

# p5: Safe Cracker

**Due** Apr 21 by 11:59pm    **Points** 90    **Submitting** a file upload

**Available** Apr 10 at 12am - Apr 22 at 11:59pm

This assignment was locked Apr 22 at 11:59pm.

[GOALS](#)   [SAFES](#)   [TOOLS](#)   [HINTS](#)   [REQUIREMENTS](#)   [SUBMITTING](#)

Announcements:

- Released - Get Started and crack your first safe using Linux tools like **objdump** and **gdb**

## Learning GOALS

- Interpret IA-32 (x86) assembly language
- Use the x86 disassembler, **objdump**
- Use the command line debugger tool **gdb** with x86.

## SAFES to be Cracked OPEN

In this assignment, you will be unlocking four "binary safes". Each binary safe is an executable program that prompts the user for five inputs via the stdin console. The user must provide the correct inputs in order to crack open the safe, but if the wrong input is entered then an alarm is triggered! Well, it just prints out the alarm, but you can imagine it having consequences on your grade if you can't figure out the inputs.

## Getting the Required Files

The four binary safes are unique for every student and are located in the following directory for your:

```
/p/course/cs354-deppeler/public/students/<your-cs-login-ID>/p5/
```

Replace <your-cs-login-ID> with ***your actual cs login*** and copy the contents of the above directory into your private/p5 working directory. There are four executable files named **s1**, **s2**, **s3**, and **s4**. You must crack the codes to your unique safes.

## Cracking the Safes

The challenge is to figure out the correct set of 5 inputs expected by each of the four safes. When you run your binary safes, you'll need to type in your guesses, one at a time, as shown below:

```
[deppeler@liederkranz] (55)$ ls
s1* s2* s3* s4*
[deppeler@liederkranz] (56)$ ./s3
input 1 (of 5)? elephant-ears
```

```

input 2 (of 5)? one-two-three
input 3 (of 5)? at-the-end
input 4 (of 5)? open-two-door
*****
***** ALARM TRIGGERED *****
*****
[deppeler@liederkranz] (57)$ ./s3
input 1 (of 5)? elephant-ears
input 2 (of 5)? one-two-three
input 3 (of 5)? at-the-end
input 4 (of 5)? open-the-door
input 5 (of 5)? alphabet-soup
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!! SAFE OPENED !!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
[deppeler@liederkranz] (58)$

```

Once you've figured out all of the inputs for a safe, create a text file containing the five lines of input to solve the associated safe followed by a single empty line (**press enter after the last input**). Name your text files accordingly: **s1.solution**, **s2.solution**, **s3.solution**, and **s4.solution**. Note: Linux does not require text files to have the .txt extension, but Mac and Windows users may find that those editors add the .txt suffix. We test the solution files you submit using input redirection with your binary safe executables as shown below. So be sure to have the filename and contents exactly as required for each safe to open.

**CAUTION: DO NOT ADD SOURCE FILE HEADERS TO YOUR SAFE SOLUTION FILES!** The solution files you submit will be used as input files to open your safe executables. If they have anything other than the required codes, or are in a file with the wrong file name or extension, they will not open your safes and will fail the tests.

```

[deppeler@liederkranz] (77)$ ls
s1* s1.solution s2* s2.solution s3* s3.solution s4* s4.solution
[deppeler@liederkranz] (78)$ cat s1.solution
951905
994563
493693
828695
278566
[deppeler@liederkranz] (79)$ ./s1 < s1.solution
input 1 (of 5)? input 2 (of 5)? input 3 (of 5)? input 4 (of 5)? input 5 (of 5)?
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!! SAFE OPENED !!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
[deppeler@liederkranz] (80)$

```

**NOTE:** The output from the file redirection will be mostly on one line and may wrap differently than shown above. But it should indicate the safe opened as shown above. All 5 inputs need to be correct to open a safe. **If the alarm is triggered, no points will be given no matter how many inputs were correct before the alarm.** You can verify your solution files by testing them as shown above. If your solution file has a problem, test using input redirection might appear to be in an infinite loop. If that happens, press ctrl-c to break out of the program and continue working on your solution files.

**CAUTION:** Create your solution files with a text editor (vim/emacs/gedit/nano) on a CS Department's instructional Linux machine. Don't create or edit or even open your solution files on **Windows or Mac** because this may result in problems with the end-of-line mark. If the line endings are changed or saved with other text editors it will cause your solutions to fail when tested with Linux input redirection. **Make sure your submitted files work by downloading a copy of the files that you submitted to Canvas and running them on the the lab computers as shown above.** If your file does not work on Linux, it is likely due to differences in the file line ending. You can check the file type with the **file** command. It must show that the file type is **ASCII text** and it must not show **"**, with CRLF line terminators":

```
> file s1.solution
s1.solution: ASCII text
```

If your file has something other than ASCII text, you can fix this in vim. Open using vim and change the file format to unix

```
> vim s1.solution
:set fileformat=unix
```

## TOOLS: **objdump** and **gdb**

To figure out how to open your binary safes, you will use two powerful tools: **objdump** and **gdb**. Both are critical in reverse engineering each binary safe to determine the inputs required to "open" the safe.

### objdump

**objdump** is a command in Linux to display information about object files. For this project, the two important command line options are:

- **-d** which disassembles a binary
- **-s** which displays the full binary contents of the executable

For example, to see the assembly code of safe s1, you would enter:

```
objdump -d s1
```

Start by looking in **main()** to begin figuring out what the code is doing.

The **-s** flag is also quite useful as it shows the contents of each segment of the executable. This can be used when looking for the initial value of a given variable.

