Assignment - PSO 3

Purdue CS426 - Computer Security - Prof. Spafford

Harris Christiansen - February 21, 2018
christih@purdue.edu

Problem 1

Read the following:

- Linux Permissions: https://wpollock.com/AUnix1/FilePermissions.htm
- Permissions Calculator: http://permissions-calculator.org/
- chmod: https://linux.die.net/man/1/chmod
- chown: https://linux.die.net/man/1/chown
- Add User To Group: https://www.cyberciti.biz/faq/howto-linux-add-user-to-group/
- Optional Reading Access Control Lists: https://wiki.archlinux.org/index.php/Access_Control_Lists

Why does a user need read privileges on a file interpreted by an interpreter (e.g Python, BASH, etc.)? (10 pts.)

- Because interpreted languages are interpreted and compiled at runtime, the interpreting program (python, php, etc) must be able to read in the file it wishes to execute. The interpreter reads this file into memory as a string, and then parses and executes it.

If the setuid and setguid bit are set on a BASH script, will a standard Linux distributions change the Effective UID or Effective GUID? Why or why not? (20 pts.)

- The setuid (SUID) flag causes an executed program to be owned by the owner of the file, instead of the calling user. Thus, the effective UID is changed for executed programs if the SUID flag is true.
- The setguid (SGID) flag causes an executed program to belong to the group of the file, instead of the group of the calling user. Thus, the effective GUID is changed for executed programs if the SGID flag is true.

Why does a user need executable permissions on all folders in the path to an executable? (10 pts.)

- In order to descend into any folder, and read/execute any of it's contents, a user must have execute permissions for that folder.
- Because of this, for a user to reach the file in-order to execute it, they must have execute permissions for every parent folder (recursively).

Problem 2

Read the following (From Spaf's Reading List): (40 pts)

- An Empirical Study of Reliability of Unix Utilities: http://ftp.cs.wisc.edu/paradyn/technical_papers/fuzz.pdf
- Fuzz Revisited: http://ftp.cs.wisc.edu/paradyn/technical_papers/fuzz-revisited.pdf

How can fuzz testing identify vulnerabilities in a program?

- Fuzz testing subjects programs to a stream of random input, testing reliability, crashes, and hangs. These crashes and hangs, and their respective sample/core dump, can expose areas in a program that may be vulnerable to corruption or attack. Many crashes and hangs often occur due to incorrect parsing, incorrect validation, or poorly constructed loops.

What were some of the causes of program crashes/hanging outlined in the article?

- Four types of causes were outlined in the article:
  - Pointer/Array: Failure to check for array/buffer bounds - which can even expose a program to attacks as unintentional memory can be manipulated.
  - Dangerous Input Functions: Use of input functions that read an unspecified length (such as `gets()`) are known to cause major security vulnerabilities.
  - Signed Characters: Not paying attention to type lengths, and signed vs unsigned types was a source of corruption.
  - End-of-File Checks: Similar to failing to check array bounds, failing to check for end-of-file can cause crashes/hangs, as well as unintended execution/manipulation.

Problem 3

Read/Watch the following: (40 pts)

- Video on stack smashing: https://www.youtube.com/watch?v=1S0aBV-Waeo
- Stack Guard: ftp://gcc.gnu.org/pub/gcc/summit/2003/Stackguard.pdf

What is a stack canary and how does it prevent stack smashing attacks?

- Solution

Describe one way a stack canary can be bypassed to gain access to the system.

- Solution

From the video, describe how and why the stack smash attack works. Why is the coder able to gain root access to the machine?

- Solution

What is another protection mechanism other than a stack canary that is used to prevent buffer overflow attacks? Explain why it prevents an attack from succeeding.

- Solution

Problem 4

In a few sentences per item, describe the following types of Malware, how they infect machines, and whether they can spread. (35 pts.)

Trapdoor

- Solution

Trojan Horse

- Solution

Viruses

- Solution

Zombies

- Solution

Rootkits

- Solution

Speculate how each of the types of Malware listed above get onto the following devices. Once on the machine how can the malware spread? If the malware cannot get onto the machine or cannot spread to other machines, state why. Explain and justify your answers. (50 pts.)

Webserver

- Solution

Smartphone

- Solution

Microwave

- Solution

2018 Mercedes S63 AMG

- Solution

1974 Honda Civic

- Solution

Problem 5

Read the following:

- Reference Monitor: https://pdfs.semanticscholar.org/c93c/234c9a7698038caf317a97405c53144bf354.pdf
- TPM Summary: https://trustedcomputinggroup.org/trusted-platform-module-tpm-summary/
- Trusted Platform Module Quick Tutorial: https://link.springer.com/content/pdf/10.1007%2F978-1-4302-6584-9_3.pdf

What are the 3 properties needed for an effective Reference Monitor? Explain each property. (15 pts.)

