Recall the TENEX password guessing vulnerability that used a side-channel attack (based on the memory protection of the buffer holding the input password string). For this assignment you will replicate that attack on Unix/Linux as well as develop a new attack in which the side channel is time instead of memory protection.

You can develop this project on any Linux machine (including in a VM) but you should test on ieng6.ucsd.edu which is where we plan to test.

Download proj1.tar off the course Web page. You can use the command:
```
wget http://www-cse.ucsd.edu/classes/fa15/cse127-a/proj1.tar
```
Then “untar” it into your home directory. You can use the command:
```
tar xf proj1.tar
```
You will find several files there:

- **Makefile**: makes timehack and memhack
- **memhack.c**: This is where you should add code for the TENEX-style memory protection side channel.
- **timehack.c**: This is where you should add code for the time-based side channel.
- **sysapp.{c,h}**: these define a simple library interact to the assignment. The check_pass() function checks if your password is valid. The hack_system() function should be used once you believe you have successfully guessed the password (this is the function we will use for grading)

The only files you should change are memhack.c and timehack.c. Please do not change the Makefile or add any additional files.

Here is our advice for how to do this assignment.

Start with the memhack problem because it is deterministic and doesn’t have problems with noise. Look at the check_pass() function and see the password string is passed by reference and how the memory it points to is checked against the reference password one character at a time.

Now look in memhack.c and see how a buffer is allocated such that the page starting at page_start is protected (i.e., accessing it will caused a segmentation fault, or SEGV) and the previous 32 characters are allocated. Now look at the demonstration function demonstrate_signals() which shows how referencing any memory in the protected page will produce a fault and how to catch that fault in your program. You do not need to use
this function, it is merely there to show you how to use signals to capture whether a memory reference touched a page or not.

Now you’ll want to create a framework (in memhack.c) to call check_pass with different inputs and catching any resulting faults so you can determine if part of the password is correct. We suggest a loop over the maximum password size (32 characters) where for the first guess you store the password such that its first character is one byte before page_start and then iterate between possible choices for the first character (when you get it right you’ll get a page fault). Repeat as needed to guess the entire password. Note that all ASCII symbols from ASCII 33 (“!”) to ASCII 126 (“~”) can be used in the password.

Ok, now go work on the timehack problem. Once again go back and look at check_pass(). An artificial delay has been added when each character is checked to make your life easier (it is possible to do this assignment without it but would require much more careful methodology… leave the delay there). Thus, the execution time of check_pass() will depend on how many characters you guess correctly.

Now go look in timehack.c and find a macro there for a “function” called rdtsc() which invokes the processor’s cycle counter (a clock that increments by one for each processor cycle that passes). In general treat rdtsc() as a free running timer that returns a long. Insert a call to rdtsc() before the call to check_pass() and the call afterwards. Print the difference between these values. Run the program a few times. Now change the guess string so the first character is correct. Run again and see how the time difference changes.

Now automate this entire process, in the style of the original memhack approach. Note that unlike the memhack attack, the timehack problem will have to deal with noise. Depending on things like what other programs are running, the status of the cache, the contents of the branch target buffer, etc… there can be significant variability in the amount of time each check takes. This WILL matter in practice. Thus you will want to run a lot of trials for each test before you reach your conclusion about each character.

Other hints:

1) Be careful in using lots of printf’s. These can blow out the instruction cache and data caches and perturb your results (i.e. overwhelm the timing effects you are trying to detect).
2) Be careful in averaging across trials. If your process is descheduled in the middle of a measurement the time can be so large that it overwhelms everything else. Instead, the median is your friend.
3) If time is not continuing to increase, then you probably made a bad guess earlier. Backtrack.
4) Our advice for debugging: make a big array to hold your timing measurements and print them out at the end.
5) Testing. Be sure to test a bunch of different passwords. We will when we grade.

Do not write a solution that simply checks all passwords exhaustively. You will not get
credit for this. This should be doable in linear time (we will stop programs that are running for excessive periods) and it will basically feel instantaneous for passwords of 8 characters or less (note we will not test passwords over 12 characters). We plan to do the testing/evaluation on ieng6.ucsd.edu.

For what it’s worth, my solution for each problem is less than 40 lines long and only half of those actually do anything (although my solution doesn’t do backtracking for timehack).

**Submitting**

We will use the *turnin* command line tool to submit project 1. The tool is accessed from your account on *ieng6.ucsd.edu*. Please only turn in the modified files: *memhack.c* and *timehack.c*

First, create a tar file. Change “YOURUSERNAME” to your username. You can use the following command:

```
tar cvf YOURUSERNAME.tar memhack.c timehack.c
```

Now, turn in the assignment. You can use the command:

```
turnin YOURUSERNAME.tar
```

You are able to submit the assignment multiple times, and we will grade your most recent submission. Please recall that late submissions will receive a penalty of 15% of their value for each day late.

Good luck!