Problem Set 1
Due: 11:59pm, Wednesday February 10

Revisions:

- **Tue Feb 09:** In Problem 4, the wording “the binary digits and decimal numbers” has been corrected to “the binary and other digits”.

- **Sun Feb 07:** (1) Added link http://tinyurl.com/wellesley-cs342-find-grep to explain subtleties of symbolic links in the context of `find` and `grep`; (2) added more notes on the negative offset case for Problem 7, including an extra credit challenge.

- **Tue Feb 02:** (1) The previous version of Problem 6 assumed you were working on an Ubuntu VM you don’t yet have. The problem has been rewritten to assume that you’re working on cs.wellesley.edu. (2) in the first note for Problem 7, the list of micro-focus Linux machines has been updated.

Overview:

The purpose of this assignment is to get you to get you in the mindset of thinking about computer security and getting you familiar with some concepts and tools you will need.

Working Together:

The basic rule of collaboration in this class is that you can talk with anyone about how to solve any problem but you can’t take away anything (notes, programs, etc.) from such a discussion and must write up all programs and solutions completely on your own. This is essentially Yoshi Kohno’s “Gilligan’s Island Rule” described at https://www.cs.washington.edu/education/courses/cse484/12sp/administrivia/overview.html, whose purpose is to ensure that you truly understand everything you write up. You must also explicitly list your collaborators on every problem (Kohno’s “Freedom of Information Rule”).

Submission:

You will submit all your work for this pset in a single PS1 Google doc linked from your CS342 Portfolio doc that you create in Problem 1. Follow these guidelines:

1. In your course Blog (see Problem 1) you should summarize any issues you encountered while working on the problems, insights you gained, resources you consulted, etc.

2. At the beginning of each problem, indicate the time you spent on that problem.

3. For Problem 2, include (1) the name of and link to the article or blog post you chose, (2) the paragraphs you wrote about the article and posted to the CS342 Google Group, and (3) the text of your additional posts to the CS342 Google Group discussing other articles.

4. For Problem 3, submit a transcript of the Linux command and its results.

5. For Problem 4, submit the the decoded T-shirt message and explain how you decoded it.

6. For Problem 5, submit the Unicode codepoint sequence and an image of how the characters should look. Also explain how you determined these.

7. For Problem 6, show the Emacs keystroke commands you typed to transform the lines of the buffer to be in sorted order from largest size down.

8. For Problem 7, submit (1) a transcript of your REPL interaction that culminates with the desired result and (2) a companion explanation of why you chose the commands you did.
Problem 1 [5]: Creating your portfolio doc and starting your course blog

Create a Google doc entitled your name CS342 Portfolio that you share with me with commenting privileges. You will use this Portfolio doc to submit all your work for this course. This doc will simply contain a list of links to docs of your submitted work. For starters, it will have a link to your Blog doc (see below) and your PS1 doc (see above.)

Create a second Google doc entitled your name CS342 Blog that you share with me with commenting privileges and that you link from your Portfolio doc. You will use this Blog doc to record your journey through the course. Every day you work on the course, you should add a dated entry to the top of this page summarizing your work for the day, emphasizing insights you gained, difficulties you encountered, resources you discovered, and security-related news/articles you read. Include images when appropriate. Put all your entries in a single document in reverse chronological order.

Problem 2 [20]: Discuss a Security Article or Blog Post

As emphasized in Lecture 1 and the associated readings (see the course schedule), we will be thinking about security topics in the context of the security mindset and the hacker curriculum.

To get you into this mode of thinking, in this problem, you will do the following:

- Find an online article or blog post related to computer security that you can analyze in terms of the security mindset and/or ideas from the hacker curriculum. Choose a topic not yet posted by someone else in the class. (This is a motivation to do this problem early!)
- Write an email to the CS-342-01-SP16 Google Group that includes (1) a link to the article and (2) one paragraph that summarizes the article and a second paragraph that analyzes the article from the perspective of the security mindset and/or the hacker curriculum.
- Write a nontrivial response to at least two of your classmates posts for this problem. E.g., add something nontrivial to the discussion of the article, relate it to another topic or article, or evaluate the analysis (pro and/or con).
- Include in your PS1 writeup the link to your article, your paragraphs about it, and the text of your discussions of other articles.

