Assignment 4
120 pts

This is a three part assignment. For the first part of the assignment, you will create an AppArmor profile for a Linux server executable; for the second part, you will implement a simple SSL client and server; for the third, you will create a certificate signing request, create your own root certificate, and use your root certificate to sign our certificate signing request. Your solution is due on November 4, 2014 no later than 10 P.M. This is an individual assignment. See Section ?? for additional information on submitting your solution.

You may not discuss your solution with other students until seven days after the assignment deadline. You may consult any online references you wish. If you use any code in your answer that you did not write yourself, you must document that fact. Failure to do so will be considered a violation of the academic integrity policy.

1 Story

The Bytex corporation makes a successful byte exchange software product called Xchg. The Xchg product consists of a server and a client. The server stores one byte, which must be a printing ASCII character, that it will exchange with clients. A client can connect to the server and perform a single operation: swap a user-supplied byte, which must be a printing ASCII character, with the byte stored on the server. For example, if the server starts with a byte value encoding the ASCII character ‘x’ and client A connects and swaps its byte value encoding the ASCII character ‘y’ with the server’s, the server would now contain ‘y’. If a client B now connects and swaps its byte encoding the ASCII character ‘z’ with the server’s, it would get back ‘y’ (the byte stored on the server) and the server would store ‘z’ for the next client.

The underlying protocol is very simple. The server listens on a pre-arranged TCP port. To exchange a byte, a client makes a TCP connection to the server and sends its byte. The server reads the first byte sent by the client, sends back the currently stored byte, and saves the byte sent by the client, which becomes the current byte. If the byte sent by the client is not a printing character, the server sends back the byte value 0 and does not modify the current byte.

Xchg 1 client and server usage. The command-line usage of the server is:

```
xchgd port char
```

where `port` specifies the port on which the server should listen, and `char` the initial ASCII byte stored on the server. After each successful exchange, the server prints the currently stored byte, followed by a newline, to standard out.

The command-line usage of the client is:

```
xchg1 host:port char
```

where `host:port` specifies the host name and port number of the Xchg server to which the client should connect, and `char` the ASCII character to send to the server. On successful completion of the exchange, the client prints the previously-stored character sent by the server, followed by a newline, to standard out.

---

1 Printing character as defined by the C standard library function `isprint` for the C locale.
Version 1 of the Xchg product is not very secure, because it does the exchange in the clear and does not authenticate clients. You’ve been hired to implement version 2, which will use SSL to secure connection. You’ve been provided with the source code to version 1, and it is your job to develop version 2.

In addition, you’ve been placed in charge of creating a CA certificate and signing client and server certificates.

2 Containing the Xchg Server with AppArmor

AppArmor is a Linux Security Module and a set of tools that allow you to limit what a process can do. You can learn more about using AppArmor from the AppArmor documentation wiki:

http://wiki.apparmor.net/index.php/Documentation

You will be provided with a VirtualBox VM image with AppArmor installed. You must develop a policy—called a profile by AppArmor—for an xchg2d executable placed in the /usr/local/bin directory. (The xchg2d executable is the Xchg 2 server) This means, in particular, that your profile will be enclosed in a block like this:

/usr/local/bin/xchg2d {
...
}

Your policy should limit the Xchg protocol server to only the resources it needs. In other words, your policy should adhere to the principle of least privilege. Your policy will be graded on whether or not it allows the executable to access any system resources it does not need to function correctly.

3 Securing the Xchg Protocol with SSL

You must implement the client and server for Xchg protocol version 2. The protocol is to work as follows: on establishing a TCP connection the server, the client initiates an SSL handshake. Once an SSL connection is established, the exchange proceeds as in version 1, using the secured connection.

Both client and server will have their own certificates, private keys, as well as a certificate for the trusted CA. Both client and server must check certificates for validity. This means that each side must verify that the other side has provided a valid certificate chain from the trusted root CA. No certificate in the certificate chain may be expired. Intermediate certificates, if present, must be CA certificates. OpenSSL can perform these checks for you, however you must ensure the checking is done and check the result.

In addition to the source for version 1 of the protocol, you will be provided client and server binaries implementing version 2, which you can use to test your implementation. You will also be provided with valid certificates for testing.

Xchg 2 client and server usage. The command-line usage of the Xchg 2 server you develop must be:

```
xchg2d ca_cert server_cert server_key port char
```

where ca_cert is the CA certificate, server_cert is the server certificate, and server_key is the server private key. As in version 1, port specifies the port on which the server should listen, and char the initial ASCII byte stored on the server. After each successful exchange, the server prints the currently stored byte, followed by a newline, to standard out. The xchg2d.c skeleton file already includes code to store the command-line arguments in appropriately named variables.

The command-line usage of the Xchg 2 client you develop must be:

```
xchg2 ca_cert client_cert client_key host:port char
```
where `ca_cert` is the CA certificate, `client_cert` is the client certificate, and `client_key` is the client private key. As in version 1, `host:port` specifies the host name and port number of the Xchg server to which the client should connect, and `char` the ASCII character to send to the server. On successful completion of the exchange, the client prints the previously-stored character sent by the server, followed by a newline, to standard out. If the exchange failed, the server should print nothing and send back the byte value 0 to the client. The `xchg2.c` skeleton file already includes code to store the command-line arguments in appropriately named variables.

