1 Protocol Analysis

You have learnt that someone is using their own cryptographic protocols to send messages. You need to analyse the protocols, and find flaws in them that will let you read the messages without the client key. You can log into the Alice account: aliceBlack:aliceGHdj%*3, and here you will find the source code for the protocols servers.

These protocols are running on port 11337 and port 11338 on the VM. You must implement your attacks against the VM and find the secret messages, which include tokens.

2 Protocol 1

In this protocol, a client $C$ and server $S$ share a symmetric key $K_{cs}$, this key is only known to the server and client. This long term key is used to set up a session key, and this session key is then used by the server to send a secret value to the client:

1. $C \rightarrow S$: “Connect Protocol 1”
2. $S \rightarrow C$: $\{N_s\}_{K_{cs}}$
3. $C \rightarrow S$: $\{N_c\}_{K_{cs}}$
4. $S \rightarrow C$: $\{N_c, N_s\}_{(N_s \oplus N_c)}$
5. $C \rightarrow S$: $\{N_s, N_c\}_{(N_s \oplus N_c)}$
6. $S \rightarrow C$: $\{\text{Secret Value}\}_{(N_s \oplus N_c)}$

The client starts a run of the protocol by sending the bytes of the ASCII for “Connect Protocol 1” to the server. The server then generates a nonce and sends it to the client encrypted with the key $K_{cs}$. The client must reply with a challenge of it’s own for the server: the nonce $N_c$ encrypted with the key $K_{cs}$. The session key is the xor of the two nonces. The encryption used is 128-bit AES in ECB mode with PKCS5 padding and the nonces are 128 bits.

The idea of this protocol is that only the server and the client know the key $K_{cs}$, so only they know the nonces, which in turn should mean that only the client and server can know the session key. Step 4 and 5 let the client and the server prove to each other that they know the key, with the aim of providing mutual belief in the session key $(N_s \oplus N_c)$. Unfortunately this protocol has a security flaw and does not achieve these aims.
2.1 The Exercise, Part 1

1. Analyse this protocol and find an attack that will let you learn the secret value from the server without having to know the key $K_{cs}$. Implement your attack and run it against the server running on the VM. The message will include a token, submit this token to the token submission site and submit your attack code to Canvas.

I recommend writing your attack code in Java and using the server code as a model. However you can write it in any language you want, as long as it will (compile) and run on the VM without additional library or other installations. N.B. If you code does not compile and run correctly on the VM you will not receive marks for this exercise (no marks for being “close”).

[4 marks]

3 Protocol 2

In this protocol, as above, a client $C$ and a server $S$ share a symmetric key $K_{cs}$, this key is only known to the server and client. This long term key is used to set up a session key, this session key is then used by the server to send a secret value to the client:

1. $C \rightarrow S : g^x$
2. $S \rightarrow C : g^y$
3. $C \rightarrow S : \{N_c\}^{g^{xy}}$
4. $S \rightarrow C : \{\{N_c + 1\}_{K_{cs}}, N_s\}^{g^{xy}}$
5. $C \rightarrow S : \{\{N_s + 1\}_{K_{cs}}\}^{g^{xy}}$
6. $S \rightarrow C : \{\text{Secret Value}\}^{g^{xy}}$

In this protocol, the client and the server use Diffie-Hellman to set up a key based on $g^{xy}$. They check who is on the other end of this channel by exchanging nonce challenges. The idea here is that, given that only the server and the client know the key $K_{cs}$, then, given the challenge $N_c$, only the server can produce $\{N_c + 1\}_{K_{cs}}$ and given the challenge $N_s$, only the client can produce $\{N_s + 1\}_{K_{cs}}$. However the protocol is flawed and an attacker can learn the secret value without knowing the key $K_{cs}$.

The encryption used is 128-bit AES in ECB mode with PKCS5 padding and the nonces are ints. $g^x$ and $g^y$ are sent as public key certificates; as these get vary in length, the length of the certificates are sent as an int before the certificates. Only the first 128-bits of $g^{xy}$ are used to make the AES key. You can find the code for the server (minus the key and the secretValue) in the Alice account. The values of $p$ and $g$ for Diffie-Hellman can be found in the server code.

The idea here is that only the server and the client know the key $K_{cs}$ so only they know the nonces which in turn should mean that only the client and server can know the session key. Step 4 and 5 let the client and the server prove to each other that they know the key, with the aim of providing mutual belief in the key. Unfortunately this protocol has a security flaw and does not achieve these aims.
3.1 The Exercise, Part 2

1. Analyse this protocol and find an attack that will let you learn the secret message from the server without having to know the key $K_{cs}$ (N.B. as you cannot observe any traffic from the client therefore this cannot be a man in the middle attack).

Implement your attack in Java and run it against the server running on the VM. I recommend looking at the server code to understand how to write your attack. The secret message will include the token that you must submit to the token submission website.

[4 marks]

4 Submission

Submit the tokens to the token submission website and your attack code to Canvas.

Getting Help

For the protocol analysis part of this exercise, I strongly recommend reading the paper: “Prudent Engineering Practice for Cryptographic Protocols” by Martin Abadi and Roger Needham (http://www.cs.bham.ac.uk/~tpc/cwi/Teaching/Papers/11principles.pdf). I recommend looking at the server code in detail and making sure that you understand every line before writing your attack code. You may also find it helpful to try writing your own client and run the server code on your own machine, with a key and secret value you know. You can also message to the Facebook group and come to my office hour, with questions and to get extra help with the exercise.