CSE 5203

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 and screen shots 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 single-quote 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. Take a screen shot of the terminal window that includes your successful command line(s).

```
[01/28/26]seed@Kaiyuan:~$ date
Wed Jan 28 16:57:00 UTC 2026
[01/28/26]seed@Kaiyuan:~$ date --date="4 hours ago"
Wed Jan 28 12:57:09 UTC 2026
[01/28/26]seed@Kaiyuan:~$ date --date="2 years ago" +%y%m%d
240128
[01/28/26]seed@Kaiyuan:~$ which date
/usr/bin/date
[01/28/26]seed@Kaiyuan:~$ ls -l /usr/bin/date
-rwxr-xr-x 1 root root 108920 Sep  5  2019 /usr/bin/date
[01/28/26]seed@Kaiyuan:~$
```

Briefly describe what the program does.

The date program displays the current system date and time. It can also format the displayed time based on the options.

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.

Display the current system date and time. It can also format the displayed time based on the options.

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.)

It's similar. When I use the command "./my_date", it prints "Wed Jan 28 17:02:21 UTC 2026" which is the same as "date". And it also prints the same result with command "date --date="4 hours ago"" and "date --date="2 years ago" +%Y%m%d"

Part 2: Process

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

I only download software from trusted and official sources. I should be cautious of unexpected downloads, email attachments or links from unknown senders. Keeping my operating system up to date can also help reduce the risk of malware.

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.

Yes, it will be different. Now I should verify software integrity by checking cryptographic hashes or digital signatures and prefer building software from source. Articles highlights that protecting against malicious downloads requires both trusting reputable sources and verifying that the downloaded files have not been tampered with.

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] 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 if you are interested.

Run the following command on both the `date` and `my_date` program files:

```
sha256sum <program_name>
```

Paste your results here (show command and output in a screen shot):

```
[01/28/26]seed@Kaiyuan:~$ sha256sum /usr/bin/date
a1cb3238a537c39e58c0f25c41e75d22d82b9e92d433185cd00438ba5e8e25a5  /usr/bin/date
[01/28/26]seed@Kaiyuan:~$ ls
Desktop  my_date  my_date.zip
[01/28/26]seed@Kaiyuan:~$ sha256sum my_date
930103a8b531f05143d18efc44c69c8734d1cb1234de1dd9e8752dc746986546  my_date
[01/28/26]seed@Kaiyuan:~$
```

Now, use 'vimdiff' and 'strings' to find the differences between the two files:

```
vimdiff <(strings /path/to/normal/date) <(strings my_date)
```

[1] https://en.wikipedia.org/wiki/Checksum

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?

The normal date binary contains many detailed error messages and strings related to date parsing, time zones and input validation, such as warnings and parsing errors. In contrast, my_date is missing most of these descriptive strings.
An attacker may hide malicious functionality in the file to minimize error output, or make the program's bad behavior harder to analyze. Also, without input validation, malicious actions can be more easily to inject into the program.

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.

I have no idea now.

Part 4: Let's get our hands dirty!

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

'Objdump' is a tool used to display information about object files and executables.

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
```

I see the disassembled machine code of the date program. The output shows memory addresses, assembly instructions and function labels that represent the internal structure of the executable. Using 'less' allows me to scroll through and search the disassembly more easily.

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
```

'objdump -d 'output mainly shows the disassembled machine instructions. And this command displays additional structural information about the binary. The output includes details like file headers, symbol tables, relocation entries and section information.

Enter the following command at the terminal, and include the screen shot output below.

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

```
[01/28/26]seed@Kaiyuan:~$ objdump -xtrds /usr/bin/date | grep -A2 bugs
grep: bugs: No such file or directory
[01/28/26]seed@Kaiyuan:~$
```

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!

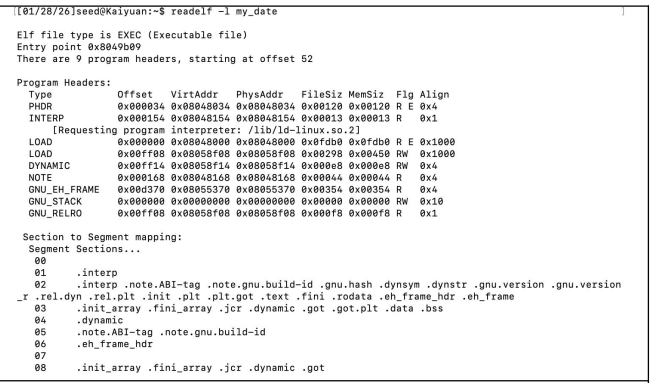
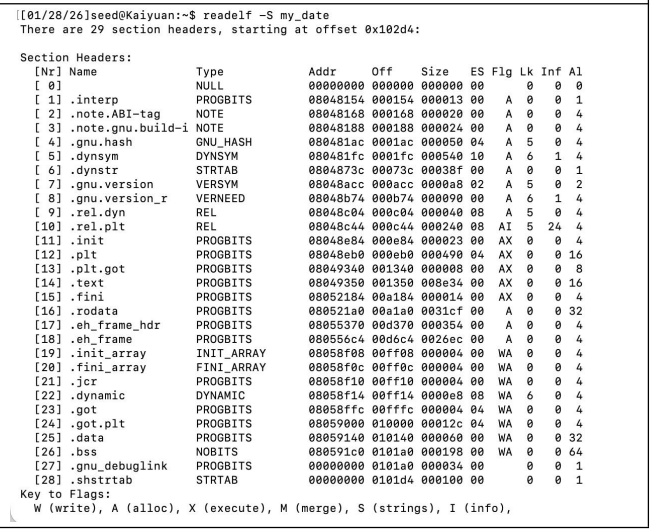
In this section, we will practice an important skill: debugging!