You may modify the `common.h` and `common.c` files if you wish. You must submit the client and server source code (`xchg2.c`, `xchg2d.c`, and optionally `common.h` and `common.c`). Do not modify the Makefile provided. Your code must build using the supplied Makefile.

## 4 Generating X.509 Certificates for SSL

**Certificate Signing Requests.** You must create two certificate signing requests, one for a client, and one for a server. Your CSR must contain a basic constraints extension, a key usage extension, and an extended key usage extension. These extensions must have values appropriate for this type of certificate you’re requesting. For this assignment, the server CSR must have the `extendedKeyUsage` value `serverAuth`, and the client CSR must have the `extendedKeyUsage` value `clientAuth`. In addition, the common name in the requested server certificate should be an IP address in ASCII dotted-quad notation (e.g. “127.0.0.1”). The two CSRs must be named `client.csr` and `server.csr`.

**CA certificate and signing.** In addition to the CSRs, you must create a value CA certificate and use it to sign exactly one of the certificate signing requests we provide (`hw4a.csr`, `hw4b.csr`, `hw4c.csr`). You must choose the CSR that is consistent with your use policy, namely that the certificate must be for an Xchg 2 client or server. Examine the signing requests carefully to determine which of the three is an appropriate CSR. Sign the CSR you believe is correct and name the signed certificate `hw4.cert`. Then submit your CA certificate (named `ca.cert`) and the signed certificate (`hw4.cert`).

## 5 Assignment Starting Files

You will be provided with several starting files for your assignment in the archive `hw4skel.tgz` available from the class Web page. The archive contains:
### File Description

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Makefile</td>
<td>Makefile for building version 1 (for your reference only)</td>
</tr>
<tr>
<td>xchg1.c</td>
<td>Source code for version 1 client (for your reference only)</td>
</tr>
<tr>
<td>xchg1d.c</td>
<td>Source code for version 1 server (for your reference only)</td>
</tr>
<tr>
<td>common.h</td>
<td>Header file for functions common to client and server (may submit)</td>
</tr>
<tr>
<td>common.c</td>
<td>Source code for functions common to client and server (may submit)</td>
</tr>
<tr>
<td>xchg2.c</td>
<td>Skeleton for version 2 client (must submit)</td>
</tr>
<tr>
<td>xchg2d.c</td>
<td>Skeleton for version 2 server (must submit)</td>
</tr>
<tr>
<td>xchg2d.armor</td>
<td>Skeleton for version 2 AppArmor profile (must submit)</td>
</tr>
<tr>
<td>hw4a.csr</td>
<td>One of three certificate signing requests (do not submit)</td>
</tr>
<tr>
<td>hw4b.csr</td>
<td>One of three certificate signing requests (do not submit)</td>
</tr>
<tr>
<td>hw4c.csr</td>
<td>One of three certificate signing requests (do not submit)</td>
</tr>
<tr>
<td>test_client.cert</td>
<td>Example client certificate (for your testing only)</td>
</tr>
<tr>
<td>test_client.key</td>
<td>Example client secret key (for your testing only)</td>
</tr>
<tr>
<td>test_server.cert</td>
<td>Example server certificate (for your testing only)</td>
</tr>
<tr>
<td>test_server.key</td>
<td>Example server secret key (for your testing only)</td>
</tr>
<tr>
<td>test_ca.cert</td>
<td>Example CA certificate (for your testing only)</td>
</tr>
<tr>
<td>x509.conf</td>
<td>Incomplete X.509 configuration file (for your convenience)</td>
</tr>
<tr>
<td>test_xchg2</td>
<td>Version 2 reference client binary (for your testing only)</td>
</tr>
<tr>
<td>test_xchg2d</td>
<td>Version 2 reference server binary (for your testing only)</td>
</tr>
</tbody>
</table>

### 6 VM Image

We have created a VirtualBox VM image configured with GCC 4.7.2, OpenSSL 1.0.1e, and AppArmor 2.7.10.3. The image will be distributed via BitTorrent; the torrent file hw4vm.torrent will be available on the class Web page. The credentials for the VM are student: hacktheplanet and root: hackallthethings. The VM is configured with the following services:

<table>
<thead>
<tr>
<th>Host Port</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>2222</td>
<td>SSH</td>
</tr>
</tbody>
</table>

### 7 Submitting the Solution

Your solution to this assignment consists of the following files: xchg2.armor, xchg2.c, xchg2d.c, common.h (optional), common.c (optional), client.csr, server.csr, ca.cert, and hw4.cert. You may also include a README file.

Your solution must by submitted via email to cs127f1@ieng6.ucsd.edu by November 4, 2014, 10:00 P.M. Pacific time. It must be a gzip-compressed tar archive, signed with your PGP key and encrypted to the cs127f1@ieng6.ucsd.edu PGP key, which is provided on the CSE 127 Web page and has key fingerprint:

```
ED49 BC3B 8992 A1E0 D2DD 66DC A1EF 6B86 7864 D1BD.
```

You may either send a signed and encrypted email, or send a plain email with the encrypted archive and separate signature.

### 8 Change History

This is the third revision of the assignment. It fixes an error in the definition of xchg2 command-line arguments. The `client_cert` and `client_key` arguments were incorrectly described as the server certificate and server key.
The second revision fixed an error in the name of the program shown in the command-line usage of the version 1 client and version 2 client. The program was incorrectly shown as xchg1d in the version 1 command, and as xchg2d in the version 2 command.