Greg Matthews
March 6th, 2018

CS 543: Homework 4
Secure Operating Systems

1.) Did the Linux program fail? What did the MCP program do?
o (DS = Discontinue (or the more vernacular "Deep Six") – it also shows you the line number which the program was stopped on)
Answer:

No, the linux program did not fail and was able to access different addresses in memory by over indexing the array variable and changing the variable values of x, y, and z. Likewise incrementing the pointer to x and changing the new pointer's value allows you to change variables y, and z, as well as the array named arr.

The MCP program did fail because the system did not allow the code to change the value of an index that is outside the array, particularly on line 440 this failed: ARR[3] := 5;

2.) Is there any way that you can stop this from happening in Linux?
Answer:

Writing code in C does not provide implicit index boundary checking for arrays, and will only raise a segmentation fault when you read/write area of unallocated memory. For the above example, x, y, and z are already allocated and can be changed when going over array index boundaries by incrementing the pointer. However, if one wants to manually include this protection, simply having a conditional that checks if the array index is larger than the length of the array, then raise an exception.

3.) What do you think is happening in the ClearPath MCP operating system?
Answer:

Memory in MCP is segmented, meaning there is no pointers but objects, and therefore you can never go over the address limit of a segmented area of memory in MCP. As a result, we see that from running the program on MCP, there was a run time error stating that there was an invalid index, because that index is not available within the memory segment.

4.) Why did the MCP program not work? (look inside the source file)
Answer:

MCP program did not work because on line 410, the program is trying to call CONVERTDATEANDTIME() with only 5 parameters instead of the needed 6 and received an error due to parameter mismatch. The location of the error is shown here:

RSLT := CONVERTDATEANDTIME(P1, P2, P3, P4, P5);

When the actual function call should look like:

CONVERTDATEANDTIME(WHICH, UD, UT, LD, LT, TZOFF);

5.) How do you think the Linux Program is passing parameters? How do you think MCP is passing parameters?
Answer:

When defining a functions in Linux, specifically in C, one can define it with an unspecified number of parameters, such as the extern char *strptime() function. When the program calls the strptime() function, that function will assume that the correct number of parameters being passed is correctly utilized, and looks into the stack and takes whatever values that were passed in, and assumes they are correct. Therefore any number of parameters can be passed to the function, and the C program compiles and executes, but returns NULL because the function could not interpret the missing or extra parameters that were passed.

MCP makes sure that every function definition has the exact number of parameters clearly defined before the function is being called, therefore any missing or extra parameters being passed are quickly noticed by the system and a parameter mismatch error.

6.) Compare and contrast the two approaches to passing parameters, are they good/bad?
Answer:

In regards to parameter passing in Linux, functions are capable of having a variable number of parameters being passed to it, allowing for huge function flexibility. For example with printf(), you can supply any number of parameters to the function. On the other hand with MCP, all functions have a fixed number of parameters that can be passed into any function, making it more strict in terms of flexibility, but at the same time making functions more reliable because there is never any ambiguity in the number of parameters that need to be passed in. Also, fixed function parameters removes the chances of crashing a program by overflowing the stack by attempting to pass in an arbitrarily large number of parameters.

7.) How does the Linux file system maintain what is in a file as data? Does it matter? Should it matter? Are colors “good enough”?
Answer:

The Linux file system maintains information on a file’s type by providing several bytes at the beginning of each file that represent the file’s type. This value is matched against all stored file types in the directory /etc/magic, or /usr/share/misc/magic to determine what file it is. Using the BASH command “file” does just this and spits out the type of the file you supplied.

For the regular user who does not consider or care about file types, does not have security concerns, and doesn’t have files that could potentially be malicious, then how Linux maintains what is in a file does not matter to them. However this should definitely matter to many others because the way in which Linux maintains what is in a file is by checking the first few bytes of the data. Who’s to say that a malicious programmer is not capable changing these values and make the file appear as something else? Even worse, if someone has access to /etc/magic, they can alter the file type information inside the file. This is also the case with extensions, there is

nothing stopping someone from creating a .txt file that is actually an executable malicious program. Therefore, it should matter how Linux maintains what is in a file.