Problem 3 [15]: Linux Skills

In this course, you will be doing a lot of work with Linux. So it’s important to gain familiarity with lots of Linux commands. Figure 1 lists some of the commands you should be familiar with. The list is by no means exhaustive! Many of the commands have lots of options; some of the common options ones are listed. To get documentation on this commands, use man and info, browse on-line resources, and refer to the many Linux books on the security bookshelves in SCI 160A.

You should also play with pipes (|) and input/output redirection (<, >, >>) and learn a little bit about shell scripts (which we will cover soon in a lab or lecture). You can learn about Linux commands and shell scripts by consulting the Linux resources on the CS342 Resources page.
For this problem, you should play around with Linux commands and shell programming. The only thing you need to submit for this problem is the following:

In your account on cs.wellesley.edu (tempest), write a single Linux command that lists every line containing the word “vulnerabilities” in every publicly readable text (non-binary) file in the directory /home/cs342. Each line should have the form filename:line, where line is the contents of the line containing the word and filename is the name of the file containing that line. The find and grep commands are very useful for this problem. The 2> error redirection mechanism is helpful for suppressing annoying error messages.

Your command should ignore case. E.g., it should list lines containing strings like “Vulnerabilities”, “VULNERABILITIES”, and “vulneraBiliTies”.

Your command should also not follow symbolic links. This will greatly reduce the number of matching lines. Using the find command in conjunction with grep is helpful in this regard. However, there are numerous subtleties involving the processing of symbolic links with find and grep that you need to understand to get the correct output. See http://tinyurl.com/wellesley-cs342-find-grep for details.

Submit a transcript of the command and the results of executing it.

Problem 4 [15]: Decoding a T-shirt
Margaret Ligon ’14, a student in the 2012 CS342 class, designed a T-shirt for the Wellesley participants in the Lincoln Laboratory Capture the Flag (CTF) contest. The back of the T-shirt is shown in figure 2. The binary and other digits on the shirt stand for a message. What is the message?

Guidelines/hints

• An ASCII table such as the one at http://www.asciitable.com/ is very helpful in this problem.

• For full credit, you must not only specify the message, but describe the details by which you decoded it.

Problem 5 [15]: Decoding a message
You have been sent a message that consists of the following bytes (expressed in hex):

49 E2 99 A5 CF 80 21

You have been informed that this message consists of Unicode characters that have been encoded into a byte sequence via the UTF-8 encoding.

a [12] What is the sequence of Unicode codepoints in this UTF-8 encoded message? For example, codepoint U+03B1 is the Greek letter α (http://unicode.org/charts/PDF/U0370.pdf). Explain how you determined the codepoints.

b [3] What would this message look like when displayed in a Unicode-enabled application?

Guidelines/hints


• Consult the following on UTF-8: http://en.wikipedia.org/wiki/UTF-8.

Problem 6 [15]: Learn Emacs
Emacs is a powerful text editor that will be very helpful to you as you play sysadmin on your Ubuntu VM, so it’s a good idea to become familiar with it.

Here are three ways to launch Emacs:
• You can launch Emacs by clicking on the GNU Emacs icon in the icon panel on the left of your Ubuntu VM. This creates a separate Emacs window with GUI features. In this window, the permissions you will have for reading and writing files will be those of the account in which you are logged in to the Ubuntu machine.

• You can also launch Emacs editor in a separate GUI window from a Linux shell via `emacs &` (the “&” means execute the command in a separate process). In this window, the permissions you will have
for reading and writing files will be those of the account in which you are logged in to the shell. This is handy if you want to edit a file as root, say. First su to root in a shell, and then launch Emacs from that shell.

- You can also launch Emacs within a Linux shell via `emacs -nw` (“-nw” means “no window”), which gives a version of Emacs in the shell without GUI features.

In this problem, you should first follow the Emacs tutorial by selecting Help>Emacs Tutorial from an Emacs GUI window or typing Ctrl-h t (first press the “control” and “h” keys at the same time, then press the “t” key). This will teach you the basics of Emacs. Although you can use GUI features like menus to perform many tasks in Emacs, it is important to lean the keystroke commands as well. Why? Three reasons: (1) once you become familiar with the keystroke commands, they are faster than using the mouse; (2) sometimes you will have access only to non-GUI versions of Emacs; and (3) you can do amazing things in Emacs by defining so-called keyboard macros based on keystrokes.

