Name: Megan Kanne
SUNet ID: mkanne

Description:
My proxy server is implemented such that it intercepts a request from a client over SSL and makes it's own forwarded SSL request to the remote server. It then receives the response and parses it for values in the certificate for the remote server. Specifically, it gets the SubjectDN and serial number of the remote server. It then gets its own privateKey, publicKey, and IssuerDN from a certificate in a keystore that it has self-signed. It generates a new certificate from this information with the remote server's serial number, its public key, its IssuerDN, and the remote server's SubjectDN. Finally, it signs this cert with its private key. When it forwards the response from the remote server onto the client, this new cert is the one included. Both the connection to the client and the connection to the remote server are over SSL.
The admin program first sets up an SSL connection to the proxy server, then authenticates. Authentication is via password from the program arguments. To authenticate, the password is compared to a hash of the correct password that was previously generated and stored in pwdFile. The hash was generated using BCrypt which implements a secure key derivation function (PRF) using a derivation of the Blowfish cipher. The comparison uses a library function which is secure. The proxy server then does the admin command if the user is authenticated or closes the socket if not.

Steps to use:
Comment: I was able to compile and run on myth, but not test as I could never get firefox on myth to use the proxy. I ran and tested locally and my log.txt file comes from those local tests.
1) make
2) run proxy server using one of the two below:
    - java -classpath :.:iaik_jce.jar mitm.MITMProxyServer -keyStore keystore -outputFile log.txt -keyStoreAlias mykey -keyStorePassword UF2QPy5A
    - java -classpath :.:iaik_jce.jar mitm.MITMProxyServer -keyStore keystore -outputFile log.txt -keyStoreAlias mykey -pwdFile passwords.txt
3) in firefox preferences, set SSL proxy to "localhost" and 8001
4) go to any page using https and see the "This Connection is Untrusted" message
5) run admin interface using one of the following to shutdown or get stats:
    - java -classpath :.:iaik_jce.jar mitm.MITMAdminClient -password 5fFRhML4 -cmd shutdown
    - java -classpath :.:iaik_jce.jar mitm.MITMAdminClient -password 5fFRhML4 -cmd stats
6) Ctrl+c the server to end (or use shutdown from admin interface)

Questions:
1) I'm not entirely sure what is question is assuming: either an attacker can intercept messages or they can read them:

- If an attacker could intercept and *read* packets, they could watch the network for a password submission from a real user of the admin server and an authenticated response from the proxy server. Because the admin server sends the password to the proxy server as plain text and the proxy server sends back its response as plain text, the attacker will easily be able to discover the password for full access to the admin server (This assumes traffic is not encrypted). The obvious way to solve this is to encrypt the messages between the two servers. Similar to our solution in project 1, we might encrypt the messages using AES in CBC mode and decrypt the same way. Both servers would need to have a private key that they share. Again, similar to Project 1, we might store the keys in an database that we also encrypt with AES in CBC mode using a key private to each server. As in Project 1, we must also attach a MAC tag to the message so that a network attacker couldn't change the encrypted packet to break the scheme. One MAC solution might be CBC-MAC.
- If an attacker can only intercept and redirect packets, they could set up an SSL connection with the server and try resending those intercepted packets from their own program to the proxy server. If the packets include a valid login request, the attacker will be authenticated and get access. This assumes the attacker can issue an SSL cert to the proxy server so it can decrypt the traffic it captures or that it can otherwise spoof a session connection to the proxy server that will allow the proxy to decrypt the packets it sends. If this is possible, I'm not entirely sure how to protect against it other than to try to have the proxy server somehow detect spoofed SSL certs.

2) Security considerations:

- For (a), if the attacker can read the file pwdFile, they can see the hash of the password that was generated by BCrypt. However, the BCrypt library uses the a key derivation function that is a secure PRF for hashing based on the Blowfish cipher. Because the hash provides the attacker no information about the input (because it is a secure PRF), they must guess the password non-trivially. It also uses a salt to protect against table attacks. We prove by contrapositive. For function F' which is the bcrypt implementation of password hashing, if an attacker could distinguish the hash in the file from random, then for a challenger where b=0 returns f<-F'(password) and b=1 returns f<-rand, they output b'. Now we could create an adversary that forwards requests from the attacker onto a different challenger which for b=0 returns F(password) and for b=1 returns f<-rand, where F is the Eksblowfish cipher. Now the adversary intercepts a query from the attacker and performs the bcrypt hashing of query by querying the challenger for Eksblowfish cipher results until they have completed the function. They then forward this onto the attacker who can distinguish it from random. Because the BCrypt PRF (Eksblowfish) is proven to be secure, that means that our hashing is secure.
- For (b), if an attacker can read and/or write to the file pwdFile between admin server invocations, then our sever is no longer secure. This is because the BCrypt library is available to them as well. So they could easily generate a valid hash of some password they choose. They would then write that hash to

the admin server. Now when they attempt to login with the password they chose, the server will get the compromised hash from the pwdFile and compare it to the password input using the BCrypt checkpw() function. Because the written hash is a valid hash for the password, this check will return true and the attacker has won!

- To make my admin server secure under (b) we might construct a digital signature for the pwdFile then have the admin server verify that signature each time before using the file. If the file was tampered with (verification fails), we close the socket. One solution would be to use a Lamport signature which is a one-time signature built from a one-way function f. We would create a Lamport key pair of a private and public key. We would then, one time only, sign the file (after it has been created and the password hashed) using the Lamport signature signing scheme. On all future server runs, we would use Lamport verification to verify that the file is not corrupted. In order to forge a message an attacker would have to invert the one-way function f. This is assumed to be intractable for suitably sized inputs and outputs.

3) To make it less likely that a web browser end user could be fooled by MITM attacks we could do a few things. First, we could just disallow them from proceeding it we see a DN, serial number, issuer, or public key mismatch. Or we could allow them to continue, but make it very clear that their browsing is unsafe and they should not input any sensitive information. For example, we could do something graphical like outline the entire web view in red or make the URL bar red with a big red unlocked lock. We could also periodically remind the user that they are browsing unsafely with some sort of popup. Another thing we might be able to do is implement a sort of popover anytime the user inputs and information into a form. This popover again warns them that they are browsing insecurely and should not submit the form. Or we may just disallow form submits altogether (GET requests with parameter data or any type of POST).
4) Didn't have time to test it out.