UNIVERSITY OF MARYLAND
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

ENEE 459C
Computer Security
Instructor: Charalampos Papamanthou

Homework 4

<u>Out</u>: 11/07/14 <u>Due</u>: 11/21/14 11:59pm

Instructions

1. Strictly adhere to the University of Maryland Code of Academic Integrity.
2. **Type** and submit your solutions as a pdf document at Canvas. Include your full name in the solutions document. Name the solutions document as x-hw4.pdf, where x is your last name.
3. No extensions will be given. Everything will be due at 11.59pm on 11/18/14.

Problem 1 (20 points)

Read the paper on Tor and briefly answer the following questions:

1. What are the main contributions of this paper?
2. What parts of the paper do you find unclear?
3. What parts of the paper are questionable, i.e. you think a conclusion may be wrong, an approach or evaluation technically flawed, or data ill-presented?
4. Sketch an attack on the system other than those presented in the paper. Devise this by yourself if possible, but cite your source(s) if you choose to google.
5. For the attack you devise, discuss:
   - How to evaluate its effectiveness.
   - What countermeasures could defend against it.
   - How much impact it could have in practice (consider what kind of people use Tor and what kind of people wish to break it).

Problem 2 (20 points)

Suppose you know that the passwords used by the students to authenticate to the university server consist of n uppercase letters of the English alphabet. Given the hash $h(p)$ of a password p describe three solutions to compute password p.

1. The first solution uses constant space, $O(26^n)$ query time and no preprocessing.
2. The second solution uses $O(26^n)$ space, constant query time and $O(26^n)$ preprocessing time.
3. The third solution uses $O(26^{\frac{2n}{3}})$ space, $O(26^{\frac{2n}{3}})$ query time and $O(26^n)$ preprocessing time.

Because of the above attacks, the administrator changes the way passwords are stored: Instead of storing $h(p)$ for password p, the administrator stores $(h(p||r), r)$, where r is some randomness that is chosen for each password. Which one of the above solutions will not work now? Why?

Problem 3 (20 points) (originally developed by Jonathan Katz)

Assume the following scheme is being used to hash passwords: An n-bit password p is padded to the left with $128 - n$ zeros and used as an AES-128 key to encrypt the all-0 plaintext; the result is the "hashed password", i.e.

$$h(p) = \mathsf{AES}_{0_{128-n}||p}(0^{128}).$$

So for the 12-bit password $p = \mathsf{0xABC}$, the result should be

$$h(p) = \mathsf{970fc16e71b75463abafb3f8be939d1c}.$$

You may assume n is a multiple of 4.

The scenario is that you are given $h(p)$ and n and need to recover p. Your attack should use a rainbow table with $2^{n/2}$ chains of $2^{n/2}$ length each. You may wish to visit this page for an example of a reduction function that you can use.

1. Write two programs, called GenTable and Crack. The first of these corresponds to the pre-processing phase in which you generate a rainbow table, while the second corresponds to the on-line phase in which you are given $h(p)$ and need to recover p.
   - GenTable should take one command-line argument and generate output to a file rainbow. The argument will be n, the password length (in bits). The bound on the size of rainbow must be no larger than $3 \cdot 128 \cdot 2^{\frac{n}{2}}$ bits. Failure to meet this space bound will result in 0 points. You can use `ls -l` to check the size of your file.
   - Crack should take two command-line arguments and generate output to standard output. The first command-line argument is the same as above. The second argument is $h(p)$ in hex. When you run Crack n $h(p)$, you may assume that GenTable n was just run to give rainbow. The output of crack should include two items: the password p or "failure", and the number of times AES was evaluated. Failure to report the number of AES evaluations accurately will be considered cheating, and will result in 0 points.
   - So running
     GenTable 12
     Crack 12 970fc16e71b75463abafb3f8be939d1c
     should give output "password is ABC, AES evaluated 191 times" (assuming that in this execution of crack AES was evaluated 191 times).
2. Include a 1-2-page writeup that describes your implementation.
3. Use your programs to recover the passwords from the challenges here:

   $$http://enee459c.github.io/homeworks/code/rainbow.txt$$

   Include the answers at the end of your writeup.

Submit your source code for your programs and your writeup.

Problem 4 (20 points)

Given the confidentiality categories TOPSECRET, SECRET, CONFIDENTIAL, UNCLASSIFIED and the integrity categories HIGH, MEDIUM, LOW, indicate and explain what type of access (read, write, both, or neither) is allowed in the following situations.

1. Paul, cleared for (TOPSECRET, MEDIUM), wants to access a document classified (SECRET, HIGH).

2. Anna, cleared for (CONFIDENTIAL, HIGH), wants to access a document classified (CONFIDENTIAL, LOW).
3. Jesse, cleared for (SECRET, MEDIUM), wants to access a document classified (CONFIDENTIAL, LOW).
4. Sammi, cleared for (TOPSECRET, LOW), wants to access a document classified (CONFIDENTIAL, LOW).
5. John, cleared for (SECRET, LOW), wants to access a document classified (SECRET, LOW).
6. Robin, who has no clearances (and so works at the (UNCLASSIFIED, LOW) level), wants to access a document classified (CONFIDENTIAL, HIGH).

Problem 5 (20 points)

In this programming assignment, you are going to exploit the account of user chell at a server running at IP 129.2.212.154, by writing into chell's files without having explicit write permission. You can log into the server by using the account wheatley@129.2.212.154 with password I'mamoron. In order for multiple users not to interfere with each other despite all being logged in as the same user, the home directory of wheatley is read-only. If you want to write programs or save files or anything, you can create a directory in /tmp and save your files there by using

```
mkdir /tmp/some_unique_directory_name
```

You will exploit a TOCTOU (time of check-time of use) race condition in a setuid program owned by user chell (as we discussed in class). Once you log in as wheatley@129.2.212.154 you'll see a program you can run, i.e., the program /home/wheatley/append. Running

```
/home/wheatley/append string filename
```

will add a new line containing `string` to the end of the file named `filename`. The append program is setuid chell, so it runs with the permissions of the chell user. The source code to the append program is in /home/wheatley/append.c. Exploit a vulnerability in the append program to append your name to the file /home/chell/list. For 15 bonus points, read the contents of the file /home/chell/file.txt.

You can find information on race conditions and techniques to exploit them here. For a walk-through of how to exploit the specific kind of access/open race seen in the append program, see here and here.