Once you have taken the tutorial, explore other Emacs resources at http://cs.wellesley.edu/~cs342/resources.html. Particularly useful is the two-page Emacs reference card. It’s a good idea to print this out for your personal use.

To get a sense for the kinds of file manipulations you can do in Emacs, in your cs.wellesley.edu account, create a `du-cs342.txt` file as follows:

```
du /home/cs342 > du-cs342.txt 2> /dev/null
```

and then view `du-cs342.txt` in Emacs. It lists subdirectories (recursively) of the /var directory, preceded by a size measure. It might include lines like the following:

```
... 
32   /home/cs342/archive/shared/papers/formatstring/examples/fmtattr_test 
260  /home/cs342/archive/shared/papers/formatstring/examples 
512  /home/cs342/archive/shared/papers/formatstring 
16   /home/cs342/archive/shared/papers/CVS 
4476 /home/cs342/archive/shared/papers 
14232 /home/cs342/archive/shared 
4    /home/cs342/archive/cs342_spring16/handouts 
28   /home/cs342/archive/cs342_spring16/public_html/cgi-bin/hilo-sessions 
636  /home/cs342/archive/cs342_spring16/public_html/cgi-bin 
35872/home/cs342/archive/cs342_spring16/public_html/lectures 
4    /home/cs342/archive/cs342_spring16/public_html/protected 
688  /home/cs342/archive/cs342_spring16/public_html/handouts 
37908 /home/cs342/archive/cs342_spring16/public_html 
37916 /home/cs342/archive/cs342_spring16 
4    /home/cs342/archive/cs342_fall14/drop/ps7/lys 
... 
```

**Writeup:** Show the Emacs keystroke commands you typed to transform the lines of the buffer to be in sorted order from largest size down.

**Note:** The Emacs `M-x apropos` command is a good way to find other Emacs commands by keyword. In this context, it’s helpful to use `M-x apropos` to search for terms like `sort` and `reverse`.

**Note:** Emacs has an amazing number of built-in commands, some of which are quite humorous. Try some of the following: `hanoi`, `doctor`, `yow`, `psychoanalyze-pinhead`, and `dissociated-press` (try this last one on a text document).

**Problem 7 [15]: Exploiting a C program**

This problem illustrates the fundamental insecurity of arrays in C and hints at the role that character buffers play in stack buffer exploits. It also give you some experience with thinking like a hacker.

Figure 3 is a C program `arrays.c` available in `~cs342/download/ps1`. When compiled and executed, it enters a read/eval/print loop (REPL) that responds to commands. For example, the `display` command shows the addresses and contents of all slots in the three integer arrays `a`, `b`, and `c` manipulated by the program:
[cs342@jay ps1] gcc -o arrays arrays.c
[cs342@jay ps1] ./arrays

Enter one of these three commands: display, setb, quit:
> display
bfba580 a[0]: 1 (int); 1 (hex)
bfba584 a[1]: 2 (int); 2 (hex)
bfba588 b[0]: 3 (int); 3 (hex)
bfba58c b[1]: 4 (int); 4 (hex)
bfba590 b[2]: 5 (int); 5 (hex)
bfba594 c[0]: 6 (int); 6 (hex)
bfba598 c[1]: 7 (int); 7 (hex)
bfba59c c[2]: 8 (int); 8 (hex)
bfba5a0 c[3]: 9 (int); 9 (hex)

Enter one of these three commands: display, setb, quit:
> display
bfba59c a[1]: 17 (int); 11 (hex)
bfba590 b[0]: 23 (int); 17 (hex)
bfba594 b[1]: 5 (int); 5 (hex)
bfba598 c[0]: 42 (int); 2a (hex)
bfba5a0 c[1]: 7 (int); 7 (hex)
bfba5a4 c[2]: 8 (int); 8 (hex)
bfba5a8 c[3]: 9 (int); 9 (hex)

Enter one of these three commands: display, setb, quit:
> quit

For this problem, submit (1) a complete transcript of your REPL interaction that ends with the desired result and (2) a companion explanation of why you chose the commands you did.

