Project 4

Due: Tuesday, May 17 at 11:59 PM (10 point penalty per day for late submission)

Goals:

- Learn all the details of a buffer overflow attack
- Reinforce understanding of X86 stack frames
- Learn how to create object code from X86 assembler

Background: So far, we have had both a lecture and a lab that covered the basics of a buffer overflow attack. In Project 4, we will complete the process – generating a complete buffer overflow attack, including inserting your own "nefarious" code - to modify the results of a specific program.

Note that the techniques you will use for this attack will not work outside our specialized lab environment. I have designed the code in this project to be particularly vulnerable to a buffer overflow attack. Besides, as good citizens, we would never try to use these methods for evil purposes.

Specifications: You will be given the C code called "grades.c" and a Makefile that compiles grades.c into a binary file called "grades". The grades program does the following:

1. Generates random grades for the CS220 class.
2. Calculates the weighted average of those grades for all students.
3. Prompts the user for a his section… either "A" for section A, or "B" for section B. (Of course, if there happens to be anything after the "A" or "B", we won't notice… after all, I did say the grades program was vulnerable to attack.)
4. Prints out the grades and the averages for all the students in your section.

The student identification numbers (SID) used in this project are the same as the student numbers used in the bomb project. For instance, if you had bomb105 for Project 3, then you will find your grade under SID 105 in the output produced by the grades program.

You need to provide an input file that includes the section letter. Your grade for this project will be the result of applying your input file to my copy of the grades binary file with a random seed of my choice.

Note that the random number generator used to generate the grades is normally seeded with the time date stamp, so grades generates a different grade for you every time it runs. However, these random grades are designed to generate a decent grade (between the high 50's and mid 70's), so if you submit an input file that simply contains a valid section ID, you are guaranteed of getting some random grade that might not be too bad. However, the point of

this project is to provide an input file which alters your grade, and guarantees that you get a very good grade.

When we run your input file, we will be making some checks on your results. If these checks fail, we will know that you have been hacking, and will subtract points from your project 4 grade. The checks we will be making are as follows:

Check	Score Impact
If any grade other than your own has been modified…	-20 points
If any of the grades in your row are less than zero, or greater than the maximum grade….	-20 points
If your average grade is calculated incorrectly, based on the algorithm in the getAvg function in grades.c…	-20 points
If Output lines (e.g. student grades) are missing compared to normal run…	-20 points
If extra messages (e.g. "segmentation violation") appear…	-20 points
Submission error (e.g. file named incorrectly) …	-10 points
First to report a valid bug in the code supplied in the tar file	+5 points

Implementation: Download the tar file for Project4 and untar. This will create a proj4 directory which contains several files, as follows:

- Makefile – A makefile to compile, test, and debug the grades executable.
- grades.c – The C code used to create the grades executable.
- test.txt – An example input file that initially specifies the "B" section.
- dbg_cmds.txt – An example input file that provides gdb startup commands

Your job is to come up with an input file which not only specifies the password, but which also alters the execution of the grades executable in such a way as to modify your individual grades, as well as the average grade, without altering anything else about the results of the program.

In order to do that, you will need to generate your own object code, figure out how to get that object code into the input file in such a way that it ends up in memory that allows program execution, and then figure out how to modify the return address to branch to your new object code. Your object code will need to repair/manage the stack frame that may have been corrupted by your buffer overflow attack, modify your grades and average grade, and then return to normal program execution.

Note that it may be possible to work on Project 4 on your own hardware, but, since there are differences between different implementations, and your result will be tested on an LDAP machine, make sure your input file works correctly on an LDAP machine before submitting.

Submission: Once you have completed implementing and testing your input file, rename the file to <userid>.txt, and load it into BlackBoard in the project 4 submission area.

Grading: Your input file will be run against the professor's version of the grades executable on an LDAP machine. Unless there are other problems, such as alteration of other student's grades, or incorrect averages (see the table above), then your grade for Project 4 will be the average grade for your SID printed out by the grade program.

Hints:

- Work on this project wherever you want to (e.g. LDAP or Cygwin or on your dual boot laptop), but test it on an LDAP machine before you submit. Not all UNIX implementations are exactly the same!
- Read through the grades.c program. There are some very big hints on where you can put your new object code so that it is easy to execute. Note that in general, you can write to pages in the stack, but cannot execute code in those pages. You can execute code in pages in the .text section once it is loaded into memory, but you cannot write to those pages. You can write to global data in the .data section once it is loaded into memory, but you cannot execute instructions in that memory. I have created a page in memory for you which you are allowed to both write AND execute instructions. You need to find that page in memory and use it.
- You may want to create a temporary version of grades.c that does the modifications you would like it to. Then you can look at the X86 code that modified your grades to figure out how these modifications can be done in X86 assembler.
- You will need to create binary object code which modifies your grades and handles the X86 stack. There are many ways to do this. I found it easiest to create a file with x86 assembler code, which I called "hack.s". Then, using the command:

  gcc –m32 –c –o hack.o hack.s

  I can get binary object code in hack.o. Use "objdump –d hack.o" to get a man-readable listing. I can figure out where the binary object code is by running objdump -h hack.o, and looking at the table of contents entry for the .text section. (Note that hack.o is an ELF format file, so your object code doesn't start at the beginning of the file.) That gives me enough information to know where the binary object code starts in the hack.o file, and how long it is.
- You will need to write a program to generate a new test.txt file that contains both ASCII data (for the section ID), and binary data (for instance, your code and the "return address"). Write this program in C if that's the easiest for you. Some students write this code in python or java or some other language they were very familiar with. I don't care what language you use… you don't need to turn in the program that writes test.txt… just turn in the correct file.
- When writing your input file, avoid the hexadecimal value 0x0A. In ASCII, 0x0A is a newline character, and the "gets" function called by grades will stop reading when it runs into a newline character.