CS342 Computer Security **Handout # 10**
Prof. Lyn Turbak **October 7, 2014**
Wellesley College

Lab 6: strace

Overview

The purpose of today's lab is to become familiar with the Linux `strace` command, a tool which allows you to examine the system calls made when executing other Linux commands.

Online documentation of Linux system calls can be found here:
http://man7.org/linux/man-pages/man2/syscalls.2.html.

This will be our first lab in which we use EDURange, an online computer security playground being built as part of an NSF-funded project led by Richard Weiss (The Evergreen State College), Jens Mache (Lewis and Clark College) and Michael Locasto (University of Calgary). Special thanks go to Kahea Hendrickson, a student at The Evergreen State College, for his work on the EDURange scenario system we will be using today.

How to Start

You will work in teams of 2 or 3 students on these exercises.

Step 1: Create an Answer Doc. For writing down your answers, each team should create one Google Doc for writing down answers to this lab. All team members should be listed at the top of the lab. Set the sharing option of this doc to *Anyone with a Link*. Each team member should immediately include a link to this document in that team member's CS342 portfolio. One team member should immediately email the link to Lyn. These links will be share with other members of the EDURange team.

Step 2: Log into the strace scenario. You should have received an email with a link to your EDURange student account. Connect to this account. It should indicate that one of your running scenarios is the strace scenario. This scenario should list a login name of the form `student_`*`XX`*, a password, and a *NAT_IP* address In a terminal window, each student should use `ssh` to connect to `student_`*`XX`*`@`*`NAT_IP`*.

Once connected, team members should work together to answer the following questions on their shared answer docs. Note: the EDURange scenario page has a *Questions* link that you should ignore for this scenario.

Q1: Your home directory in the account on the EDURange strace scenario NAT instance contains various files that will be used in this scenario. One is the file `empty.c`, whose contents is:

```
int main () {}
```

Compile this program as follows:

```
gcc -o empty empty.c
```

Now run `strace` to execute the `empty` program:

```
strace ./empty
```

What do you think the output of strace indicates in this case? How many different system call functions do you see?

Q2: The `-o` option of strace writes its output to a file. Do the following:

```
strace -o empty1 ./empty
strace -o empty2 ./empty
diff empty1 empty2
```

Explain the differences reported between traces `empty1` and `empty2`.

Q3: Study the following program `copy.c`.

```
# include <stdio.h>
# include <stdlib.h>

int main (int argc, char** argv) {
  char c;
  FILE* inFile;
  FILE* outFile;
  char outFileName[256];
  if (argc != 3) {
    printf("program usage: ./copy <infile> <outfile>\n");
    exit(1);
  }
  snprintf(outFileName, sizeof(outFileName), "%s/%s",
           getenv("HOME"), argv[2]);
  inFile = fopen(argv[1], "r");
  outFile = fopen(outFileName, "w");
  printf("Copying %s to %s\n", argv[1], outFileName);
  while ((c = fgetc(inFile)) != EOF) {
       fprintf(outFile, "%c", c);
  }
  fclose(inFile);
  fclose(outFile);
}
```

Compile it to an executable named `copy`, and use `strace` to execute it as follows:

```
strace ./copy tiger.txt mytiger.txt
```

Explain the non-boilerplate parts of the trace by associating them with specific lines in `copy.c`.

Q4: The file `strace-identify` was created by calling `strace` on a command. The first line of the trace has been deleted to make it harder to identify. Determine the command on which `strace` was called to produce this trace.

Q5: The file `mystery` is an executable whose source code is not available. Use `strace` to explain what the program does in the context of the following examples:

```
mystery foo abc
mystery foo def
mystery baz ghi
```

Q6. *Note: please do Q7 and Q8 before this problem; it turns out they're helpful for solving this problem.* Create a one-line "secret" file. Here's an example, though of course you choose something different as your secret:

```
echo "My phone number is 123-456-7890" > secret
```

Now display the secret to yourself using `cat`:

```
cat secret
```

Is your file *really* secret? How much do you trust the `cat` program? Run strace on `cat secret` to determine what it's actually doing Can other students read your secret? Can you read the secrets of other students? Explain everything you observe.

Q7. Here is a simple shell script in `script.sh`:

```
#!/bin/bash
echo "a" > foo.txt
echo "bc" >> foo.txt
echo `id -urn` >> foo.txt
chmod 750 foo.txt
/bin/cat foo.txt | wc
```

Compare the outputs of the following calls to `strace` involving this script. Explain what you see in the traces in terms of the commands in the script.

```
./script/sh
strace ./script.sh
strace -f ./script.sh
```

Q8. Sometimes strace prints out an overwhelming amount of output. One way to filter through the output is to save the trace to a file and search through the file with `grep`. But strace is equipped with some options that can do some summarization and filtering. To see some of these, try the following, and explain the results:

```
find /etc/pki
strace find /etc/pki
strace -c find /etc/pki
strace -e trace=file find /etc/pki
strace -e trace=open,close,read,write find /etc/pki
```