CS 232 Project 4: Program Forensics

Ben DeWeerd
Sawyer Travis

Discoveries:

1. Tool: running the program without arguments: `./mystery`
    a. What we learned:
        i. Prints out “I just deleted all your files… [delay of ~2 seconds] not”, then deletes itself.
        ii. We should be more careful about running random programs from the command line.
2. Tool: `size <program_name>`
    a. Reference: https://www.howtoforge.com/linux-size-command/
    b. How it works:
        i. Typing the command [size mystery] shows the size of the program’s text, data, bss, dec, and hex.
    c. What we learned:
        i. We learned that the mystery file contains…
            1. 4906 bytes of text.
            2. 744 bytes of data.
            3. 48 bytes of BSS [unitialized data].
            4. 5698 bytes of dec.
            5. 1642 bytes of hex.
3. Tool: `ls -lh`
    a. How it works:
        i. Prints out all files, ‘L’ flag => long format, ‘H’ flag => human readable, prints units with sizes.
    b. What we learned:
        i. Total file size is 14kB.
4. Tool: `time <program_name>`
    a. How it works:
        i. The *time* command returns the runtime of each file within the executable
        ii. We ran time ./mystery [without any arguments]
    b. What we learned:
        i. Real: 2.006s
        ii. User: 0.002s
        iii. Sys: 0.000s
        iv. Total run time: 2.008s
5. Tool: `pidof <program_name>`
    a. Reference: https://www.cyberciti.biz/faq/howto-display-process-pid-under-linux-unix/

b. How it works:
    i. Run the command while the process is running to determine its process ID.
c. What we learned:
    i. In this instance, the process id was 12384.

6. Tool: stopping program execution.
    a. How it works:
        i. Run the program, then immediately use `<ctrl> + <c>` to stop execution.
    b. What we learned:
        i. The program does its dirty work, deleting itself, before the 2-second delay takes place. Using 'ls' in the program directory displays no output even when the program was terminated prematurely.
7. Tool: adding '-h' flag
    a. How it works:
        i. Adding the flag shows the program's command line arguments [`--help` does not].
    b. What we learned - command line arguments:
        i. `-h: show this help message`
        ii. `-n <i>: allocate <i> items`
        iii. `-p <port>: use port instead of default (10234) and send data out that port on TCP`
        iv. `-s: sort`
        v. `-e <seed>: use <seed> to seed the random number generator`
8. Tool: `strace ./<program_name>`
    a. Reference: https://linuxconfig.org/how-to-trace-system-calls-made-by-a-process-with-strace-on-linux
    b. How it works:
        i. Running the program with strace shows all of the system calls the program makes.
        ii. Running strace with the '-c' flag generates a summary of the number of system calls made and the time taken by each, in table form.
        iii. We ran '`strace -c ./mystery -n 6`'
    c. What we learned:
        i. The write() system calls use nearly all the time the program needs to run for small values of n - see summary table. For larger values of n, other calls take a slightly more significant amount of time.
        ii. To output 6 values, the program uses 39 system calls and has 2 errors - see summary table.

```
bendeweerd@BenZenbook:~/Documents/CS_232_Projects/Project_4$ strace -c ./mystery -n 6
1804289383
846930886
1681692777
1714636915
1957747793
424238335
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
100.00    0.000024           4         6           write
  0.00    0.000000           0         1           read
  0.00    0.000000           0         2           close
  0.00    0.000000           0         3           fstat
  0.00    0.000000           0         7           mmap
  0.00    0.000000           0         4           mprotect
  0.00    0.000000           0         1           munmap
  0.00    0.000000           0         3           brk
  0.00    0.000000           0         6           pread64
  0.00    0.000000           0         1         1 access
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         2         1 arch_prctl
  0.00    0.000000           0         2           openat
------ ----------- ----------- --------- --------- ----------------
100.00    0.000024                    39         2 total
```

9. Tool: `nm <program_name>`
    a. Reference: https://www.linuxtopia.org/online_books/an_introduction_to_gcc/gccintro_89.html
    b. How it works:
        i. Gives they symbol table of the program
        ii. We ran the command "`nm mystery`"
        iii. A list of variables and methods is printed

1.
```
sjt29@cs-ssh:~/cs232/mystery$ nm mystery
                 U accept@@GLIBC_2.2.5
                 U atoi@@GLIBC_2.2.5
                 U bind@@GLIBC_2.2.5
00000000006020f8 B __bss_start
                 U bzero@@GLIBC_2.2.5
                 U close@@GLIBC_2.2.5
0000000000400c76 T cmpfunc
0000000000602128 b completed.7585
00000000006020e8 D __data_start
00000000006020e8 W data_start
0000000000400bb0 t deregister_tm_clones
0000000000400c30 t __do_global_dtors_aux
0000000000601e18 t __do_global_dtors_aux_fini_array_entry
00000000006020f0 D __dso_handle
0000000000601e28 d _DYNAMIC
00000000006020f8 D _edata
0000000000602130 B _end
                 U exit@@GLIBC_2.2.5
                 U fflush@@GLIBC_2.2.5
0000000000401254 T _fini
0000000000400c50 t frame_dummy
0000000000601e10 t __frame_dummy_init_array_entry
0000000000401578 r __FRAME_END__
                 U fwrite@@GLIBC_2.2.5
0000000000602000 d _GLOBAL_OFFSET_TABLE_
                 w __gmon_start__
0000000000401420 r __GNU_EH_FRAME_HDR
                 U htons@@GLIBC_2.2.5
00000000004009a0 T _init
0000000000601e18 t __init_array_end
0000000000601e10 t __init_array_start
0000000000401260 R _IO_stdin_used
                 w _ITM_deregisterTMCloneTable
                 w _ITM_registerTMCloneTable
0000000000601e20 d __JCR_END__
0000000000601e20 d __JCR_LIST__
                 w _Jv_RegisterClasses
```

c. What we learned:
    i. The memory address for each symbol shown
    ii. Each letter tells the type of symbol
        1. For example, “U” means that that particular symbol is undefined
        2. Other letters like “B” tells the user what part of memory that symbol is stored in. “B” mean that the symbol is stored in the BSS data section.

What the program does:

The program acts in one of two ways: as a random number generator when run with flags, and as a mean program that deletes itself when run without flags. It has multiple command-line arguments (#7 above) which determine its behavior.

Bug:

Changing the seed doesn’t affect the program’s output - it produces the same results when using the ‘-n’ flag regardless of the seed.