**CS255: Introduction to Cryptography** **Winter 2013**

# Programming Project #2

Due: Friday, March 15th, 2013, 11:59 pm

## 1 Overview

### 1.1 Introduction

For programming project 2, you will implement a *man-in-the-middle (MITM)* attack on SSL,[1] using an SSL proxy server. You will also implement a simple (command-line) administrative interface for the proxy that will make use of password authentication.

### 1.2 Background

Recall that an eavesdropper on an SSL connection has little power because of the encryption being used, but if an attacker is able to trick the user into using the attacker's public key rather than the intended receipient's, this security is lost. While a real attacker would likely intercept and manipulate the network packets direcly to implement this attack, you will have to make use of an SSL proxy. After a client (i.e. a web browser) is configured to make use of an SSL proxy, all client SSL requests are intercepted by the proxy and relayed to the intended remote webserver.

After an initial plaintext proxy *CONNECT* request by the client, normally the proxy just forwards the encrypted data to the server. However, instead of forwarding the initial request to the remote server, your proxy will setup its own connection with the remote server and setup a connection to the client using its own certificate. Then all traffic between the client $\leftrightarrow$ proxy and the proxy $\leftrightarrow$ web-server is SSL encrypted, but with different keys. This means that the proxy has access to the plaintext data sent and received by the client. Having the proxy use a single, fixed SSL server certificate is not ideal, though, because modern web browsers check the *common name (CN)* field of the certificate against the domain name of the remote server. So, to mount a more transparent MITM attack, the proxy will have to generate new server certificates on the fly, for each new client request. Web browsers will still complain **once** that the certificate is not trusted, but if the user clicks past this warning, then the attacker wins.

You will be learning :

- **keytool (command line utility)** to generate and manage keys and certificates.
- **IAIK-JCE APIs** to create and sign certificates programmatically.
- **JSSE (Java Secure Socket Extension)** to do secure networking.

### 1.3 Requirements

We will provide you with code for a basic SSL proxy, and you will need to do the following:

[1] We emphasize that this project is for educational purposes only, and should *never* be used outside of this class.

- Modify the SSL proxy to dynamically generate new SSL server certificates, based on the domain name of the requested remote web server.
- Implement password authentication, over an SSL connection, for a simple administrative interface.
- Implement a challenge/response based user authentication scheme. (Extra Credit)

We will examine each of these features in detail below. Since we have not yet covered in the lectures all of the topics explored by this project, you may wish to start first on those aspects of the project that you can do immediately and save the other parts for later.

## 2 Description

### 2.1 Secure communication

You will be working with network sockets. The JCE provides an abstraction for secure sockets in the javax.net.ssl package and this relieves us from explicitly performing the key exchange, encryption and integrity of the messages transferred over these sockets.

### 2.2 Public Key Infrastructure

#### 2.2.1 Offline Key Generation

The SSL proxy has a public/private key pair which is generated offline using **keytool**. The **keytool** is used to generate a *keystore* for each entity in the system. Before the system is bootstrapped, you will have to generate a public/private key pair for the SSL proxy. The public key of the proxy is self-signed.

#### 2.2.2 Generating new server certificates

After connecting to a remote webserver, the proxy will have to create a new server certificate which has the same Distinguished Name (DN) field and serial number as the remote webserver's certificate. (The DN includes the Common Name, CN, which represents the domain name for which the certificate is valid.) This new certificate will then be presented to the client, for use in an SSL session. You will use classes from the IAIK library to create and sign these new server certificates.

### 2.3 Password Authentication

In addition to implementing the MITM attack with the proxy server, you will implement a simple remote administrative interface for the proxy server. This will allow the attacker to remotely log into the server and issue commands. In order to ensure only the attacker can log in, the interface will use password authentication. To connect to the proxy server, the administrative program will set up an SSL connection to the proxy server and transmit the attacker's password, along with a command to execute. The proxy server maintains a password file, which contains a salted hash of the user's password. When the proxy receives a log in request, it should compare the hash of the received password with the stored hash from the appropriate user, executing the requested command if they match, and otherwise closing the connection.

You will need to implement the following commands:

- **shutdown**: shutdown the MITM proxy server
- **stats**: list how many requests have been proxied

### 2.4 Challenge/Response Authentication (Extra Credit)

For *extra credit* you may implement a more sophisticated challenge/response authentication method along with the password authentication described above. In this the proxy server will issue a challenge to the user, which they must then answer with a response that proves their identity. Several such methods exist, and we leave it to you to decide on the precise details of the method. If you do choose to implement such a system, you should provide a brief explanation of it, and why it provides a secure authentication mechanism.

### 2.5 System setup

There are four main types of entities in our system: the user's web browser, the remote webserver the user is attempting to connect to, the proxy server intercepting the connection, and the administrative client to the proxy server. Once the proxy server is started and begins listening for connections, the user's web browser should be configured to use an SSL proxy with the hostname and port used by the MITMProxyServer. All SSL connections by the web browser will then be routed through the proxy server.

