CSE 523S

Studio: PPT and practice with binaries

Overview

Today we will explore a practical example of a security vulnerability that involves the three components of People, Process, and Technology.

Keep detailed notes of your answers below (place your comments in between the provided horizontal lines); you may want to refer to these when working on future assignments.

Pay attention to formatting issues when copying lines in this doc to the terminal. You may have to re-type characters such as ‘,’.

Part 1: People

For this part, you will be working on your SEED Labs VM, so start that machine now. If you don’t have access to a working SEED Labs VM, then you will need to either pause here to get one up and running, or work in a group with someone who has the VM working and use their machine together.

1.1 date

In this exercise, we will be working with the Linux utility program called **date**. Enter the following command lines in a terminal window.

date

date --date=”4 hours ago”

date --date=”2 years ago” +%Y%m%d

Where is the **date** program in the filesystem? Show how you found it below (i.e. show the command line and its output).

电脑萤幕的截图

描述已自动生成

-------------------------------------------------------------------------------------

Date program is in /usr/bin. In Linux, we always use which to show the direction of the program. In fact, /usr/bin is an directory in PATH environment variable, when we type a program, linux will try to search the program based on PATH.

-------------------------------------------------------------------------------------

Briefly describe what the program does.

-------------------------------------------------------------------------------------

Based on man, date print out the system date and time in a default pattern:

week month day hour:minute:second AM/PM timezone year

When using --date=”4 hours ago”, it will print out the date and time 4 hours ago in default pattern .

When using --date=”4 years ago” +%Y%m%d, it will print out the date in YearMonthDay format 4 years ago.

-------------------------------------------------------------------------------------

1.1 my\_date

Download my\_date.zip from Canvas to your VM, extract and run it. Take a few minutes to explore it. (If you have an EIT-provided AWS SEED VM, then you will need to use the scp utility on your terminal command line to copy my\_date.zip from your local laptop to your remote VM.)

Briefly describe what the program does.

-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------

Are the two programs similar? Identical? Briefly explain your answer. (For now, don’t use diff tools. Simply try to examine the programs as a user.)

-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------

Part 2: Process

How can you, as a user, protect yourself from downloading malicious files? Any initial thoughts?

-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------

Go over these three articles, published between 11/18-11/20 2019:

* November 18, 2019: <https://github.com/monero-project/monero/issues/6151>
* November 19, 2019: <https://bartblaze.blogspot.com/2019/11/monero-project-compromised.html>
* November 20,2019: <https://thehackernews.com/2019/11/hacking-monero-cryptocurrency.html>

Would you answer the previous question differently now? Please explain.

-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------

Part 3: Technology

We can use file signatures and checksums to validate the integrity of the files and programs we download.

A **checksum** is “a small-sized block of data derived from another block of digital data for the purpose of detecting errors that may have been introduced during its transmission or storage.” [[1]](#footnote-1) In other words, checksums help preserve data integrity by detecting whether data has been modified.

A hash function often serves as a checksum for digital data, and “secure” or “cryptographic” hash functions have been designed to be robust to intentional data modification (e.g. by an attacker) as well as benign errors in transmission or storage.

One common secure hash function is the SHA-256 algorithm, which you can read more about [here](https://en.wikipedia.org/wiki/SHA-2) if you are interested.

Run the following command on both the date and my\_dateprogram files:

sha256sum <program\_name>

Paste your results here (show command and output):

-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------

Now, use ‘vimdiff’ to find the differences between the two files:

vimdiff <(strings /path/to/normal/date) <(strings my\_date)

Scroll all the way down to near the end of the files, where the text strings are. What are the differences you see in the program text strings? Why would an attacker introduce those differences?

-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------

Now, let’s think like an attacker. Can you modify the my\_date program to use another domain (e.g. “.com” instead of “.net”)? If yes, briefly describe below what you did to modify the program. If you don’t know how to do that, move on to the next section, where we’ll give you some hints and introduce you to a few helpful tools.

