DATE does not point to GNU date"
    exit 1
fi

# Baseline the dates so we have something to compare with
NOWDATE=`${DATE}`
NOWUTC=`utcseconds "${NOWDATE}"`

# Get the number of seconds to compare the UTC time with
WARNSECS=`expr ${WARNDAYS} \* 86400`

# Touch the files to avoid issues
touch ${CERT_TMP} ${ERROR_TMP}

# IF a HOST and PORT were passed on the cmdline, use that
if [ "${HOST}" != "" ] && [ "${PORT}" != "" ]
then
    print_heading
    check_cert "${HOST}" "${PORT}"

# If a file and a "-a" are passed on the command line, check all
# of the certificates in the file to see if they are about to expire
elif [ "${REPORT_ALL}" == "TRUE" ] && [ -f "${SERVERFILE}" ]
then
    print_heading
    while read HOST PORT
    do
        check_cert "${HOST}" "${PORT}"

    done < ${SERVERFILE}

# There was an error, so print how the tool works
else
```

```
    useage
fi

# Remove the temporary files
rm -f ${CERT_TMP} ${ERROR_TMP}
```