Graham Dungan
April 5, 2025
UIN: 332001764

CSCE 413: Software Security
Class 31: Strace

Demonstration Quickstart

A demonstration script `exploit.sh` has been included. This script will run both `1.bin` and `2.bin` with strace. For both of these programs, it will first run strace normally and then with parameters that expose how they actually work.
To run this demo,

1. Enter the `Class31` directory.
   `cd Class31`
2. *(Optional)* If the script does not have execution permission, add such permissions.
   `chmod +x ./exploit.sh`
3. Run the `exploit.sh` script.
   `./exploit.sh`

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

```
user@DESKTOP-HJPALP7:~/csce_413/class_31$ ls
1.bin  2.bin  commented_logs  exploit.sh  fake_ptrace.c  fake_ptrace.so  strace
user@DESKTOP-HJPALP7:~/csce_413/class_31$ ./exploit.sh
1.bin: Running ./1.bin normally.
World
Hello

1.bin: Running ./1.bin with follow forks.
Hello
World

2.bin: Running ./2.bin normally.

2.bin: Running ./2.bin with a LD_PRELOAD and attaching.
       Started ./2.bin with PID: 717
strace: Process 717 attached
strace: Process 721 attached
I'm a malware
user@DESKTOP-HJPALP7:~/csce_413/class_31$ ls
1.bin         1_strace_followforks.log  2_strace.log                commented_logs  fake_ptrace.c   strace
1_strace.log  2.bin                     2_strace_preload_attach.log  exploit.sh      fake_ptrace.so
user@DESKTOP-HJPALP7:~/csce_413/class_31$
```

Folder Contents

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

Running with Strace

`1.bin` with Strace

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

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

This will run strace and output everything to `1_strace.log`.

```
user@DESKTOP-HJPALP7:~/csce_413/class_31$ strace -o 1_strace.log ./1.bin
World
Hello
user@DESKTOP-HJPALP7:~/csce_413/class_31$ cat 1_strace.log
execve("./1.bin", ["./1.bin"], 0x7ffedd99b430 /* 28 vars */) = 0
brk(NULL)                               = 0x5ca5000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe5b2bff000
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
fstat(3, {st_mode=S_IFREG|0644, st_size=23543, ...}) = 0
mmap(NULL, 23543, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fe5b2bf9000
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\0\220\243\2\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\0\0"..., 784, 64) = 784
fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 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\0\0"..., 784, 64) = 784
mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fe5b29e7000
mmap(0x7fe5b2a0f000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7fe5b2a
0f000
mmap(0x7fe5b2b97000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7fe5b2b97000
mmap(0x7fe5b2be6000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7fe5b2b
e6000
mmap(0x7fe5b2bec000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fe5b2bec000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe5b29e4000
arch_prctl(ARCH_SET_FS, 0x7fe5b29e4740) = 0
set_tid_address(0x7fe5b29e4a10)         = 775
```

It is seen that, while running, `1.bin` has output "Hello World" on two separate lines. Inspecting the log, lines 30-38 hold the most significance:

```
1 clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0
      x7fe5b29e4a10) = 776
2 fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
3 --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=776, si_uid=1000, si_status=0, si_utime
      =0, si_stime=0} ---
4 getrandom("\xe7\xe3\x64\xe6\xbd\xa2\xe6\xca", 8, GRND_NONBLOCK) = 8
5 brk(NULL)                               = 0x5ca5000
6 brk(0x5cc6000)                          = 0x5cc6000
7 write(1, "Hello\n", 6)                  = 6
8 exit_group(0)                           = ?
9 +++ exited with 0 +++
```

Line 30 (represented as line 1 here) creates a child process with PID 776. The remainder of the program (the parent process) prints "Hello" and then exits. Since there is nothing to indicate that "World" is being printed in this program, the child process must be printing this. We will investigate this further in the Analysis section.

`2.bin` with Strace

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

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

This will run strace and output everything to `2_strace.log`.

```
user@DESKTOP-HJPALP7:~/csce_413/class_31$ strace -o 2_strace.log ./2.bin
user@DESKTOP-HJPALP7:~/csce_413/class_31$ cat 2_strace.log
execve("./2.bin", ["./2.bin"], 0x7fff86735430 /* 28 vars */) = 0
brk(NULL)                               = 0x55a37c74e000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8466073000
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
fstat(3, {st_mode=S_IFREG|0644, st_size=23543, ...}) = 0
mmap(NULL, 23543, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f846606d000
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\0\220\243\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
fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 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, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f8465e5b000
mmap(0x7f8465e83000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f8465e
83000
mmap(0x7f846600b000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7f846600b000
mmap(0x7f846605a000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7f84660
5a000
mmap(0x7f8466060000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f8466060000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f8465e58000
arch_prctl(ARCH_SET_FS, 0x7f8465e58740) = 0
set_tid_address(0x7f8465e58a10)         = 783
set_robust_list(0x7f8465e58a20, 24)     = 0
rseq(0x7f8465e59060, 0x20, 0, 0x53053053) = 0
```

It is seen that the output prints nothing when being run with strace. We will now inspect `2_strace.log`, lines 30-32,

```
1 ptrace(PTRACE_TRACEME)                  = -1 EPERM (Operation not permitted)
2 exit_group(0)                           = ?
3 +++ exited with 0 +++
```

It is seen here that the program is making a call to ptrace, a system call made to allow a parent process to trace the child. In this instance, the program is checking if it's parent can trace it, which it fails. Barring irregular behavior, this system call returns a -1 (fails) typically if there exists a program already tracing it. Since we are using strace, it follows that the program is already being traced. This could be an anti-analysis line, which will be further examined in the Analysis section.