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
```

---

readelf -l my_date	displays the program headers, showing how the executable is mapped into memory at runtime.	
readelf -S my_date	lists the section headers of the binary	

readelf -W -s my_date	displays the symbol table, including function and variable symbols used by the program	[01/28/26]seed@Kaiyuan:~$ readelf -W -s my_date Symbol table '.dynsym' contains 84 entries: Num: Value Size Type Bind Vis Ndx Name 0: 00000000 0 NOTYPE LOCAL DEFAULT UND 1: 00000000 0 FUNC GLOBAL DEFAULT UND __ctype_toupper_loc@GLIBC_2.3 (2) 2: 00000000 0 FUNC GLOBAL DEFAULT UND unsetenv@GLIBC_2.0 (3) 3: 00000000 0 FUNC GLOBAL DEFAULT UND strcmp@GLIBC_2.0 (3) 4: 00000000 0 FUNC GLOBAL DEFAULT UND open64@GLIBC_2.1 (4) 5: 00000000 0 FUNC GLOBAL DEFAULT UND fflush@GLIBC_2.0 (3) 6: 00000000 0 FUNC GLOBAL DEFAULT UND _exit@GLIBC_2.0 (3) 7: 00000000 0 FUNC GLOBAL DEFAULT UND free@GLIBC_2.0 (3) 8: 00000000 0 FUNC GLOBAL DEFAULT UND memcpy@GLIBC_2.0 (3) 9: 00000000 0 FUNC GLOBAL DEFAULT UND mbsinit@GLIBC_2.0 (3) 10: 00000000 0 FUNC GLOBAL DEFAULT UND clock_gettime@GLIBC_2.17 (5) 11: 00000000 0 FUNC GLOBAL DEFAULT UND fclose@GLIBC_2.1 (4) 12: 00000000 0 FUNC GLOBAL DEFAULT UND time@GLIBC_2.0 (3) 13: 00000000 0 FUNC GLOBAL DEFAULT UND fseeko64@GLIBC_2.1 (4) 14: 00000000 0 FUNC GLOBAL DEFAULT UND memcmp@GLIBC_2.0 (3) 15: 00000000 0 FUNC GLOBAL DEFAULT UND gettimeofday@GLIBC_2.0 (3) 16: 00000000 0 FUNC GLOBAL DEFAULT UND dcgettext@GLIBC_2.0 (3) 17: 00000000 0 FUNC GLOBAL DEFAULT UND __stack_chk_fail@GLIBC_2.4 (6) 18: 00000000 0 FUNC GLOBAL DEFAULT UND textdomain@GLIBC_2.0 (3) 19: 00000000 0 FUNC GLOBAL DEFAULT UND __getdelim@GLIBC_2.0 (3) 20: 00000000 0 FUNC GLOBAL DEFAULT UND iswprint@GLIBC_2.0 (3) 21: 00000000 0 FUNC GLOBAL DEFAULT UND fopen64@GLIBC_2.1 (4) 22: 00000000 0 FUNC GLOBAL DEFAULT UND fwrite@GLIBC_2.0 (3) 23: 00000000 0 FUNC GLOBAL DEFAULT UND tzset@GLIBC_2.0 (3) 24: 00000000 0 FUNC GLOBAL DEFAULT UND __ctype_get_mb_cur_max@GLIBC_2.0 (3) 25: 00000000 0 FUNC GLOBAL DEFAULT UND __fpending@GLIBC_2.2 (7) 26: 00000000 0 FUNC GLOBAL DEFAULT UND mbrtowc@GLIBC_2.0 (3) 27: 00000000 0 FUNC GLOBAL DEFAULT UND __cxa_atexit@GLIBC_2.1.3 (8) 28: 00000000 0 FUNC GLOBAL DEFAULT UND error@GLIBC_2.0 (3) 29: 00000000 0 FUNC GLOBAL DEFAULT UND getenv@GLIBC_2.0 (3)
readelf -x 16 my_date	dumps the raw hexadecimal contents of section number 16	[01/28/26]seed@Kaiyuan:~$ readelf -x 16 my_date Hex dump of section '.rodata': 0x080521a0 03000000 01000200 74696d65 20257320 ....time %s 0x080521b0 6973206f 7574206f 66207261 6e676500 is out of range. 0x080521c0 5b007465 73742069 6e766f63 6174696f [.test invocatio 0x080521d0 6e004d75 6c74692d 63616c6c 20696e76 n.Multi-call inv 0x080521e0 6f636174 696f6e00 73686132 32347375 ocation.sha224su 0x080521f0 6d007368 61322075 74696c69 74696573 m.sha2 utilities 0x08052200 00736861 32353673 756d0073 68613338 .sha256sum.sha38 0x08052210 3473756d 00736861 35313273 756d000a 4sum.sha512sum.. 0x08052220 2573206f 6e6c696e 65206865 6c703a20 %s online help: 0x08052230 3c25733e 0a00474e 5520636f 72657574 <%s>..GNU coreut 0x08052240 696c7300 656e5f00 25612025 62202565 ils.en_.%a %b %e 0x08052250 2025483a 254d3a25 5320255a 20255900 %H:%M:%S %Z %Y. 0x08052260 2f757372 2f736861 72652f6c 6f63616c /usr/share/local 0x08052270 65002d2d 7266632d 33333339 002d2d69 e.--rfc-3339.--i 0x08052280 736f2d38 36303100 545a3d55 54433000 so-8601.TZ=UTC0. 0x08052290 44617669 64204d61 634b656e 7a696500 David MacKenzie. 0x080522a0 65787472 61206f70 6572616e 64202573 extra operand %s 0x080522b0 00545a00 7374616e 64617264 20696e70 .TZ.standard inp 0x080522c0 75740069 6e76616c 69642064 61746520 ut.invalid date 0x080522d0 25730063 616e6e6f 74207365 74206461 %s.cannot set da 0x080522e0 74650066 696c6500 72656665 72656e63 te.file.referenc 0x080522f0 65007266 632d3832 32007266 632d3238 e.rfc-822.rfc-28 0x08052300 32320073 65740075 63740075 74630075 22.set.uct.utc.u 0x08052310 6e697665 7273616c 0068656c 70007665 niversal.help.ve 0x08052320 7273696f 6e00686f 75727300 6d696e75 rsion.hours.minu 0x08052330 74657300 7365636f 6e647300 6e730000 tes.seconds.ns.. 0x08052340 54727920 27257320 2d2d6865 6c702720 Try '%s --help' 0x08052350 666f7220 6d6f7265 20696e66 6f726d61 for more informa 0x08052360 74696f6e 2e0a0000 55736167 653a2025 tion....Usage: %

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. Copy-paste the text rather than a screen shot.