Colors are good enough for the average user, however for people who need to know exact file types, colors are not good enough because they only give generic information on a files data, e.g., directory, executable, symbolic link. However if you need to know what type of executable you are running for example, then file type information is very crucial.

8.) How do you think the ClearPath MCP program stopped the attack?
Answer:

Starting with running the part3 program on the Linux system, the program was able to overwrite the executable "part1" file, and change it into a raw G3, byte-padded file, making it unable to be executed with the error: "cannot execute binary file." Conversely, ClearPath MCP was able to stop this kind of malicious attack by not allowing the write capability to code or system files, which is evident in the run time error on line 640:

FILE OBJECT/SYMBOL/PART3 I/O ERROR: WRITE ON CODE OR SYSTEM FILE @ (00000640)

Particularly this line in the code was not able to be executed:

WRITE (MCPFILE, OFFSET(P), PRINTBUF);

As a result, MCP is able to increase the security with code and system files because it does not allow them to be overwritten, which is not the case with a Linux system.

9.) Can we do that in Linux?
Answer:

This type of attack can be reduced by removing write privileges from all users except the owner by using chmod 700 file_name. However if someone was able to access the owner account, they would still be able to change chmod privileges and attack someone's code and system files. The best way to secure files from being modified is using Chattr (change attribute) which is an immutable file system attribute of Linux. Chattr protects files from being written into by any user, including root user by calling: chattr +i file_name. The root user however, is capable or removing the restriction by calling: chattr -i file_name, however this decreases the number of possible account breaches from owner level using chmod, to root level, meaning a hacker would need access to root to be able to remove write protection from chattr.

10.) Compare the security of the Linux environment to the security of the ClearPath MCP environment – overall conclusions?
Answer:

After getting to explore the ClearPath MCP and the Linux Operating System, there is no question that MCP has the higher level of security in many areas such a data storage, linking program functions, and write protection for code and system files. Linux allows users to have more flexibility in many areas that MCP is more restrictive on, but at the cost of security. MCP's

filesystem always shows exactly what's inside each file by showing the filetype, and makes sure there is code-file modification protection so executables cannot be maliciously broken. Linux can show file type information, however there is nothing stopping someone from overwriting the first few bytes of the file that represents the filetype, and altering its functionality. MCP also has very strict program linkage, in that a function will only be linked to when there correct number and type of parameters are passed, which reduces the flexibility of having variable function parameters, but results in a more reliable system that always provides the expected output. Another important area is how data is stored within MCP versus Linux. MCP uses objects rather than pointers, and utilizes segmented memory, meaning it is impossible over index areas of memory and attempt to access addresses out of bounds, whereas in Linux the use of pointers allows you to increment them past your bounds, and overwrite allocated memory areas. In conclusion, MCP trades flexibility and variability for higher system security, and higher reliability, whereas Linux is vice versa.

(Bonus.) ClearPath MCP environment feedback
Answer:

Although the user interface takes some getting used to, the system has limited modification capabilities, and has an interesting way that code is represented on the CANDE interpreter, it's very eye opening to see the degree of security in so many areas to file system management, data storage, etc, especially when comparing to the Linux Operating System. The most desirable trait that I see is a huge strength of the system, is that nothing is hidden to the user in terms of what is in a file, you always know whether the file is data, executable, whatever it is, without being concerned that it could be falsified or modified. The built-in protection from writing to a code or system file, which is immensely important and should be focused more in the Linux Operating System. On a final note, the biggest gripe I had with the system was the way in which you call the CANDE interpreter, the non-intuitive use of ?+s, and switching between action and choice lines is cumbersome.