Guidelines/hints

- You can either run this program in your Ubuntu VM or on one of the micro-focus Linux machines (e.g., cardinal, finch, orangutan, gorilla, chimp, tamarin, gibbon, baboon, lemur, etc. you can connect to these remotely via ssh).

- Study arrays.c carefully to understand exactly how it works. But you are not allowed to change it in any way!

- It is straightforward to use the setb command to change b[1], but how the heck you change a[1] and c[0]? That’s the crux of this problem, and it requires thinking like a hacker.

- The addresses of all integer array slots and character array slots are displayed for a reason. Study them to understand the key to your exploits. Of course, in a typical program they wouldn’t be printed, but you’d have to imagine they were there. You’d also have to do a lot more experimentation without them.

- Changing array slots at a positive offset from a[0] is fairly easy, but changing them at negative offset is hard, because the REPL converts negative indices entered by the user to 0. Nevertheless, there are two very different ways to get around this problem. Hints: (1) an unexpected command string can be used in a nefarious way; (2) an unusual offset can used as an exploit. Solve this in one way for full credit. But if you find both ways, you get extra credit!
# include <stdio.h> // I/O operations
# include <string.h> // string operations

// Display the word addresses and integer contents of len slots of given array
void display_array(char* name, // string name for array
                    int* array, // array of ints is pointer to (word address of) int in 0th slot
                    int len) { // need to pass length of array separately
    int i = 0; // initialize index to 0
    int* end = array + len; // end is address of word after last array slot
    for (; array < end; array++) { // Loop iterates through word addresses of array slots.
        // Incrementing adds 4 to the address b/c array is int pointer
        printf("%x %s[%i]: %i (int);\t%x (hex)\n", array, name, i, *array, *array);
        // *array is contents of current slot
        i++; // increment index in sync with address of next slot
    }
}

// Display the byte addresses and character contents of all slots in given string
void display_chars(char* str) { // str is pointer to char in 0th slot
    while (*str != 0) { // Loop while terminating null byte hasn’t been reached
        printf("%x: %c (char), %x (hex)\n", str, *str, *str);
        str++; // Incrementing adds 1 to address b/c str is a char pointer
    }
}

// Return the maximum of two integers
int max(int a, int b) { if (a > b) return a; else return b; }

// Program entry point
int main (int argn, char** argv) { // argn and argv are ignored in this program
    int a[2] = {1,2}; // Allocate integer arrays on stack
    int b[3] = {3,4,5};
    int c[4] = {6,7,8,9};
    char command[8]; // Stack space allocated for command string character buffer.
    // No one would type more than 8 characters, would they? ;-)
    int index; // Stack space allocated for "setb" command index
    int value; // Stack space allocated for "setb" command value

    // Read/eval/print loop: read a command from user, perform action, and repeat
    while (1) { // 1 is how "true" is written in C; "infinite" loop exited via "quit" command
        printf("Enter one of these three commands: display, setb, quit:\n""); // Prompt for command
        scanf("%s", &command); // Read command into character buffer
        if (strcmp(command,"display")==0) { // strcmp compares strings; 0 result means they're equal
            // Display the addresses and contents of slots in all arrays
            display_array("a", a, 2);
            display_array("b", b, 3);
            display_array("c", c, 4);
        } else if (strcmp(command,"setb")==0) { // Change slots in array b
            printf("Choose an index for array b: "); // Prompt for index of array b to change
            scanf("%i", &index); // Store it into index variable
            index = max(index,0); // Don’t allow negative indices; convert negative index to 0
            printf("Choose a new value for b[%i]: ", index); // Prompt for new value at index
            scanf("%i", &value); // Store it into value variable
            b[index] = value; // Change the indexth slot of b to new value
        } else if (strcmp(command,"quit")==0) {
            return 0; // Exit loop by returning from main function
        } else { // Case for unrecognized commands
            printf("\n"%s" is not a recognized command.\n"");
            display_chars(command); // Display the addresses and contents of characters in command
            printf("Try again.\n");
        }
    }
}

Figure 3: A C program manipulating some arrays.