When the browser attempts to make an SSL connection, the proxy will parse the CONNECT request and make its own SSL connection to the requested server. The proxy will use the connection to obtain the remote server's certificate. The proxy will then create a forged certificate which copies the entries in the remote server's certificate (the server's Distinguished Name (DN), and its serial number), but uses the proxy's own public key. The proxy then signs this generated certificate with its own private key (the public/private key pair is loaded from a keystore at startup). When the key is used as a server key, it should be accompanied by the server's DN, while when it is used It then passes this generated certificate back to the web browser, setting up an SSL connection between itself and the browser. The proxy then passes data, which of course it sees in the clear, between the two connections.

Note that here the proxy reuses the same key pair over and over for each remote server's forged public key. In each case, the certificate confirming the validity of this key should be self-signed (i.e., signed by the proxy's own private key), although the data (DN and serial number) will differ between remote servers. The "Issuer" should be the proxy server's own DN, which you determine when you run `keytool` — you can use whatever data you want here, but preferably something that sounds like a Really Official Certificate Authority to present to the user in-browser.[2]

In addition to listening for connections from web browsers, the proxy server listens for connections from the admin client on a separate port. When the admin client wants to connect to the proxy server, it opens an SSL connection to the proxy on its hostname and admin port and transmits a password. The proxy server then consults its password file (specified at startup and stored on disk) and authenticates the user.

[2]This is not the best way to write the proxy: an astute user, or a vigilant browser, may realize that they are receiving the same public key over and over again. A better approach would be to generate a fresh key pair for each server, and sign this key with the proxy server's own key, which itself is self-signed (mimicking a CA key). However, this approach gets quite messy in Java and we will not require it in your implementation.

## 3 Implementation

As with the first programming project, we have provided you with starter code. The starter code illustrates the basic socket and thread programming. See the following section for links of tutorials on socket and thread programming. In addition to Sun Java JCE library, you need IAIK JCE extension library to create and sign X509 certificates. The library is included in the starter code.

### 3.1 Description of the code

Here is a brief description of some of the starter code. The files you need to modify are in **bold**, and contain the text `TODO(cs255)` at the places you will need to change:

| | |
|---|---|
| **Makefile** | Makefile for the project; modify this file to compile any new classes that you add. |
| **MITMProxyServer.java** | Starts up the SSL proxy server. |
| **HTTPSProxyEngine.java** | The core SSL proxy code. |
| **MITMSSLSocketFactory.java** | Used in the creation of new SSL sockets. |
| **MITMAdminServer.java** | Creates connections with authorized admin clients. |
| MITMAdminClient.java | Command line tool for remotely accessing the proxy server. |
| ProxyDataFilter.java | Logs the (plaintext) data exchanged between the client and remote webserver. |
| ConnectionDetails.java | Holds information about the two endpoints of a TCP connection. |
| CopyStreamRunnable.java | Blindly copies data from an InputStream to an OutputStream. |

| | |
|---|---|
| MITMPlainSocketFactory.java | Used to create unencrypted sockets, to handle the initial browser proxy CONNECT request. |
| ProxyEngine.java | Abstract parent class of HTTPSProxyEngine. |
| StreamThread.java | Copies data from an InputStream to an OutputStream, using a ProxyDataFilter to record the data that's being streamed through. |

## 3.2 Running the code

You should spend some time getting familiar with the provided framework and reading the comments in the starter code. You will need to copy the *cs255-p2-starter-code.tar.gz* file to your account. You will also need to set your CLASSPATH environment variable correctly (either set it permanently using whatever mechanism your shell uses, or prefix **CLASSPATH=".:iaik_jce.jar"** to any **java** commands). In Eclipse, you can instead add *iaik_jce.jar* as a library.

1. Change your browser settings to make use of the SSL proxy. (In Firefox, this is done by opening "Preferences", "Advanced", "Network", clicking "Settings...", "Manual proxy configuration", and entering "localhost" and "8001" in the line that reads "SSL proxy".)
2. Start the SSL proxy:

   **java mitm.MITMProxyServer -keyStore <yourkeystore> -keyStorePassword <kspwd> -outputFile <logfile>**
3. Run an admin client:

   **java mitm.MITMAdminClient -password <pwd> -cmd <cmd>**

## 3.3 Crypto Libraries and Documentation

*NOTE: We require that your submission work with the Java API version on the myth machines. Also, you should use the version of the IAIK library that we provide with the starter code.*

In addition, you may find the `bcrypt` library useful. A Java implementation can be found at:
`https://www.mindrot.org/projects/jBCrypt/#download`
If you wish, you can include the file `BCrypt.java` (from the 0.3 release) in your submission.