```
[01/28/26]seed@Kaiyuan:~$ gdb my_date
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.2) 9.2
Copyright (C) 2020 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 "x86_64-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"...
/opt/gdbpeda/lib/shellcode.py:24: SyntaxWarning: "is" with a literal.
Did you mean "=="?
  if sys.version_info.major is 3:
/opt/gdbpeda/lib/shellcode.py:379: SyntaxWarning: "is" with a literal.
Did you mean "=="?
  if pyversion is 3:
Reading symbols from my_date...
(No debugging symbols found in my_date)
gdb-peda$ run
Starting program: /home/seed/my_date
Wed Jan 28 18:19:44 UTC 2026
[Inferior 1 (process 41066) exited normally]
Warning: not running
gdb-peda$ quit
[01/28/26]seed@Kaiyuan:~$
```

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?

In GDB, the instruction pointer points into the executable code region, showing where the current code is executing. The stack pointer points into the stack region, which stores return addresses and saved data for function calls. The stack and code addresses are in a different region. And the stack addresses typically grow toward lower addresses as functions are called. This separation reflects that code and stack occupy different mapped regions with different permissions.

Now try the following.

```
gdb my_date
(gdb) set args "--version"
(gdb) break __libc_start_main
(gdb) run
(gdb) maintenance info sections
0x80521a0->0x805536f at 0x0000a1a0: .rodata ALLOC LOAD READONLY DATA
HAS_CONTENTS
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

---

It set a breakpoint at the beginning of the function to pause the program before main() run. The command "maintenance info sections" locates the .rodata section in memory. I used an extra command "searchmem "GNU" {start} {end} to find the start address of constant string "GNU coreutils". Next, I used the command "x/20s" to confirm I found the correct start address. Then overwrite the first four bytes of the string by using "set". Setting *{start} = 0x20554c42 changes the bytes to 42

4c 55 20, which is "BLU ". The final c continues the program and the output is the modified string"BLU coreutils" .

I also modified the name "David MacKenzie" to "Kaiyuan Xu" through the same method.

```
[gdb-peda$ searchmem "David" 0x80521a0 0x805536f
Searching for 'David' in range: 0x80521a0 - 0x805536f
Found 1 results, display max 1 items:
my_date : 0x8052290 ("David MacKenzie")
[gdb-peda$ set {char[15]} 0x8052290 = "Kaiyuan Xu"
[gdb-peda$ c
Continuing.
date (GNU coreutils) 8.25
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.net/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 Kaiyuan Xu.
[Inferior 1 (process 42408) exited normally]
Warning: not running
gdb-peda$
```

Snag: Before learning the command 'searchmem', I'm confused how to find the start address of the specific words. I wondered if I should to count the bites and modify the content after "GNU coreutils". By searching among the websites, I finally found this command which helped me much easier to locate the first address of the word.

COMPLETE

That's all! Please submit a .pdf version of this file to Gradescope.