Graham Dungan
April 5, 2025
UIN: 332001764

CSCE 413: Software Security
Class 32: Ltrace

Demonstration Quickstart

A demonstration script `demo.sh` has been included. This script will run both `1.bin` and `2.bin` with strace and ltrace, and will output their logs.
To run this demo,

1. Enter the `Class32` directory.
   `cd Class32`
2. *(Optional)* If the script does not have execution permission, add such permissions.
   `chmod +x ./demo.sh`
3. Run the `demo.sh` script.
   `./demo.sh`

What follows is a screenshot of the output of `demo.sh`.

```
❯ ./demo.sh
Running 1.bin with ltrace...
__libc_start_main(0x400526, 1, 0x7ffd4599fb58, 0x400550 <unfinished ...>
puts("malware"malware
)                                                                                 = 8
+++ exited (status 0) +++



------------------------------------------------
Running 1.bin with strace...
execve("./1.bin", ["./1.bin"], 0x7ffd8886f9f0 /* 31 vars */) = 0
brk(NULL)                               = 0x185d0000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffcdcb05ec0) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe174c93000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=49147, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 49147, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fe174c87000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\315A\vq\17\17\tLh2\355\331Y1\0m"..., 68, 896) = 68
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fe174a5e000
mprotect(0x7fe174a86000, 2023424, PROT_NONE) = 0
mmap(0x7fe174a86000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7fe174a86000
mmap(0x7fe174c1b000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7fe174c1b000
mmap(0x7fe174c74000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7fe174c74000
mmap(0x7fe174c7a000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fe174c7a000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe174a5b000
```

Folder Contents

The submission for this assignment also includes a folder `commented_logs/`. This folder contains the logs generated by the `demo.sh` script with brief comments explaining important behaviors of the `*.bin` files.

Running with Ltrace

`1.bin` with Ltrace

We can run `1.bin` with ltrace using the command;

```
ltrace -o 1_ltrace.log ./1.bin
```

```
❯ ltrace -o 1_ltrace.log ./1.bin
malware
❯ cat 1_ltrace.log
__libc_start_main(0x400526, 1, 0x7ffee657df08, 0x400550 <unfinished ...>
puts("malware")                                                          = 8
+++ exited (status 0) +++
```

```
1 __libc_start_main(0x400526, 1, 0x7ffee657df08, 0x400550 <unfinished ...>
2 puts("malware")                                                          =
      8
3 +++ exited (status 0) +++
```

Ltrace reveals that this program uses the libc library to initialize the main function, uses the "puts" library function to write "malware" to stdout, and exits with a status of 0.

`2.bin` with Ltrace

We can run `2.bin` with ltrace using the command;

```
ltrace -o 2_ltrace.log ./2.bin
```

```
❯ ltrace -o 2_ltrace.log ./2.bin
Couldn't find .dynsym or .dynstr in "/proc/7340/exe"
malware
❯ cat 2_ltrace.log
```

It is seen that nothing is output during the ltrace, except for "Couldn't find .dynsym or .dynstr in '/proc/7340/exe'" and prints malware as per usual. This is most likely to occur if the file is statically linked, as it has no dynamic symbol table. We will investigate this further in the following sections.

Running with Strace

`1.bin` with Strace

We can run `1.bin` with strace using the command;

```
strace -o 1_strace.log ./1.bin
```

```
❯ strace -o 1_strace.log ./1.bin
malware
❯ cat 1_strace.log
execve("./1.bin", ["./1.bin"], 0x7ffc4ec47460 /* 31 vars */) = 0
brk(NULL)                               = 0x1dd7a000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd9a5a3050) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f6d0d346000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=49147, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 49147, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f6d0d33a000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\315A\vq\17\17\tLh2\355\331Y1\0m"..., 68, 896) = 68
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0
```

```
1  execve("./1.bin", ["./1.bin"], 0x7ffc4ec47460 /* 31 vars */) = 0
2  brk(NULL)                                = 0x1dd7a000
3  arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd9a5a3050) = -1 EINVAL (Invalid argument)
4  mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f6d0d346000
5  access("/etc/ld.so.preload", R_OK)       = -1 ENOENT (No such file or directory)
6  openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
7  newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=49147, ...}, AT_EMPTY_PATH) = 0
8  mmap(NULL, 49147, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f6d0d33a000
9  close(3)                                 = 0
10 openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
11 read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) = 832
12 pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
13 pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48
14 pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\315A\vq\17\17\tLh2\355\331Y1\0m"..., 68, 896) = 68
15 newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0
16 pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
17 mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f6d0d111000
18 mprotect(0x7f6d0d139000, 2023424, PROT_NONE) = 0
19 mmap(0x7f6d0d139000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000)
       = 0x7f6d0d139000
20 mmap(0x7f6d0d2ce000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0
       x7f6d0d2ce000
21 mmap(0x7f6d0d327000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000)
       = 0x7f6d0d327000
22 mmap(0x7f6d0d32d000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0
       x7f6d0d32d000
23 close(3)                                 = 0
24 mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f6d0d10e000
25 arch_prctl(ARCH_SET_FS, 0x7f6d0d10e740) = 0
26 set_tid_address(0x7f6d0d10ea10)          = 2057
27 set_robust_list(0x7f6d0d10ea20, 24)      = 0
28 rseq(0x7f6d0d10f0e0, 0x20, 0, 0x53053053) = 0
29 mprotect(0x7f6d0d327000, 16384, PROT_READ) = 0
30 mprotect(0x600000, 4096, PROT_READ)      = 0
31 mprotect(0x7f6d0d380000, 8192, PROT_READ) = 0
32 prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
33 munmap(0x7f6d0d33a000, 49147)             = 0
34 newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}, AT_EMPTY_PATH) = 0
35 getrandom("\xf3\x5f\x15\x64\xf3\xae\x80\xc4", 8, GRND_NONBLOCK) = 8
36 brk(NULL)                                = 0x1dd7a000
37 brk(0x1dd9b000)                          = 0x1dd9b000
38 write(1, "malware\n", 8)                 = 8
39 exit_group(0)                            = ?
40 +++ exited with 0 +++
```

