CS.444 Spring 2017

Security Programming Assignment: SSL/TLS

Due: March 2, 10 pm.

In this exercise, you will learn how TLS helps to secure connections to a web server, and how these connections can be intercepted via active attacks. This project will use the "ssl" package of Python as well as the "mitmproxy" software package.

Part 1: Writing a simple TLS-capable (HTTPS) web client

Your first task is to develop a small Python web client that can access web objects via TLS. This portion of the assignment can re-use code from the socket programming assignment. The new elements in the assignment are to add support for SSL/TLS to your basic client.

To do this you must use the Python "ssl" package, which provides a TLS/SSL wrapper for Python sockets. Documentation for the "ssl" package can be found online here:

https://docs.python.org/3/library/ssl.html

There is some example code online, though please do not copy and paste -- write your own implementation. Adding client-side support should incorporate the following steps:

1. Instantiating an SSL/TLS context
2. Setting TLS/SSL client options
3. Setting the default TLS root certificate paths (e.g., with load_default_certs())
4. Instantiating a TCP socket
5. Wrapping the TCP socket with the SSL/TLS context
6. Initiating a connection (to an SSL/TLS server at port 443) and performing an HTTP GET
7. Closing the socket

Steps (4, 6 and 7) should already be present in your existing code. Your task is to add the remaining steps to your program. Your program should correctly validate TLS certificates and hostnames.

Your program should also accept the following options as flags on the command line.

--tlsv1.0, --tlsv1.1, --tlsv1.2, --sslv3. These options should force a specific version of TLS to be used by the client If no option is passed in, you can use the default options chosen by the "ssl" package.

--ciphers <cipher list>. This should operate like the similar command in the **curl** utility. To use the --ciphers flag pass in a list of colon-separated ciphersuite names in your order of preference. For example: --ciphers DHE-DSS-AES256-SHA:DH-RSA-AES256-SHA. You can use the "**openssl** ciphers" command to obtain a list of ciphers installed on your machine.

--cacert <CA certificate file>. This option points to a local file in PEM format containing the CA (root) certificate(s) to be used in verifying the server certificate. If this file is not provided, your client should load a default list using the load_default_certs() function.

Your deliverable from this part of this assignment is a Python program that can be run with the following syntax, where "server_host" is the IP or host name, "server_port" is the server port, "optional_flags" contains the flags above, and "filename" is the path to a file on the server:

```
tls_client.py <optional flags> server_host server_port filename
```

The output of this program is the data sent back by the server, including HTTP headers.

Part 2: Writing a simple TLS-capable (HTTPS) web server

Your second task is to develop a small Python web server that can serve web objects via TLS. This portion of the assignment can re-use your server code from the previous socket programming assignment. The new elements in the assignment are to add support for SSL/TLS to your basic server.

To make this work you will also need to generate a self-signed certificate for the domain your server runs on (e.g., localhost), and you will need to provide this certificate to your server. You can find instructions for generating this certificate with the **openssl** command line tool online.

To test your server with your client, you will need to disable certificate validation.

Part 3: Printing TLS/SSL certificates

In this part of the assignment you will modify your program from Part 1 to print only the TLS/SSL certificate information for the web server. You should print your certificate information in a human-readable text format (rather than binary). The deliverable program should have the same options as the client from part 1.

```
tls_cert.py <option_flags> server_host server_port filename
```

The output of this program is the human-readable certificate of the web server. You do not need to print the data or headers sent back from the server, only the certificate. An example output might look like:

```
{'issuer': ((('countryName', 'IL'),),
            (('organizationName', 'StartCom Ltd.'),),
            (('organizationalUnitName',
              'Secure Digital Certificate Signing'),),
            (('commonName',
              'StartCom Class 2 Primary Intermediate Server CA'),)),
 'notAfter': 'Nov 22 08:15:19 2013 GMT',
 'notBefore': 'Nov 21 03:09:52 2011 GMT',
 'serialNumber': '95F0',
 'subject': ((('description', '571208-SLe257oHY9fVQ07Z'),),
             (('countryName', 'US'),),
             (('stateOrProvinceName', 'California'),),
             (('localityName', 'San Francisco'),),
             (('organizationName', 'Electronic Frontier Foundation, Inc.'),),
             (('commonName', '*.eff.org'),),
             (('emailAddress', 'hostmaster@eff.org'),)),
 'subjectAltName': (('DNS', '*.eff.org'), ('DNS', 'eff.org')),
 'version': 3}
```