Problem Set 2: Crypto

due by 7:00 P.M. on Friday, 10 October 2008

Be sure that your code is thoroughly commented
to such an extent that lines' functionality is apparent from comments alone.

Goals.

- Better acquaint you with functions and libraries.
- Allow you to dabble in cryptanalysis.
- Introduce you a bit early, perhaps, to file I/O.

Recommended Reading.

- Sections 11 – 14 and 39 of `http://www.howstuffworks.com/c.htm`.
- Chapters 7, 8, and 10 of *Programming in C*.

diff pset2.pdf hacker2.pdf.

- Hacker Edition challenges you to crack actual passwords.
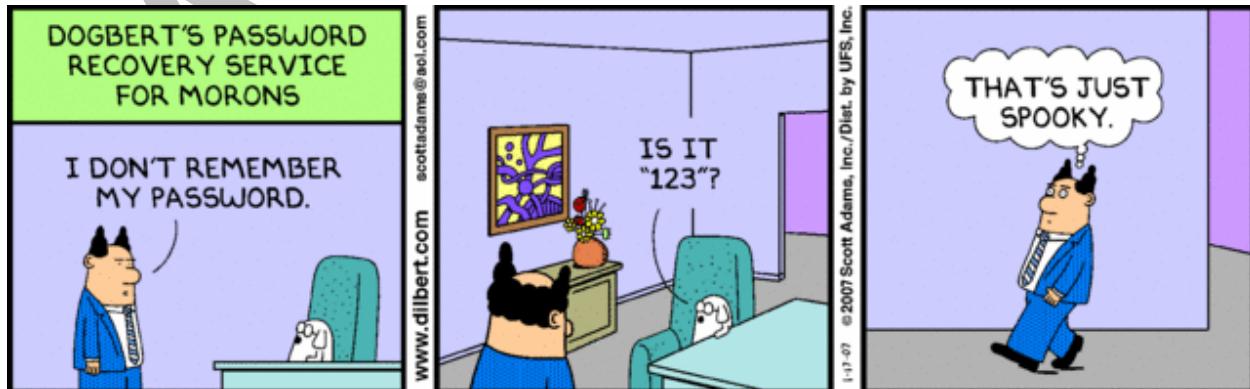
Academic Honesty.

All work that you do toward fulfillment of this course's expectations must be your own unless collaboration is explicitly allowed (*e.g.*, by some problem set or the final project). Viewing or copying another individual's work (even if left by a printer, stored in an executable directory, or accidentally shared in the course's virtual terminal room) or lifting material from a book, magazine, website, or other source—even in part—and presenting it as your own constitutes academic dishonesty, as does showing or giving your work, even in part, to another student.

Similarly is dual submission academic dishonesty: you may not submit the same or similar work to this course that you have submitted or will submit to another. Moreover, submission of any work that you intend to use outside of the course (*e.g.*, for a job) must be approved by the staff.

You are welcome to discuss the course's material with others in order to better understand it. You may even discuss problem sets with classmates, but you may not share code. In other words, you may communicate with classmates in English, but you may not communicate in, say, C. If in doubt as to the appropriateness of some discussion, contact the staff.

You may even turn to the Web for instruction beyond the course's lectures and sections, for references, and for solutions to technical difficulties, but not for outright solutions to problems on problem sets or your own final project. However, failure to cite (as with comments) the origin of any code or technique that you do discover outside of the course's lectures and sections (even while respecting these constraints) and then integrate into your own work may be considered academic dishonesty.

All forms of academic dishonesty are dealt with harshly.

Grades.

Your work on this problem set will be evaluated along three primary axes.

Correctness. To what extent is your code consistent with our specifications and free of bugs?
Design. To what extent is your code written well (*i.e.*, clearly, efficiently, elegantly, and/or logically)?
Style. To what extent is your code readable (*i.e.*, commented and indented with variables aptly named)?

Getting Started (in teh Cloud).

0. Surf on over to the course's website and follow the link to the course's **Bulletin Board**; you may be prompted to log in. Take a look around!

    Henceforth, consider the course's bulletin board *the* place to turn to anytime you have questions. Not only can you post questions of your own, you can also search for or browse answers to questions already asked by others.