-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------

Part 4: Let’s get our hands dirty!

Let’s start with **objdump.**  Briefly describe what the **objdump** program does.

-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------

Enter the following command at the terminal, replacing {loc} with the location you found above for the built-in date program (you probably want to find-replace {loc} for your location so that you can copy and paste directly). This will pipe the output of objdump -d into a program called less. Here are some quick and easy commands to learn to make less easier to navigate. Note that, much like Vi(m) in “command mode,” less interprets your keypresses as commands at the output window.

Vi(m)/less commands: Scroll up (J), scroll down (K), start a forwards search for a string (/), start a backward search for a string (?), quit (q), scroll down one window (SPACE), scroll forwards ½ window (d), and scroll back ½ window (b). Now you know enough to be quick using less!

Explain below what you see when you enter the following command, using only a few sentences.

objdump -d {loc}/date | less

-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------

Enter the following command at the terminal, contrast this output with the previous output, and explain what the -xtrds switches mean.

objdump -xtrds {loc}/date | less

-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------

Enter the following command at the terminal, and note the output below.

objdump -xtrds {loc}/date | grep –A 2 bugs

-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------

Make a copy of **date** in your home directory. Launch either **ghex** or **bless** . Briefly describe the tool you chose and its purpose below. (If you have an EIT-provided AWS SEED VM, skip the remainder of this part and proceed to Part 5.)

-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------

Open your local copy of date using **bless** or **ghex**, and search for the text ‘west’. In the right-hand column, you should see the string “west”, and the hex numbers “77 65”. Change these numbers to “65 61”. Save and quit the file. Go back to the terminal and run (the local version of) “./date --help”. Do you see the differences? What is it? What does “65 61” represent?

-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------

Now go back to the “attacker” question at the end of part 3, and complete the question if you were not able to before now.

Part 5: Debug!

In this section, we will practice an important skill: debugging!

5.1 Readelf

First, let’s look at a useful tool: readelf. ELF is the object file format used in Linux and many other systems. Execute the following commands at the terminal, and explain what each one shows you. If you’d like, you can keep a copy of each output in this file for future reference. For this Part, {loc} now refers to the location of the my\_date program on your system.

readelf -l {loc}/my\_date

readelf -S {loc}/my\_date

readelf -W -s {loc}/my\_date

readelf -x 16 {loc}/my\_date

-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------

5.2 Using gdb

At the terminal, enter the following

gdb

(gdb) help

Use the interactive help feature to explore gdb’s options.

5.3 Examining processes

At the terminal, enter the following.

gdb my\_date

(gdb) run

(gdb) quit

Copy your output below.

-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------

Try the following commands to get a feel for the tool.

gdb my\_date

(gdb) set args “--help”

(gdb) break \_\_libc\_start\_main

(gdb) run

(gdb) frame

(gdb) bt

(gdb) info frame

(gdb) info registers

(gdb) x /16xw $esp

Explore other registers and memory locations. What can you say about where the code and stack are located?

-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------

Now try the following.

gdb my\_date

(gdb) set args “--version”

(gdb) break \_\_libc\_start\_main

(gdb) run

(gdb) maintenance info sections

note the start address of the .rodata section

(gdb) x/20s {.rodata address}

note the start address of the string “GNU coreutils” as {start}

(gdb) set \*{start} = 0x20554c42

(gdb) c

What does the code above do? Can you modify it to do something else that demonstrates that you understand how it works? Did you run into a snag? Provide your explanation and your modification below. A reference like this might be helpful: <http://www.asciitable.com>

-------------------------------------------------------------------------------------

-------------------------------------------------------------------------------------

**COMPLETE**

That’s all! Please submit a .pdf version of this file to Gradescope before the assignment deadline.

1. <https://en.wikipedia.org/wiki/Checksum> [↑](#footnote-ref-1)