Use output redirection to save the output of **objdump** in a file, so that you don't have to repeatedly regenerate it every time. Both command line options can be used at the same time to create a full dump of the contents of the executable as well as the disassembled contents.

# gdb

By now, you've used the debugger, **gdb**, to help find segmentation faults and hopefully practice syntax for 2D arrays on the heap. But it's an even more powerful tool in your search for clues to open each binary safe. The following **gdb** commands in addition to the ones specified in p3 are useful:


- **finish**: continue until the current function returns and prints the return value (if any)
- **print**: prints the contents of variable or memory location or register
- **set var <variable\_name>=<value>**: changes the content of a variable to the given value while in debugger (does not change source code)
- **info registers**: shows you the contents of all of the registers of the system
- **x/nfu <addr>**: The powerful **examine** command shows you the contents of memory.
  - **n** specifies the number of units,
  - **f** specifies how to format the memory, and
  - **u** says what unit to use (b for 1 byte, h for 2 bytes, w for 4 bytes).
  - **addr** is the hexadecimal address you want to look at.

For example, "**x/2ub 0x54320**" is a request to display 2 bytes of memory formatted as unsigned decimal integers (**n** is 2 units, **f** is unsigned, **u** is byte units) starting at the address 0x54320.

## HINTS

- Every C program has a **main()** function. Figure out how to locate it.
- A safe program's loop in **main()** iterates five times. Remember that each safe requires five inputs.
- On a wrong input, function **fail()** is called. This results in the alarm being triggered. Find where "fail" is called.
- If all five inputs are correct, function **success()** is called.
- The *callee* function's arguments are set up by the *caller* in its stack just prior to the function call.
- Function **strtol()** corresponds to the use of **atoi()** in C source code.
- The two parameters of **strcmp()** are addresses to two C strings.
- The return value of a function is stored in **%eax**.
- Safe **s1** is the easiest and the other three more challenging.

For some safes, cracking them requires more than just finding the codes.


- **GDB\_Cheat\_Sheet.pdf** (<https://canvas.wisc.edu/courses/330348/files/30412116?wrap=1>)  ([https://canvas.wisc.edu/courses/330348/files/30412116/download?download\\_frd=1](https://canvas.wisc.edu/courses/330348/files/30412116/download?download_frd=1)) (you really only need break points, step, next, and how to display variables, registers, etc.

A short video showing how to step through x86 assembly instructions.

<https://www.youtube.com/watch?v=wluZajISL-E>  (<https://www.youtube.com/watch?v=wluZajISL-E>)



(<https://www.youtube.com/watch?v=wluZajlSL-E>)

- [x86-cheat-sheet.pdf](https://canvas.wisc.edu/courses/330348/files/30412095?wrap=1) (<https://canvas.wisc.edu/courses/330348/files/30412095?wrap=1>)  ([https://canvas.wisc.edu/courses/330348/files/30412095/download?download\\_frd=1](https://canvas.wisc.edu/courses/330348/files/30412095/download?download_frd=1)) (a compact reminder of the x86 assembly instructions we expect you to understand)
- [p5 Getting Started](https://canvas.wisc.edu/courses/330348/pages/p5-getting-started) (<https://canvas.wisc.edu/courses/330348/pages/p5-getting-started>) (things to try and tips for how to disassemble an executable to learn about its source)

## REQUIREMENTS

We will test your solution files by running them as shown in the sample output above. It's your responsibility to ensure that your solutions correctly crack each of the four binary safes on the CS Linux lab machines and that your solution text files contain the codes on each line and are correctly named.

## SUBMITTING Your Work

**Leave time before the deadline to complete these steps for submission. It is critical you check that your submission files work as required.**

There is a LATE period after the due date and time. Submitting work at or after the due date and before the end of the LATE period results in your submission being marked late. No submissions or updates to submissions are accepted after the LATE period.

**1.) Submit only the files listed below** under Project p5 in Assignments on Canvas as a single submission. Do not zip, compress, submit your files in a folder, or submit each file individually. All four files must be in your final submission.

1. s1.solution
2. s2.solution
3. s3.solution
4. s4.solution

**Repeated Submission:** You may resubmit your work repeatedly so we strongly encourage you to use Canvas to store a backup of your current work. If you resubmit, Canvas will modify your file names by appending a hyphen and a number (e.g., s1.solution-1.c). This is not a problem for us and you do not have to try get your submission without these numbers. But when you download and retry your solution, you will need to use the new name of the file or change the name of the solution file.

**2.) Verify your submission** to ensure it is complete and correct. If not, resubmit all of your work rather than updating just some of the files.

- **Make sure you have submitted all the files listed above.** Forgetting to submit or not submitting one or more of the listed files will result in you losing credit for the assignment.
- **Make sure the files that you have submitted have the correct contents.** Submitting the wrong version of your files, empty files, skeleton files, executable files, corrupted files, or other wrong files will result in you losing credit for the assignment.
- **Make sure your file names exactly match those listed above.** If you resubmit your work, Canvas will modify your file names as mentioned in **Repeated Submission** above. These Canvas modified names are accepted for grading.

Project p5 (2)			
Criteria	Ratings		Pts
Follow the filename and format requirements -2 points for any missing file or incorrect file name	10 pts Full Marks	0 pts No Marks	10 pts
s1	20 pts Full Marks	0 pts No Marks	20 pts
s2	20 pts Full Marks	0 pts No Marks	20 pts
s3	20 pts Full Marks	0 pts No Marks	20 pts
s4	20 pts Full Marks	0 pts No Marks	20 pts
			Total Points: 90