- Solution

What functionality does a TPM chip usually contain? (20 pts.)

- Solution

Describe two applications that use a TPM chip and the part the TPM chip plays. (30 pts.)

- Solution

Problem 6

Skim the following:

- readelf: https://linux.die.net/man/1/readelf
- objdump: https://linux.die.net/man/1/objdump
- gcc: https://linux.die.net/man/1/gcc
- (Skim More Closely Pages 1-8 to 1-15) ELF Format: http://www.cs.cmu.edu/afs/cs/academic/class/15213-f00/docs/elf.pdf
- Journey To the Stack: http://duartes.org/gustavo/blog/post/journey-to-the-stack/

What is does the register ebp typically hold and what ebp value is pushed on the stack in a normal stack frame (x86 32 bit)? (10 pts.)

- Solution

What does the following x86 assembly instruction do (note destination address comes second here)? (10 pts.)

lea -0x20(%ebp),%eax

- Solution

Coding Problem 1

Navigate to the folder `Coding Problem 1/` enclosed with this assignment. There you will find a Linux executable file (ELF, 32bit, Optimization Level = 0) that has several vulnerabilities you will exploit, "buf overflow 1." Run the program and enter a password to familiarize yourself with the program (./buf overflow 1).

For this problem you will need to answer the following questions using readelf and objdump or other programs of your choice.

- In the source code below, we can see that the programmer hardcoded a password. Use one of the tools above to disassemble the binary and try to guess the password amongst the strings present. Include a screenshot of the section in the executable where the password is located. Include a screenshot entering the password and the successful authentication into the program using the password. What is the correct password? (20 pts.)
- Looking carefully at the source code or disassembled file identify a potential buffer overflow and how it can be used to bypass the password authentication code. (30 pts.)

Do the following (140 pts.)

Run the objdump -D -s buf overflow 1 and navigate to the disassembled code for the authenticate function.

- Solution

Include the disassembled output of the authenticate function.

- Solution

In relation to ebp, where is the variable pass stored (Hint: Use the initial value of pass to find the instruction)? Explain how you figured this out.

- Solution

In relation to ebp, where is the variable buff stored (Hint: Use the call to gets( ) as a reference point)? Explain how you figured this out.

- Solution

How many bytes long is the buffer that holds the entered password? Explain how you determined this.

- Solution

What is the minimum number of characters a user has to enter in order to overflow the buffer and write a nonzero value to the variable pass (Hint: the null terminator in a string has a value of 0)?

- Solution

Use a hexeditor like Bless to open the binary file and search for the correct password found at the start of this exercise. Change the last 4 characters of the password to 2018 and save the binary. Try to enter the correct password, but with the last 4 characters equal to 2018. Did it work? Include a screenshot of the program running with your entry attempt.

- Solution

Briefly explain how to eliminate the vulnerabilities in this program.

- Solution

Problem 6 Final Question

Is it a good idea to store sensitive information as a plaintext character array? What are some alternatives? How does the Linux login program handle storing user passwords? (20 pts.)

- Solution

Problem 7

Navigate to folder `Coding Program 2/`. There you find some code

- buf overflow 2 - Binary of the source code.
- Input Gen - Folder containing a makefile and source to generate binary output to be piped into buf overflow 2

Run the binary to get familiar with its operation. Take a look at the source for buf overflow 2 on the next page. You can see that there is again a vulnerability. Also you will see that function2 does not get called under normal operation of the code. You will do some stack smashing to execute function2.

Do/Answer the following: (150 pts.)

Use objdump to disassemble the binary. Navigate to portion of output for function1.

- Solution

In relation to ebp, where is the beginning of the character buffer used to store the string?

- Solution

What is the minimum number of bytes you need to write to the character buffer in order to overwrite the return address?

- Solution

How many bytes are in an address for a 32 bit binary? What is the minimum number of addresses you need to write from the beginning of the character array to overwrite the return address?

- Solution

What is the address of function2?

- Solution

Modify the file, Input Gen/main.c to rewrite the return address with the address for function2 when function1 is called. Run make to compile the binary for the input generator, Input Gen/input gen. Pipe the output of the input generator to the original program. To do this make sure your working directory is Coding Program 2/ and then run the command, Input Gen/input gen | ./buf overflow 2 .

- Solution

Include your output of the programming calling function2 (Note it is ok if an error occurs after function2 runs). Include the full source code for your input generator and explain why the attack succeeded.

- Solution

Problem 7 Final Question

List 3 other unsafe C functions in std.h, stdio.h, or string.h and alternatives that protect against buffer overflow/stack smashing attacks. (15 pts.)

- Solution

Problem 8

Question

Question

- Solution

Question

- Solution

Question

- Solution

Problem 9: Extra Credit

Question

Question

- Solution

Question

- Solution