    It is expected, of course, that you respect the course's policies on academic honesty. Posting snippets of code about which you have questions is generally fine. Posting entire programs, even if broken, is definitely not. If in doubt, simply email `help@cs50.net` with your question instead, particularly if you need to show us most or all of your code. But the more questions you ask publicly, the more others will benefit as well!

    Lest you feel uncomfortable posting, know that students' posts to the course's bulletin board are anonymized. Only the staff, not fellow students, will know who you are!

1. O hai, ~~guinea pig~~ beta tester! For this problem set, we're going to have you use `hacker2.cs50.net`, a prototype of our new cluster "in the cloud," instead of `nice.fas.harvard.edu`.[1]

    Thanks to Amazon's Elastic Compute Cloud (EC2), we're in the process of replacing the physical servers on which the course has relied for years (*i.e.*, `nice.fas.harvard.edu`) with "virtual machines" owned by Amazon but controlled by us.

    If unfamiliar with virtual machines, head on over to Wikipedia for a few minutes:

    `http://en.wikipedia.org/wiki/Virtual_machine`

    And also read up on cloud computing itself:

    `http://en.wikipedia.org/wiki/Cloud_computing`

    Then read up on the specific cloud that you're about to be in:

    `http://aws.amazon.com/ec2/`

    Truth be told, we're quite excited by what we'll be able to do this semester with all this technology. Not only will EC2 allow the course to grow and shrink its infrastructure as problem sets and final projects demand, it will also enable us to provide you, toward term's end, with a virtual machine of your own to which you have "root" (*i.e.*, administrator) access. All this for just $0.10 per hour. Not bad!

---

[1] However, you will still submit your work on `nice.fas.harvard.edu`, per this document's end.

Much like you have an account on `nice.fas.harvard.edu`, you're about to have an account on `hacker2.cs50.net` too. For simplicity, your username on `hacker2.cs50.net` will be identical to your FAS username, but realize that they are different accounts (with different home directories). If you change your password on one, it will not change on the other. If you edit some file on one, it will not change on the other. Functionally, though, you should find `hacker2.cs50.net` to be quite similar to `nice.fas.harvard.edu`.

We do hope you'll forgive if any technical difficulties sneak up on us this week! Glenn, Keito, and I (the course's sysadmins) will be watching the course's bulletin board constantly, as we're eager for feedback, particularly if things break.

If curious, here's a sketch of what you're about to try out!

