Lab 4: Basic Protocols

Due: 10pm Friday, May 9, 2014

Challenge Problems

Problem 1: Diffie-Hellman Key Exchange

There is a server at 192.168.14.40 on port 4001. To retrieve its flag, you must first perform a Diffie-Hellman key exchange with the server to derive a shared secret key. You can then use the shared secret key to send an encrypted request ("GET FLAG") and decrypt the server's encrypted response (the plaintext will be "FLAG *flag*").

For a review of the Diffie-Hellman key exchange, see Mark Loiseau's tutorial (http://blog.markloiseau.com/2013/01/diffie-hellman-tutorial-in-python/). Here, our server uses the prime p = 999,959 and the generator g = 2.

Note: In order to make this exercise possible without importing additional libraries, we're using small numbers that are **NOT** safe for use in the real world. (Using g = 2 is OK. In practice, p should be **much** larger.)

The protocol works as follows:

1. When you connect, the server immediately sends you its public key: "PUBKEY *pubkey*", where *pubkey* is a standard decimal ASCII-encoded integer. If the server's private key is *s*, then the server's public key is equal to g^s mod p.

2. You should reply with your public key in the same format: "PUBKEY *pubkey*". If your private key is *c*, then you can compute your public key as g^c mod p.

3. Both sides then derive the session key as MD5(g^(sc) mod p). (Note: The PyCrypto MD5 implementation wants a string for its input, so be sure to call str() on your integer g^(sc) before passing it to MD5.)

4. You can now encrypt your request ("GET FLAG") with AES-128 in CBC mode using the session key. You send your encrypted request to the server as raw binary bytes.

5. The server attempts to decrypt your request, and if successful, it sends back a response ("FLAG *flag*") encrypted with the same key in the same way.

Question: What's wrong with this scheme? How many ways can you find to attack it?

Problem 2: Diffie-Hellman MITM Attack

There is a server at 192.168.14.40 on port 4002. To retrieve its flag, clients must first perform a Diffie-Hellman key exchange with the server to derive a shared secret key. The client can then use the shared secret key to send an encrypted request ("GET FLAG *cookie*") and decrypt the server's encrypted response (the plaintext will be "FLAG *flag*").

Note that this server only provides the flag if a valid cookie is included in the request.

Some clients cannot connect to this server directly, so they will attempt to connect to your IP address on port 4202. Your job is to proxy their connections to the server and perform a man-in-the-middle attack to extract the flag.

Problem 3: Needham-Scroeder Protocol

In this problem, you must use a simplified version of the Needham-Schroeder protocol to authenticate yourself to Bob. After you have authenticated, you can ask Bob for the flag ("GET FLAG") and he will send it to you ("FLAG *flag*").

Bob's address is 192.168.14.20, and he's listening on port 4333. Your trusted third party, the server S, is at 192.168.14.40, listening on port 4003.

The protocol proceeds as follows: (Here, {xyz}_K means that xyz is encrypted with key K.)
1. A sends to S: A, B, Na
2. S sends to A: {Na, Kab, B, {Kab, A}_Kbs }_Kas
3. A sends to B: {Kab, A}_Kbs
4. B sends to A: CHALLENGE {Nb}_Kab
5. A sends to B: RESPONSE {Nb - 1}_Kab
6. A sends to B: GET FLAG
7. B sends to A: FLAG *flag*

The server knows you as "student" and knows Bob as "bob". So for Message 1, you might send "student bob 1234". Your secret key which you share with the server is (in hex) 20 14 04 03 20 14 04 03 20 14 04 03 20 14 04 03. All encryption is performed using AES-128 in CBC mode with PKCS7 padding (http://tools.ietf.org/html/rfc5652#section-6.3).

To simplify debugging, all binary data (keys, ciphertext, etc.) are encoded in hexadecimal ASCII in the messages. So, for example, you can print Kab exactly as you receive it. But before using it to encrypt anything, you must transform it back to its native binary format using binascii.a2b_hex().

Problem 4: Attacking the Needham-Scroeder Protocol

For this problem, you can use the same trusted third party as in Problem 3. This time, your job is to get the flag from Charlie.

Unfortunately, Charlie does not want to give you the flag. However, he *is* willing to share the flag with Bob.

Charlie is listening on TCP port 4004 at 192.168.14.30.

To help with your attack, you can look at a pcap file containing traces of a session between Bob and Charlie using an older version of the program with a critical security flaw. The pcap file has been posted on the testbed network at http://192.168.14.10:8000/bc.pcap

Hint: Use the attack that we discussed in class. The pcap file should give you all the information you need.