Homework 4
CS 642: Information Security

November 23, 2014

This homework assignment covers topics in cryptography. ***You may work with a partner***. It is due **Dec 5, 2014** by midnight local time.

The deliverable should be a nicely formated hw4.txt or hw4.pdf file, in addition to an executable script "recover-plaintext" for part 2 of problem 2. Turn these in via Moodle.

Each part of each problem is worth 2 points, so you have the opportunity to get partial credit for each. Make answers concise (you will not get credit for rambling and long answer that happens to contain some correct portions).

1 Problem 1

In Diffie-Hellman key exchange, a group $(G, \cdot)$ is fixed along with a generator $g \in G$. Here "$\cdot$" represents a group operation such as multiplication modulo a large prime. The two participants pick exponents x and y randomly from $\mathbb{Z}_{|G|}$ and exchange values $X = g^x$ and $Y = g^y$. They can then compute the shared secret g^{xy}. Suppose the shared secret is instead computed as $X \cdot Y$. Is this secure? Explain why or why not.

2 Problem 2

A colleague has built a password hashing mechanism. It applies SHA-256 to a string of the form "username,password,salt" where salt is a randomly chosen value. For example, the stored value for username "user" and password "12345" and salt "999999" is

0x873b8b6a77af4bb6cee4cae09eaa81b27556c7cd9786e754a169114b6d3674d5

For example, the Perl code to generate this is:

```
#!/bin/usr/perl

use Digest::SHA qw(sha256_hex);

printf sha256_hex( "username,12345,999999") ;
```

or in one line:

```
perl -e 'use Digest::SHA qw(sha256_hex); print sha256_hex("username,12345,999999");'
```

The same process was used to generate the hash

0x37448ba7de7f5b4396697edaeddcd7bc840964e6ce82016915b830a91d69eb2f

for user "ristenpart" and salt "134153169".

1. Recover the password used to generate the second hash above. Hint: The password consists only of numbers.
2. Give a pseudocode description of your algorithm and the worst case running time for it.
3. Discuss the merits of your colleague's proposal. Suggest how your attack might be made intractable.

3 Problem 3

Another colleague decided to build a symmetric encryption scheme. Included on the website is a tarball of the encryption and decryption source code in Python, as well as a test key file and a challenge ciphertext. As you will see, your colleague used an Encrypt-then-MAC scheme, with a simple padding rule like the one used in TLS's record layer. Your job is to asses its security.

1. Give a pseudocode descryption of the encryption and decryption algorithms in the Python scripts.
2. In a plaintext recovery attack, the attacker desires to recover the message encrypted within a ciphertext. You should write a program or script that takes as input a ciphertext output by `badencrypt.py` for some key file and retrieves the first 16 bytes of message underlying the ciphertext. (We will check only the first 16 bytes of output to match the target plaintext, but feel free to recover the entire message.) It should do so quickly; we will cut it off if it is taking far too long. My reference implementation takes about 50 seconds.

   Your attack script can make calls to `baddecrypt.py` in the same working directory and that will use the same key file used to generate the ciphertext input to it. In Python, for example, one could use the following code snippet:

   ```
   from subprocess import Popen, PIPE
   ...
   proc = Popen(["./baddecrypt.py",keyfile,hexCiphertext],stdout=PIPE)
   output = proc.communicate()[0]
   ```

   Your attack script will not have direct access to the key file and should not attempt to gain access to the process memory of baddecrypt or any other files to steal the key directly.

   Your attack script will be tested by running it on one of the CSL machines in the same working directory as baddecrypt.py with the key used to encrypt the challenge ciphertext given in the tarball on the homework webpage. We may end up testing the script with other ciphertexts, however.
3. Give pseudocode for a new implementation that fixes the vulnerability that allows the plaintext recovery above. The decryption algorithm must still give the same kinds of error and success messages (i.e., you can't just remove all outputs!).