2.   Head on over to

       `http://cs50.net/password/`
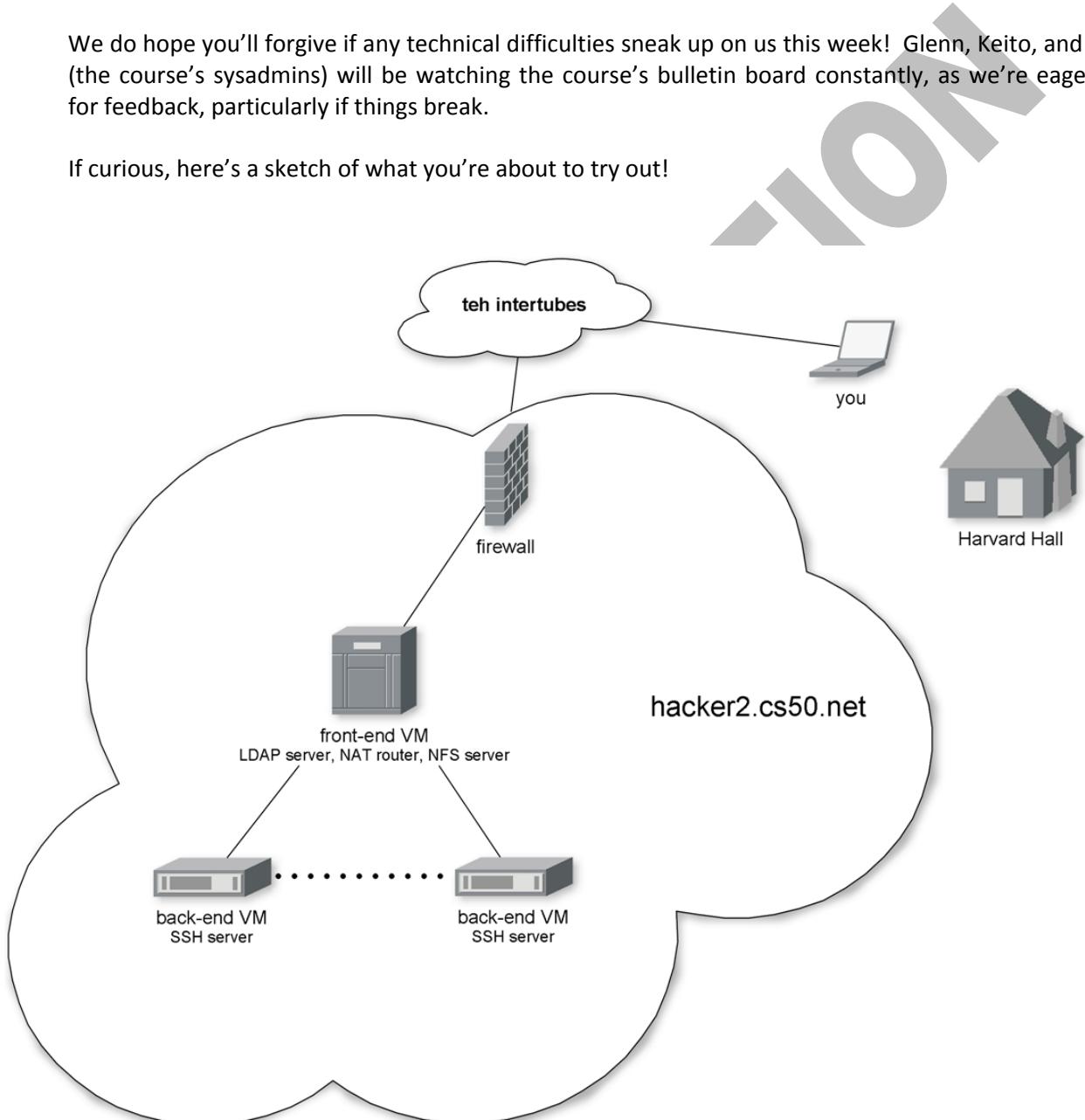
       to find out your password on `hacker2.cs50.net`. Then, actually SSH to `hacker2.cs50.net` (much like you would to `nice.fas.harvard.edu`), create a directory called `hacker2` in your brand-new home directory, and then navigate your way to `~/hacker2/`. (Remember how?) All of the work that you do for this problem set should reside in this directory so that we know where to look if we need lend you a hand. No need to put `hacker2/` in a `~/cs50/` subdirectory like you did on `nice.fas.harvard.edu`. We know it's for 50!

       Your prompt should now resemble the below.

       `username@cs50.net (~/hacker2):`

Passwords et cetera.

3.   On most, if not all, systems running Linux or UNIX is a file called `/etc/passwd`. By design, this file is meant to contain usernames and passwords, along with other account-related details (*e.g.*, paths to users' home directories and shells). Also by (poor) design, this file is typically world-readable. Thankfully, the passwords therein aren't stored "in the clear" but are instead encrypted using a "one-way hash function." When a user logs into these systems by typing a username and password, the latter is encrypted with the very same hash function, and the result is compared against the username's entry in `/etc/passwd`. If the two ciphertexts match, the user is allowed in. If you've ever forgotten some password, you may have been told that "I can't look up your password, but I can change it for you." It could be that person doesn't know how. But, odds are they just can't if a one-way hash function's involved.[2]

       Even though passwords in `/etc/passwd` are encrypted, the crypto involved is not terribly strong. Quite often are adversaries, upon obtaining files like this one, able to guess (and check) users' passwords or crack them using brute force (*i.e.*, trying all possible passwords). Only in recent years have (most) system administrators stopped storing passwords in `/etc/passwd`, instead using `/etc/shadow`, which is (supposed to be) readable only by root.[3] Below, though, are some `username:ciphertext` pairs from some outdated (fake) systems.

       ```
       phb:50Tybs3rY0aeo
       kitteh:50i2t3sOSAZtk
       blaise:505YXx3Mz50bg
       wbrandes:50T3oJBEcO1hA
       gcostanza:503/FQG.3pZTE
       sjobs:50R7A2X2CPB2I
       tdickson:HAHulqt5Yxuh6
       malan:HAqKvmEF5.iE6
       ```

---

[2] If you like this stuff, consider taking Computer Science 120 or 220r.

