

## Ceboyz Frg 2: Pelcgb

qhr ol 7:00cz ba Sev 9/24

Cre gur qverpgvbaf ng guvf qbphzrag'f raq, fhozvggvat guvf ceboyz frg vaibyirf fhozvggvat fbhepr pbqr ba `pybhq.pf50.arg` nf jryy nf svyyvat bhg n Jro-onfrq sbez (gur ynggre bs juvpu jvyy or ninvynoyr nsagre yrpgher ba Jrq 9/22), juvpu znl gnxr n srj zvahgrf, fb orfg abg gb jnvg hagvy gur irel ynfg zvahgr, yrfg lbh fcraq n yngr qnl haarprffnevyf.

Or fher gung lbhe pbqr vf gubebhtuyl pbzragrq gb fhpu na rkrag gung yvarf' shapgvbanyvgl vf nccnerag sebz pbzragrf nybar.

### Tbnyf.

- Orggre npdhnvag lbh jvgu shapgvbaf naq yvoenevrf.
- Nybj lbh gb qnooyr va pelcgnanylfvf.
- Vagebqhpr lbh n ovg rneyl, creuncf, gb svyr V/B.

### Erpbzraqrq Ernqvaf.

- Frpgvbaf 11 – 14 naq 39 bs `uggc://jjj.ubj fghssjbexf.pbz/p.ugz`.
- Puncgref 7, 8, naq 10 bs *Cebtenzzvat va P*.

### qvss cfrg2.cqs unpxre2.cqs.

- Unpxre Rqvgvba punyyratrf lbh gb penpx npghny cnffjbeqf.





## Problem Set 2: Crypto

due by 7:00pm on Fri 9/24

Per the directions at this document's end, submitting this problem set involves submitting source code on `cloud.cs50.net` as well as filling out a Web-based form (the latter of which will be available after lecture on Wed 9/22), which may take a few minutes, so best not to wait until the very last minute, lest you spend a late day unnecessarily.

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 hacker2.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 in writing by the course's instructor. Collaboration in the completion of problem sets is not permitted unless otherwise stated by some problem set's specification.

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, 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. Nor may you provide or make available solutions to problem sets to individuals who take or may take this course in the future. 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 course's instructor.

You may 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. If the course refers some matter to the Administrative Board and the outcome for some student is *Warn*, *Admonish*, or *Disciplinary Probation*, the course reserves the right to impose local sanctions on top of that outcome for that student that may include, but not be limited to, a failing grade for work submitted or for the course itself.

## 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)?

## Help!

- Surf on over to

`http://help.cs50.net/`

and log in if prompted. Then take a look around!

Henceforth, consider `help.cs50.net` *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 flag your discussion as "private" or 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.

## Getting Started.

- Alright, here we go! SSH to `cloud.cs50.net`, create a directory called `hacker2` in your home directory, and then navigate your way to that directory. (Remember how?) Your prompt should now resemble the below.

```
username@cloud (~/.hacker2):
```

If not, retrace your steps and see if you can determine where you went wrong. You can actually execute

```
history
```

at the prompt to see your last several commands in chronological order if you'd like to do some sleuthing. You can also scroll through the same one line at a time by hitting your keyboard's up and down arrows; hit Enter to re-execute any command that you'd like. If still unsure how to fix, remember that `help.cs50.net` is your new friend!

All of the work that you do for this problem set must ultimately reside in your `hacker2` directory for submission.

## Passwords et cetera.

- 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.<sup>1</sup>

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.<sup>2</sup> Below, though, are some `username:ciphertext` pairs from some outdated (fake) systems.

```
guest:50Bt2CexZzo7k
jcaesar:50zPJlUFIYY0o
cpisonis:HAR0lc1egPNKQ
pskroob:50Bpa7n/23iug
mscott:50q.zrL5e0Sak
gcostanza:50vfotBdeBr.o
bvigener:505YXx3Mz50bg
dmalan:50SRu3A7lmm4A
```

Crack these passwords, each of which has been encrypted with C's DES-based (not MD5-based) `crypt` function. Specifically, write, in `crack.c`, a program that accepts a single command-line argument: an encrypted password.<sup>3</sup> 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 standard output 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

---

<sup>1</sup> If you like this stuff, consider taking Computer Science 220r.

<sup>2</sup> Take a look at `/etc/passwd` on `cloud.cs50.net`, for instance; wherever you see `'x'` a password once was.

<sup>3</sup> 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.

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@cloud (~/hacker2): ./crack 50Bt2CexZzo7k  
guest
```

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 link your program with `-lcrypt`. (If you use `make` to compile your code, that switch will be included automatically.)

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 `cloud.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.

We should note that this problem set is no invitation to seek out other passwords to crack.<sup>4</sup> 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?

---

<sup>4</sup> In fact, do bear in mind the policies at [http://www.fas-it.fas.harvard.edu/services/student/policies/rules\\_and\\_responsibilities](http://www.fas-it.fas.harvard.edu/services/student/policies/rules_and_responsibilities).

## How to Submit.

In order to submit this problem set, you must first execute a command on `cloud.cs50.net` and then submit a (brief) form online; the latter will be posted after lecture on Wed 9/22.

- SSH to `cloud.cs50.net`, if not already there, and then execute:

```
cd ~/hacker2
```

If informed that there is “no such file or directory,” odds are you skipped some step(s) earlier! Not a problem, we’ll help you fix in a moment. If the command indeed works, proceed to execute this command next:

```
ls
```

At a minimum, you should see `crack.c`. If not, odds are you skipped some more steps earlier! If you do see those files, you’re ready to submit your source code to us:<sup>5</sup>

```
~cs50/pub/bin/submit hacker2
```

That command will essentially copy your entire `~/hacker2` directory into CS50’s own account, where your TF will be able to access it. You’ll know that the command worked if you are informed that your “work HAS been submitted.” If you instead encounter an error that doesn’t appear to be a mistake on your part, do try running the command one or more additional times. (The `submit` program’s a bit old and annoyingly flakey.) But if informed that your “work HAS been submitted,” it indeed has.

Now, if you instead see some message informing you that your code is not where it should be, you’ll need to fix that before you submit. Review the last two pages of Problem Set 1’s spec for some tips!

- Anytime after lecture on Wed 9/22 but before this problem set’s deadline, head to the URL below where a short form awaits:

```
http://www.cs50.net/psets/2/
```

If not already logged in, you’ll be prompted to log into the course’s website.

Once you have submitted that form (as well as your source code), you are done!

This was Problem Set 2.

---

<sup>5</sup> Note that there is no slash between this tilde (~) and `cs50`.