

Luiz Wakano

Python

Offensive and Defensive Tool Construction

785110

Table of Contents

[Objectives 4](#_Toc31495971)

[Important Information 4](#_Toc31495972)

[Problem 1 5](#_Toc31495973)

[Problem 2 5](#_Toc31495974)

[Problem 3 6](#_Toc31495975)

[Problem 4 6](#_Toc31495976)

[Problem 5 7](#_Toc31495977)

Offensive and Defensive Tool Construction

Python Programming I

Objectives

This lab focuses on the following objectives:

* Analyze the Linux filesystem using Python.
* Explore the use of python in building basic tools to gather information about the filesystem.
* Use variables, expressions and statements in Python.
* Use built-in modules to assist in the development of Python Tools.

# Important Information

All scripts must have the following elements:

1. File and Header comments, which follows the following format:

***# Filename: m##XXX.py***

***# Author: Craig Mac***

***# Course: ITSC203***

***# Details: This exercise checks to see if students read the suggested items or***

***# prior to class or doing the labs.***

***# Resources: https://www.cs.siue.edu/programming-style-guide***

1. Comments on lines where you used some unique computation that might be tricky to comprehend a month later.

***list1 = [x for x in range(20) if x % 4 == 1] # Using list comprehension to ….***

# Problem 1

During exploitation you will often find yourself needing to generate a pattern which can be used to determine where the return address is located on the stack. Write a Python program named **m2p1.py** (module 2, problem 1).

With this exercise you will do the following tasks:

1. Create a tool that generates a non-repeating sequence of alphanumeric characters. Questions to ask while completing this task:
   1. What characters from the ASCII table can I use?
   2. Will uppercase be treated differently from lowercase letters when placed on the stack?
2. Your program should take as input the length of the string. The program will generate a random non-repeating string of that length.
3. Alternately, your program will also be able to take as input a string of any length and return the beginning offset of that pattern in the generated string from step 1.
4. Create a file called **currentbuffer.txt**, which stores the current string to evaluate for the pattern to be checked. When creating the pattern ensure you save it to the file.
   1. You need some approach to guarantee that you do not overwrite the contents of the currentbuffer.txt, until you have checked the pattern against it.
   2. After you have checked the pattern, you can then delete the file as it is no longer needed.

To test your program, simply generate a sequence then select a part of that sequence.

# Problem 2

Write a Python program named **m2p2.py** (module 2, problem 2) that takes a single parameter, a filename.

The program will then display information about the file in the following format:

File Name : <filename>

File Size : 125 bytes

Inode : 2356755

Last Mod : Tues Jan 8 07:00:53 2019

What are inodes: https://www.youtube.com/watch?v=\_6VJ8WfWI4k

# Problem 3

Write a Python program named **m2p3.py** (module 2, problem 3) that dumps the ELF file header and identifies its parameters (use the link for more details https://en.wikipedia.org/wiki/Executable\_and\_Linkable\_Format). Print the parameters using the following format:

File : mytest

Magic : 0x564c457F

Format : 64 bit

Endian : big

Machine : x86-64

To accomplish this task simply compile the following program on your ubuntu VM and use the created executable as your test ELF file.

#include<stdio.h>

int main()

{

printf(“Hello World\n”);

return(0);

}

gcc mytest.c -o mytest

# Problem 4

Write a Python program named **m2p4.py** (module 2, problem 4) that takes a single parameter, a file name and prints a hexadecimal dump of it using the following format:

[00000000]: 23 69 6e 63 6c 75 64 65 20 3c 73 74 64 69 6f 2e #include <stdio.

[00000010]: 68 3e 0a 0a 76 6a 69 64 0a 6d 61 69 6e 28 29 0a h>..void.main()

Total Length 31 (1Fh)

# Problem 5

Write a Python program named **m2p5.py** (module 2, problem 5) that prints all user login/logout information for the past 24 hours. Use the sequence: date, user, where from, sorted in descending time order based on username:

**HINT**: Can Linux command-line tools be accessed from within Python. How can those results be accessed and manipulated?

<https://unix.stackexchange.com/questions/126166/how-to-interpret-all-fields-of-utmpdump-var-log-utmp>

The fields are (member names of struct utmp in parantheses - see man 5 utmp):

* Type of record (ut\_type)
* PID of login process (ut\_pid)
* Terminal name suffix, or inittab(5) ID (ut\_id)
* Username (ut\_user)
* Device name or tty - "/dev/" (ut\_line)
* Hostname for remote login, or kernel version for run-level messages (ut\_host)
* Internet address of remote host (ut\_addr\_v6)
* Time entry was made (ut\_time or actually ut\_tv.tv\_sec)

The possible values for the first field (ut\_type or "type of record") are explained in utmp(5) (6 for example is LOGIN\_PROCESS, or "Session leader process for user login").

utmpdump /var/log/wtmp (execute on a linux system to see the side effect)

**THIS IS ONLY IF YOU WANT TO SEE THE RESULTS USING PYTHON2**

To try this problem using Python2.7.x do the following:

To complete the task, you will complete the following steps

1. Create the following users on your VM (fred, bob, alice, john and joan)
2. Login remotely via ssh with two of the users, you can pick
3. Install Python module **pyutmp**:
   1. **sudo apt update**
   2. **sudo apt install python2-pip**
   3. **pip install pyutmp**

After gathering the information, present the output as shown below:

2016-11-07 14:57:39 fred /dev/pts/24

2016-11-04 16:34:59 bob server.sait.ca

2016-10-02 01:03:45 alice somewhere.google.com