The following are some links to useful general documentation:

- **Java API**
  http://java.sun.com/j2se/6/docs/api

- **IAIK-JCE API**
  http://javadoc.iaik.tugraz.at/iaik_jce/3.13/iaik/security/provider/IAIK.html
- **Java Keytool Manual**
  http://download.oracle.com/javase/6/docs/technotes/tools/solaris/keytool.html
- **JCA Reference Guide**
  http://download.oracle.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html
- **JSSE Reference Guide**
  http://download.oracle.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html
- **Java Tutorial on Socket Programming**
  http://download.oracle.com/javase/tutorial/networking/sockets/
- **Java Tutorial on Concurrency**
  http://download.oracle.com/javase/tutorial/essential/concurrency/index.html
- **IBM Tutorial on JSSE (Introductory)**
  http://www.ibm.com/developerworks/java/tutorials/j-jsse/
- **IBM Tutorial on JSSE (Advanced)**
  http://www.ibm.com/developerworks/java/library/j-customssl/

Some classes/interfaces that may be relevant:

- java.security.SecureRandom
- java.security.KeyStore
- java.security.PublicKey
- java.security.PrivateKey
- javax.net.ssl.KeyManagerFactory
- javax.net.ssl.KeyManager
- javax.net.ssl.TrustManagerFactory
- javax.net.ssl.TrustManager
- java.net.ServerSocket
- java.net.Socket
- javax.net.ssl.SSLSocket
- javax.net.ssl.SSLServerSocket
- javax.net.ssl.SSLSocketFactory
- javax.net.ssl.SSLContext

- javax.net.ssl.SSLSessionContext
- java.security.cert.Certificate
- javax.security.cert.X509Certificate
- iaik.x509.X509Certificate

## 4 Miscellaneous

### 4.1 Questions

- We strongly encourage you to use Piazza as your first line of defense for the programming projects. TAs will be monitoring it daily, and other students may have faced the same question and be able to answer it for you.
- The staff can also be reached at `cs255ta@cs.stanford.edu`, but response time may be longer and questions are more likely to be inadvertently dropped. Note that even if your question would divulge part of your solution, you can still create a "private question" on Piazza, so there should not be many questions that require an email.

### 4.2 Deliverables

Your submission should be in the form of a ZIP archive, entitled `P2_id.zip` where `id` is your SUNet ID (or `P2_id1_id2.zip`, for groups of two) — e.g., `P2_jzim_mzhandry.zip`. This archive file should be attached to an email sent to `cs255ta@cs.stanford.edu` with the subject line: `P2_id` (or `P2_id1_id2`). The archive should contain the following:

- Your well-decomposed, well-commented solution to the assignment. (Please run `make clean` before you submit.)
- Your keystore file, which should be called `keystore`, should have its own password, and should contain a key pair (under alias `mykey`) that is itself password-protected. (The key password and the keystore password should be identical.)
- Your password hash file (for the admin server), which should be called `pwdFile`. You need not provide the code used to generate it, but the admin password should not be the same as the key/keystore password.
- A file `passwords.txt` containing exactly two newline-terminated lines: on the first line, the key/keystore password; and on the second line, the admin password (whose hash should be stored in `pwdFile`).
- A proxy log file, `log.txt`, from one of your test sessions. (Take care not to leave any passwords or credit card numbers in it!)
- A writeup document, `README.pdf` (or, if you cannot produce a PDF, `README.txt`), containing:
  - The names and SUNet IDs of the people in your group.

  - A description of the design choices you made in implementing each of the required security features (*illustrate with diagrams as needed; be concise and clear*).
  - A sequence of steps which will be required to run your system (this should be as simple as possible).
  - Short answers to the following questions:
    1. Suppose an attacker controls the network hardware and can intercept or redirect messages. Show how such an attacker can control the admin server just as well as a legitimate admin client elsewhere on the network. Give a complete and specific description of the changes you would make to fix this vulnerability.
    2. Suppose an attacker is trying to gain unauthorized access to your MITM server by making its own queries to the admin interface.[3] Consider the security of your implementation against an attacker who (a) can read the admin server's password file, but cannot write to it; (b) can read and/or write to the password file between invocations of the admin server. For each threat model, either show that your implementation is secure, or give an attack. (N.B.: For full credit, your implementation should at least be secure under (a).) What, if anything, would you need to change in order to make it secure under (b)? *If your answer requires any additional cryptographic tools, you should fully specify them (including the names of any algorithms, cryptosystems, and/or modes of operation that you would use.)*
    3. How would you change a web browser to make it less likely that an end user would be fooled by a MITM attack like the one you have implemented? (This is an important question to ask because when dealing with security, we never just build attacks: we also need to think of ways to prevent them.)
    4. (*optional — worth 0 points*) How does your MITM implementation behave on dynamic web pages, such as Google Maps? Why is this the case? How could you modify your proxy to be more convincing in these cases?

### 4.3 Grading

This project is worth 20 points (20% of your final grade). The breakdown will be approximately as follows:

- SSL proxy implementation (7 points)
- SSL admin interface (5 points)
- documentation—design, instructions, etc. (3 points)
- answers to questions 1-3 above (5 points)
- challenge-based admin authentication—working implementation and good design description/justification (2 points) *(NOTE: These extra credit points will only take effect if you achieve a perfect score on the rest of the assignment.)*

[3] Here we will assume either that the adversary has no control over the network hardware, or that the problem of the previous question has been fixed.