Running without Strace

`1.bin` without Strace

We will run `1.bin` normally without the use of strace,

```
user@DESKTOP-HJPALP7:~/csce_413/class_31$ ./1.bin
Hello
World
```

As seen here, the program outputs the same as before. We will attempt to find where the call made to print "World" is coming from in the Analysis section.

`2.bin` without Strace

We will run `2.bin` normally without the use of strace,

```
user@DESKTOP-HJPALP7:~/csce_413/class_31$ ./2.bin
I'm a malware

[1]+  Stopped                 ./2.bin
```

It is seen that the program printed "I'm a malware" and then stopped, which is different than the output seen in strace. This is most likely the purposeful stopping of the program if it is/is not being traced. This will be investigated further in the Analysis section.

Behavior Analysis

Analysis of `1.bin`

It is seen that `1.bin` created a child process while running, and that the parent process did not have a print function for "World". It is most likely that the program is creating a child process that prints "World". We can use strace `-f` follow-forks flag to observe the output of its children,

```
strace -f -o 1_strace_followforks.log ./1.bin
```

This will create a log file `1_strace_followforks.log`,

```
user@DESKTOP-HJPALP7:~/csce_413/class_31$ strace -f -o 1_strace_followforks.log ./1.bin
Hello
World
user@DESKTOP-HJPALP7:~/csce_413/class_31$ cat 1_strace_followforks.log
4813  execve("./1.bin", ["./1.bin"], 0x7ffc8cde32e8 /* 28 vars */) = 0
4813  brk(NULL)                         = 0x28906000
4813  mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe4933f2000
4813  access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
4813  openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
4813  fstat(3, {st_mode=S_IFREG|0644, st_size=23543, ...}) = 0
4813  mmap(NULL, 23543, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fe4933ec000
4813  close(3)                          = 0
4813  openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
4813  read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0"..., 832) = 832
4813  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
4813  fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0
4813  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
4813  mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fe4931da000
4813  mmap(0x7fe493202000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7
fe493202000
4813  mmap(0x7fe49338a000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7fe49338a00
0
```

We can analyze the contents of this log file, specifically lines 30-56,

```
1  4813  clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0
       x7fe4931d7a10) = 4814
2  4814  set_robust_list(0x7fe4931d7a20, 24 <unfinished ...>
3  4813  fstat(1,  <unfinished ...>
4  4814  <... set_robust_list resumed>)    = 0
5  4813  <... fstat resumed>{st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
6  4813  getrandom( <unfinished ...>
7  4814  fstat(1,  <unfinished ...>
8  4813  <... getrandom resumed>"\x78\x55\xa1\x76\xdd\xed\xe2\xfd", 8, GRND_NONBLOCK) = 8
9  4814  <... fstat resumed>{st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
10 4813  brk(NULL <unfinished ...>
11 4814  getrandom( <unfinished ...>
12 4813  <... brk resumed>)                = 0x28906000
13 4814  <... getrandom resumed>"\xff\x63\x34\xf4\xc2\x29\xb3\xeb", 8, GRND_NONBLOCK) = 8
14 4813  brk(0x28927000)                   = 0x28927000
15 4814  brk(NULL <unfinished ...>
16 4813  write(1, "Hello\n", 6 <unfinished ...>
17 4814  <... brk resumed>)                = 0x28906000
```

```
4813  <... write resumed>)              = 6
4814  brk(0x28927000 <unfinished ...>
4813  exit_group(0 <unfinished ...>
4814  <... brk resumed>)                = 0x28927000
4813  <... exit_group resumed>)         = ?
4814  write(1, "World\n", 6 <unfinished ...>
4813  +++ exited with 0 +++
4814  <... write resumed>)              = 6
4814  exit_group(0)                     = ?
4814  +++ exited with 0 +++
```

It is seen that, on line 30 (line 1 in this instance), the parent process `4813` creates a child process `4814`. Line 16 shows the parent process printing "Hello" and the child process writing "World" on line 23. Both parent and child processes terminate afterwards. It is seen that we have found the true behavior of `1.bin`- the process creates a child, the parent prints "Hello", the child prints "World", and then both terminate.

