|  |  |  |  |
| --- | --- | --- | --- |
|  | 25th March 2013 |  | |
|  | |  | |
| Computer & Network Security *UFCEMV-20-3*  Coursework B1 | | | |
|  |  | |  |
|  |  | | David Norton – 10005864 Felix Burr – 11018689 |

Computer & Network Security Assignment

# Analysis of the basic version of the Encryption Server System

A few of the current systems vulnerabilities include:

* All traffic sent between the client and server is unencrypted. This makes life very easy for a man in the middle attack.
* Passwords are stored in plaintext so can be easily viewed and stolen.
* There is no login or authentication system for clients, so anyone can access the server.
* No certificate authentication so neither can prove they are legitimate.

# Implement (trivial) password-based authentication (Part ii)

From the code implemented, the issue of an authentication process has been solved, although trivially. The password system is still weak, especially as the file holding usernames and passwords is still in plaintext. All traffic is still being sent insecurely, so the client details can be viewed as plaintext when sent. So remaining issues are:

* Unencrypted traffic between client and server.
* Client details stored in plaintext.
* No certificate authentication so neither can prove they are legitimate.

# Improving the password-based authentication (Part ii)

From the new password–based authentication code, although the rest of the traffic being sent is unencrypted, the password is now sent using a MD5 hash making it more secure. However, the client details are still being held in a plain text file. So remaining issues include:

* Client details stored in plaintext.
* Unencrypted traffic between client and server, although password is now hashed.
* No certificate authentication so neither can prove they are legitimate.

# Symmetric Encryption and Key Exchange (Part ii)

With Diffie-Hellman implemented, the client and server now have a secure connection as they are able to share a secret key. However, this is currently only set for transferring the file, so although this is now secure, the rest of the communication is not. So remaining issues include:

* Client details stored in plaintext.
* Unencrypted traffic between client and server, although password and file have been secured.
* No certificate authentication so neither can prove they are legitimate.

# Setting up Public Key Infrastructure (Part I & Part II)

See screenshots in attached folder too view how these tasks were completed.

# SSL based communication and signed certificate

**Problems Encountered**

1. Initially we struggled connecting and communicating with the CA.
2. Differences in undertaking this task lead to a minor warning on 11018689 code producing different errors to 10005864.
3. Connecting both ES & CL with an SSL connection.

**Problems Fixed**

1. It took us a good part of a day to realise that CansCA had a main method and should run simultaneously with the server and client to initiate connection to the CA rendering our first set of errors obsolete.
2. The way we resolved 11018689s warnings was by comparing code and noticing trial code in CAListener was overwriting the base 64 sections. Simply rectifying and taking out the line of code enabled our group to continue work ethic and progress with the tasks.
3. We were able to use the SSL connection. When attempted to connect via SSL we got the following error:

Exception in thread "main" java.net.ConnectException: Connection refused: connect

We believe was due to it not being set up correctly on the server side. However, in CansCA when changing from listen(); to listen\_SSL(); and making changes we still receive the same error. The certificate is saved correctly in the path “src/Files/signed.crt “.

Within an hour of the deadline we were finally able to solve this issue through several minor changes. However, the code became slightly messy, so we kept a copy of the old version to refer back to.

# Authentication via Certificates (Part II)

With the CA fully implemented and communications transferred over SSL, the client can now communicate securely with both the CA and the server. Unlike with Diffie-Hellman, all traffic is secured rather than just the file. The user not being required to enter a username and password also removes the weakness of having the user details stored in plain text, and stops these details being stolen while in transit.

The only security issue that we can now think of is the theft of certificates would allow anyone to access the server without the need for any username / password, potentially making the system weaker, although these details being discarded make the system more secure in another way as mentioned above.

# Overall assessment of the security of our ES

Overall the following security measures have been added to the system:

* Username / Password system – This created some level of security so only those who knew the login credentials could access the system.
* Password stored as MD5 hash – This helped protect against man in the middle attacks.
* Diffie-Hellman – This helped to secure the file as it was sent between the server and client.
* SSL – This allows all data transferred between the client and server to be secure, preventing man in the middle attacks.
* Use of certificates – This helps prevent against unauthorized user login and proves man in the middle attacks useless as they still need the correct certificate to access the server.

In in all the combination of these security additions makes the connection between the client and server very secure against man in the middle attacks and prevents unauthorised users from logging into the server.

I assume several weaknesses exist as these stages have been implemented one at a time, rather that considering them all at once. The order in which these stages are carried out could provide gaps in security that could be taken advantage of. There is also the weaknesses in the security features themselves, for instance many are beginning to say that SSL has its weaknesses so data transfer is not fully secure.

# Other

Any passwords are set as: password

* For running tasks 1-4, simply run the server (EncryptionServer.java) then run the client (FileClient.java).
* For task 5iii run the file SignCertificate.java in the uk.ac.uwe.cans.util folder.
* For tasks 6-7 run the server (EncryptionServer.java), then the CA (CansCA), then run the client (FileClient.java).

All work was completed as a pair with equal contributions.

# References

1. Apache (2013) *The client-side of Diffie-Hellman key agreement for Kerberos*. Available from: <http://svn.apache.org/repos/asf/directory/sandbox/erodriguez/kerberos-pkinit/src/main/java/org/apache/directory/server/kerberos/pkinit/DhClient.java> [Accessed 25 March 2013].
2. Apache (2013) *SSLSocketFactory constructor throwing java.lang.IllegalStateException: SSLContextImpl is not initialized.* Available from: <https://issues.apache.org/jira/browse/HTTPCLIENT-1211> [Accessed 26 March 2013].
3. Oracle (2013) *JavaTM Secure Socket Extension (JSSE) Reference Guide.* Available from: <http://docs.oracle.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html> [Accessed 26 March 2013].