As seen here, line 1 runs the program with execve. Line 5 attempts to load any preloaded shared libraries. Subsequent lines, until line 35, concern mapping memory addresses, memory protection, and establishing a thread/process. Line 35 obtains a random number, most likely for stack protection. Line 38 writes "malware" to stdout, and the program subsequently exits at line 40. This follows the expected behavior as seen in ltrace, as the program initializes, writes "malware" to console, and exits.

`2.bin` with Strace

We can run `2.bin` with strace using the command;

`strace -o 2_strace.log ./2.bin`

```
❯ strace -o 2_strace.log ./2.bin
malware
❯ cat 2_strace.log
execve("./2.bin", ["./2.bin"], 0x7fff517a4a40 /* 31 vars */) = 0
uname({sysname="Linux", nodename="DESKTOP-BP9AMH5", ...}) = 0
brk(NULL)                               = 0xc2e3000
brk(0xc2e41c0)                          = 0xc2e41c0
arch_prctl(ARCH_SET_FS, 0xc2e3880)      = 0
readlink("/proc/self/exe", "/home/user/csce_413/class_32/2_b"..., 4096) = 40
brk(0xc3051c0)                          = 0xc3051c0
brk(0xc306000)                          = 0xc306000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
write(1, "malware\n", 8)                = 8
exit_group(0)                           = ?
+++ exited with 0 +++
```

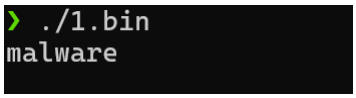
```
1  execve("./2.bin", ["./2.bin"], 0x7fff517a4a40 /* 31 vars */) = 0
2  uname({sysname="Linux", nodename="DESKTOP-BP9AMH5", ...}) = 0
3  brk(NULL)                               = 0xc2e3000
4  brk(0xc2e41c0)                          = 0xc2e41c0
5  arch_prctl(ARCH_SET_FS, 0xc2e3880)      = 0
6  readlink("/proc/self/exe", "/home/user/csce_413/class_32/2_b"..., 4096) = 40
7  brk(0xc3051c0)                          = 0xc3051c0
8  brk(0xc306000)                          = 0xc306000
9  access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
10 fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
11 write(1, "malware\n", 8)                = 8
12 exit_group(0)                           = ?
13 +++ exited with 0 +++
```

It is seen that `2.bin` has fewer system calls than `1.bin`. As seen above, line 1 runs the program. Line 2 attempts to read system information such as the system name/type and usernames. Line 6 finds the absolute path of where the binary is. Line 10 calls fstat to ensure that it is a terminal and follows on line 11 with a write call to write "malware". The program then terminates. Given that we expect this program to be statically linked, it would make sense that there are much fewer system calls when running the program.

Running without Ltrace/Strace

`1.bin` without Ltrace/Strace

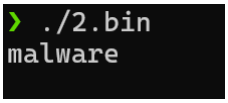
We can run `1.bin` normally,



It is seen that the program simply prints "malware" and exits.

`2.bin` without Ltrace/Strace

We can run `2.bin` normally,



It is seen that the program similarly prints "malware" and exits.

Behavior Analysis

It is seen that both programs behave somewhat similarly- when run, they print "malware". The difference between these files, however, is in how they were compiled. Ltrace did not work on `2.bin` because it is statically linked, which we can check with `file 2.bin`,

It is seen here that `2.bin` is statically linked and, thus, has no dynamically linked functions to call. It follows that, since it was statically linked, it does not make as many calls to external libraries, such as libc, and thus has a smaller number of system calls seen in strace. We can conclude that both programs print "malware" to stdout when run, where `1.bin` is dynamically linked and `2.bin` is statically linked, making it so that one cannot run ltrace on the former.