[3] Take a look at `/etc/passwd` on `hacker2.cs50.net`, for instance; wherever you see `'x'` a password once was.

Crack these passwords, each of which has been encrypted with C's DES-based `crypt` function. Specifically, write, in `crack.c`, a program that accepts a single command-line argument: an encrypted password.[4] If your program is executed without any command-line arguments or with more than one command-line argument, your program should complain and exit immediately, with `main` returning any non-zero `int` (thereby signifying an error that our own tests can detect). Otherwise, your program must proceed to crack the given password, ideally as quickly as possible, ultimately printing to `stdout` the password in the clear followed by `'\n'`, nothing more, nothing less, with `main` returning `0`. The underlying design of this program is entirely up to you, but you must explain each and every one of your design decisions, including any implications for performance and accuracy, with profuse comments throughout your source code. Your program must be designed in such a way that it could crack all of the passwords above, even if said cracking might take quite a while. That is to say, it's okay if your code might take several minutes or days or longer to run. What we demand of you is correctness, not necessarily optimal performance. Your program should certainly work on inputs other than these as well; hard-coding into your program the solutions to the above is not acceptable.

So that we can automate some tests of your code, your program must behave per the below; highlighted in bold is some sample input.

```
username@cs50.net (~/hacker2): crack 50Tybs3rY0aeo
123
```

Assume that users' passwords, as plaintext, are no longer than eight characters long. As for their ciphertexts, you'd best pull up the man page for `crypt`, so that you know how the function works. In particular, make sure you understand its use of a "salt." (According to the man page, a salt "is used to perturb the algorithm in one of 4096 different ways," but why might that be useful?) As implied by that man page, you'll likely want to put

```
#define _XOPEN_SOURCE
#include <unistd.h>
```

at the top of your file and compile your program with `-lcrypt`.

You might also want to read up on C's support for file I/O, as there's quite a number of English words in `/usr/share/dict/words` on `hacker2.cs50.net` that might (or might not) save your program some time.

By design, `/etc/passwd` entrusts the security of passwords to an assumption: that adversaries lack the computational resources with which to crack those passwords. Once upon a time, that may have been true. Perhaps some still do. But when it comes to security, assumptions are dangerous. May that this problem set make that claim all the more real.

---

[4] In case you test your code with other ciphertexts, know that command-line arguments with certain characters (*e.g.*, '?') must be enclosed in single or double quotes; those quotation marks will not end up in `argv` itself.

We should note that this problem set is no invitation to seek out other passwords to crack.[5] Do not conflate these Hacker Editions with "black hat" editions. We hope, though, that by understanding better the design of today's systems, you might one day build better systems yourself. Besides acquainting you further with C, this problem set urges you to start questioning designs, as vulnerabilities (if not regrets) often result from poor ones.

If you'd like to play with the staff's own implementation of `crack`, well, sorry! :-) Where'd be the fun in that?

Submitting Your Work.

4.   Ensure that your work is in `~/hacker2/` on `hacker2.cs50.net`. In order to submit your work, though, we're going to have you copy it over to `nice.fas.harvard.edu`. Execute the command below on `hacker2.cs50.net`.

```
scp -r ~/hacker2 username@nice.fas.harvard.edu:~/cs50/
```

As you may have inferred, this command will (securely) copy recursively your `hacker2` directory that's on `hacker2.cs50.net` to your `~/cs50/` directory on `nice.fas.harvard.edu`.

Now it's time to submit that copy. SSH to nice.fas.harvard.edu and double-check that the code now in ~/cs50/hacker2/ is indeed the code that you want to submit. Then submit your work by executing the command below.

```
cs50submit hacker2
```

Thereafter, follow any on-screen instructions until you receive visual confirmation of your work's successful submission. You will also receive a "receipt" via email to your FAS account, which you should retain until term's end. You may re-submit as many times as you'd like; each resubmission will overwrite any previous submission. But take care not to re-submit after the problem set's deadline, as only your latest submission's timestamp is retained. Of course, before resubmitting, you'll need to re-copy your code from `hacker2.cs50.net` to `nice.fas.harvard.edu` if you make changes on the former.

---

[5] In fact, do bear in mind the policies at
`http://www.fas-it.fas.harvard.edu/services/student/policies/rules_and_responsibilities.`