Analysis of `2.bin`

We have seen that `2.bin` terminated when traced with strace and printed "I'm a malware" when ran normally. This behavior is caused by the system call made to `ptrace`. It follows that if ptrace did not detect that strace was being used to trace the program, we could continue analyzing its behavior. For this, I created a shared object file `fake_ptrace.c` that serves to act as a preloaded `ptrace` replacement.

```c
#define _GNU_SOURCE
#include <stdio.h>
#include <dlfcn.h>
#include <sys/ptrace.h>
#include <errno.h>
#include <sys/types.h>
#include <stdarg.h>
#include <unistd.h>
const int WAIT_TIME = 1;
long (*orig_ptrace)(enum __ptrace_request request, ...) = NULL;
long ptrace(enum __ptrace_request request, ...) {
    if (!orig_ptrace) {
        orig_ptrace = dlsym(RTLD_NEXT, "ptrace");
    }
    va_list args;
    va_start(args, request);
    if (request == PTRACE_TRACEME) {
        sleep(WAIT_TIME);
        va_end(args);
        return 0;
    }
    pid_t pid = va_arg(args, pid_t);
    void* addr = va_arg(args, void*);
    void* data = va_arg(args, void*);
    va_end(args);
    return orig_ptrace(request, pid, addr, data);
}
```

If ptrace is called with `PTRACE_TRACEME` (which `2.bin` does) it will falsify the output, always returning true.

After some experimentation, I also added a `sleep` call. This will give us some time to attach strace to the running program instead of running the program with strace. This will help further avoid anti-tampering features. This can be compiled with,

```
gcc -shared -fPIC -o fake_ptrace.so fake_ptrace.c -ldl
```

Now that we have created `fake_ptrace.so` such that it will return that the program is never being traced and will initialize a sleep long enough to attach strace to the it, we can run the following command;

```
LD_PRELOAD=./fake_ptrace.so ./2.bin & pid=$! && strace -f -p $pid -o 2_strace_preload_attach.log
```

This command will run `2.bin` with the fake `ptrace` preloaded. It will then capture the PID of this program and attach strace,

```
user@DESKTOP-HJPALP7:~/csce_413/class_31$ LD_PRELOAD=./fake_ptrace.so ./2.bin & pid=$! && strace -f -p $pid -o
2_strace_preload_attach.log
[2] 7450
strace: Process 7450 attached
strace: Process 7460 attached
I'm a malware
[2]-  Done                    LD_PRELOAD=./fake_ptrace.so ./2.bin
user@DESKTOP-HJPALP7:~/csce_413/class_31$ cat 2_strace_preload_attach.log
7450  restart_syscall(<... resuming interrupted read ...>) = 0
7450  clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD <unfinished ...>
7460  set_robust_list(0x7f321bbf1a20, 24 <unfinished ...>
7450  <... clone resumed>, child_tidptr=0x7f321bbf1a10) = 7460
7460  <... set_robust_list resumed>)    = 0
7450  wait4(-1,  <unfinished ...>
7460  fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
7460  getrandom("\x68\xd8\x3a\xd0\x68\x71\xd8\xf4", 8, GRND_NONBLOCK) = 8
7460  brk(NULL)                         = 0x5630313e6000
7460  brk(0x563031407000)               = 0x563031407000
7460  write(1, "I'm a malware\n", 14)   = 14
7460  exit_group(0)                     = ?
7460  +++ exited with 0 +++
7450  <... wait4 resumed>[{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 7460
7450  --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=7460, si_uid=1000, si_status=0, si_utime=0, si_
stime=0} ---
7450  exit_group(0)                     = ?
7450  +++ exited with 0 +++
```

As seen here, the program output "I'm a malware" and strace successfully produced a log of its system calls.

```
1  7450  restart_syscall(<... resuming interrupted read ...>) = 0
2  7450  clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD <unfinished
       ...>
3  7460  set_robust_list(0x7f321bbf1a20, 24 <unfinished ...>
4  7450  <... clone resumed>, child_tidptr=0x7f321bbf1a10) = 7460
5  7460  <... set_robust_list resumed>)    = 0
6  7450  wait4(-1,  <unfinished ...>
7  7460  fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0), ...}) = 0
8  7460  getrandom("\x68\xd8\x3a\xd0\x68\x71\xd8\xf4", 8, GRND_NONBLOCK) = 8
9  7460  brk(NULL)                         = 0x5630313e6000
10 7460  brk(0x563031407000)               = 0x563031407000
11 7460  write(1, "I'm a malware\n", 14)   = 14
12 7460  exit_group(0)                     = ?
13 7460  +++ exited with 0 +++
14 7450  <... wait4 resumed>[{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 7460
15 7450  --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=7460, si_uid=1000, si_status=0,
       si_utime=0, si_stime=0} ---
16 7450  exit_group(0)                     = ?
17 7450  +++ exited with 0 +++
```

The program created a child on line 2 with PID `7460`, as seen on line 4. The child then prints "I'm a malware" on line 11 and terminates alongside the parent on lines 13 and 17. The true behavior of this program is that the parent process runs `ptrace` to check if it is being traced. If not, it creates a child process and prints "I'm a malware".