People, Process and Technology

Overview

Today we will explore a practical example of a security vulnerability that involves both People, Process, and Technology.
Keep detailed notes of your answers below (place your comments in between the provided horizontal lines); you will likely need 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 signs such as “,’.

Part 1: People

For this part, you will be working on your SEEDLabs 16.4 VM, so start that now. If you didn’t complete HW1 and don’t have your SEEDLab machine working, then you will need to pause here and work on the homework first.

1.1 date

In this exercise, we will be working with the Linux utility **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 program **date** in the filesystem? Show how you found it below (ie, show the command line and its output - you may want to take a look at your cheat sheet from last week)

[02/02/22]seed@VM:Byeongchan$ whereis date
date: /bin/date /usr/share/man/man1/date.1.gz

Briefly describe what the program does.

It shows the date of the system.

1.1 my_date

Download my_date.zip from Canvas to your VM, extract and run it. Take a few minutes to explore it.

Briefly describe what the program does.

It’s almost like date command we explore previous question.

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)

I think it is almost similar. I looked through ‘help page’ and the only thing different is command itself. I executed two command and both return same result.

Part 2: Process

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

---

1. When opening an email, be careful not to open attachments
2. Be careful not to click the 'Next' button without looking at. Some programs they are automatically launched when you visit some web site.

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?

Wow,, it means that whenever I download something, I should check the hashes of the binary file. But I think not every user can do that in real situation.
To summarize, when you installing which is really important to your money, check hashes of the file as well.

Part 3: Technology

We can use file signatures and checksums to validate the integrity of the files and programs we download.
Checksum: “a digit representing the sum of the correct digits in a piece of stored or transmitted digital data, against which later comparisons can be made to detect errors in the data.”

A common way to verify data integrity is to use the md5 message-digest algorithm.

Run the following command on both date and my_date

Md5sum <*program_name*>

Paste your results here:

[02/02/22]seed@VM:Byeongchan$ md5sum my_date
09a78eda4ef5d3d56cf533d68f2726c1 my_date

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

vimdiff /bin/date my_date

What are the differences? Why would a hacker do that?

---

org -> net
I think it’s really easy to fool users when they compromised popular command. Because the users execute it all the time without questioning.

Now, switch to think like a hacker. Can you modify the ‘my_date’ program to use another domain? (Let’s say, “.com” instead of “.net”). If yes, then briefly describe what you did between the lines. If you don’t know how to do that, move on to the next section, we’ll give you some hints and introduce you to a few tools.

Is it just modify the file after opening with ‘vi’ and edit it? Does it work on binary file? I'm not sure

---

Part 4: Let's get our hands dirty!

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

Objdump is an executable program on Unix-like system. We can use it as a disassembler to view an executable in assembly form.

Enter the following command at the terminal, replacing {loc} with the location you found above (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)`, `less` interprets your keypresses as commands at the output window.

Scroll up (J), scroll down (K), start a forwards search for a string (/), start a backward search for a string (?), quit less (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 what you see below, using only a few sentences.

objdump -d {loc}/date | less

I can see hexacode and assembly code by function.

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

I found the man page but the meaning is not sure to me.
[-x|--all-headers]
[-t|--syms]
[-r|--reloc]
[-d|--disassemble[=symbol]]
[-s|--full-contents]
Not sure, but the xtrds option is more human readable.

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

objdump -xtrds {loc}/date | grep bugs

[02/02/22]seed@VM:Byeongchan$ objdump -xtrds /bin/date | grep bugs
 8053490 6e206275 67732074 6f203c68 7474703a n bugs to <http:

Make a copy of **date** in your home directory. Launch either **ghex** or **bless** . Briefly describe the tool you chose and its purpose below.

Both of them show hexadecimal of the 'date' command in graphic interface.

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?

---

77 65 means 'we'
65 61 means 'ea'
[02/02/22]seed@VM:Byeongchan$ date --help | grep east
[02/02/22]seed@VM:Byeongchan$ ./date --help | grep east
Show the time on the east coast of the US (use tzselect(1) to find TZ)
[02/02/22]seed@VM:Byeongchan$

Now go back to the "hacker" question at the end of part 3. Can you modify the program to point to ".com" instead of ".org"? (or instead of ".net" if you worked on my_date)?

Yes, now I can do it. but just changing the text,, is it worth to fool users?

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.

readelf -l {loc}/date
readelf -S {loc}/date
readelf -W -s {loc}/date
readelf -x 16 {loc}/date

---

readelf -l {loc}/date : it shows program headers
readelf -S {loc}/date : it shows section headers
readelf -W -s {loc}/date : it shows symbols
readelf -x 16 {loc}/date : it shows in hexadecimal

I looked it throguth man page, but I'm not sure about what they exactly mean. Please cover it on the class.

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 date
(gdb) run
(gdb) quit
Copy your output below.

[02/02/22]seed@VM:Byeongchan$ gdb date
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1

Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from date...(no debugging symbols found)...done.
gdb-peda$ run
Starting program: /home/seed/date
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/i386-linux-gnu/libthread_db.so.1".
Wed Feb 2 11:17:06 CST 2022
[Inferior 1 (process 3791) exited normally]
Warning: not running or target is remote
gdb-peda$ quit

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

---

stack address shown below, esp points stack address
code address show below, eip points <__libc_start_main>

gdb-peda$ info register
eax   0xb7fff918 0xb7fff918

ecx 0xbfffeda4 0xbfffeda4
edx 0xb7fea780 0xb7fea780
ebx 0xb7fff000 0xb7fff000
esp 0xbfffed7c 0xbfffed7c
ebp 0x0 0x0
esi 0x2 0x2
edi 0x8049b09 0x8049b09
eip 0xb7d82540 0xb7d82540 <__libc_start_main>

Now try the following.

gdb 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? Did you run into a snag? Provide your explanation and your modification below. A reference like this might be helpful: http://www.asciitable.com

---

Code above replaces ‘GNU’ -> ‘BLU’
x/20s means ‘display memory contents in 32bit string value at designated address’
0x 20 55 4c 42
20 : space
55 : U
4c : L
42 : B

gdb-peda$ x/20s 0x80521a0
0x80521a0: "\003"
0x80521a2: ""

```
0x80521a3:    ""
0x80521a4 <_IO_stdin_used>:        "\001"
0x80521a6 <_IO_stdin_used+2>:    "\002"
0x80521a8:    "time %s is out of range"
0x80521c0:    "["
0x80521c2:    "test invocation"
0x80521d2:    "Multi-call invocation"
0x80521e8:    "sha224sum"
0x80521f2:    "sha2 utilities"
0x8052201:    "sha256sum"
0x805220b:    "sha384sum"
0x8052215:    "sha512sum"
0x805221f:    "\n%s online help: <%s>\n"
0x8052236:    "GNU coreutils"
0x8052244:    "en_"
0x8052248:    "%a %b %e %H:%M:%S %Z %Y"
0x8052260:    "/usr/share/locale"
0x8052272:    "--rfc-3339"
gdb-peda$ set *0x8052236 = 0x20554c42
gdb-peda$ c
Continuing.
date (BLU coreutils) 8.25
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Written by David MacKenzie.
[Inferior 1 (process 3893) exited normally]
Warning: not running or target is remote
gdb-peda$
```

COMPLETE