CS255 Project 2 Report

Lynnelle Ye (lynnelle@stanford.edu)
Frank Chen (frankchn@stanford.edu)

Design Choice Description

Our design choices were very standard. We modified all the files that were highlighted as relevant within the assignment description.

We first generate our public and private key pair with the RSA algorithm. We tried the DSA algorithm but we found that the signing algorithm was not deterministic, causing our certificates to be rejected by Firefox because it had detected certificates with different signatures but the same serial number, which Firefox rightly flags and rejects. Switching to the RSA algorithm solves this problem.

In forging certificates on the fly, we did the following steps:

1. Created a new X509 certificate using the IAIK library
2. Set the creation and expiry dates to be the same
3. Set the SubjectDN, IssuerDN and serial number to be the correct values
4. Set the public key to be our generated public key
5. Self-sign the certificate using our private key
6. Generate a certificate chain of length 1 consisting of only our certificate
7. Initializing relevant libraries and passing everything back

We also chose to cache the certificates in a HashMap to speed up the proxy server.

For the admin server, we chose to implement salted multi-round bcrypt under the SHA-1 hash function to keep the administration passwords relatively secure against attackers who were able to read our password files from the server.

Extra Credit: We implemented an option to connect to the admin server using MAC-based challenge/response rather than direct password authentication. When this option is turned on, the server sends a random 130-bit challenge to the client. The client is expected to use their password to apply HMAC-SHA256 to the challenge message and respond with the resulting MAC. The server keeps the correct password stored in plaintext, with which it computes the correct MAC for the challenge and checks whether it equals the client's response.

If the server is set up with this authentication system, a legitimate client is protected against replay attacks by eavesdroppers on its traffic with the server, since as long as HMAC-SHA256 is secure, answers to past challenges will give eavesdroppers no help in answering future ones.

(According to the birthday paradox calculation, we expect to see the same challenge repeated after 2^65 authentications, which is large enough to be safe.) Unfortunately, the server is forced to store enough data to compute the correct response in plaintext, so we can no longer protect against an attacker capable of reading the server's password database.

We use the bouncycastle library to provide some base64 encoding and decoding functionality for the extra credit portion of the assignment.

Sequence of Steps to Run

We have included a run.sh to run our server on Mac OS X / Linux based machines. We have also included a runclient.sh to run our client to communicate with the server. The command sent to the server is embedded in runclient.sh.

Extra Credit: We have included a run-mac.sh and runclient-mac.sh to run the server and client with the MAC administrative interface respectively. We note that the password file is stored as pwdfileMAC in the server and this has to be kept secret as the password is stored in plaintext (per the Challenge-Response MAC section of the ID protocol lecture).

Short Answer 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.*

   In this case, an active network attacker could simply modify the packets from the server to the client and vice versa. This will simply enable the attacker to (a) sniff the administrative password from the server to client and (b) rewrite any commands sent from a legitimate client to the server.

   To fix this problem, we should establish a secure connection by implementing TLS and embedding relevant certificates for the server and the client in the software itself. This will prevent network attackers (who do not have access to the binary with relevant encryption keys) from reading and tampering with any network packets. Malformed network packets will simply be dropped by the system.

2. *Suppose an attacker is trying to gain unauthorized access to your MITM server by making its own queries to the admin interface. 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 (i.e. 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)?

We are secure against attacker A because our passwords are salted and hashed 10,000 times with Bcrypt under SHA1, which prevents rainbow table based attacks. Attackers would have to do a significant amount of work in order to break the system.

We are not secure against attacker B. We will have to sign the password file with a private key and embed the corresponding public key in the server and verify that the password file has not been tampered with before loading it. However, we note that if attackers can both read from and write to password files, then we may not be able to assume that the binary or execution environment is safe. In those cases, it would be impossible to defend against a determined attacker.

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.)*

   We can use certificate pinning as described in class. Websites can request for their SSL certificates to be saved when a new user first visits the website. Different certificates from that point on will be automatically rejected.

4. *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?*

   Our MITM implementation does not always work as these servers would communicate with other domains over SSL (e.g. https://www.coursera.org communicates with https://dt5zaw6a98blc.cloudfront.net for a lot of javascript code). As we did not see or is able to confirm security exceptions for these other domains, dynamic web pages will often fail to load or load incompletely.

   We can intercept requests to the primary domain and rewrite https:// links to http:// links so that they do not need to go through our proxy and thus the pages would load correctly.