The strace utility

1 Overview

This exercise introduces use of the Unix strace utility to record system calls make by a running program. The lab includes the following objectives:

- How to run strace.
- Understanding how the output of strace corresponds to system calls.
- Potential use of strace for dynamic analysis of software.

1.1 Background

This lab assumes some familiarity with the Unix command line and some introduction to system calls.

The strace utility trace system calls, so you will want to become familiar with the kinds of system calls that might be made by a service that handles UDP traffic. You can learn about UDP traffic by using man, e.g.,

```
man udp
```

Or try your favorite web sources. Learn about strace using:

```
man strace
```

2 Lab Environment

This lab runs in the Labtainer framework, available at http://nps.edu/web/c3o/labtainers. That site includes links to a pre-built virtual machine that has Labtainers installed, however Labtainers can be run on any Linux host that supports Docker containers.

From your labtainer-student directory start the lab using:

```
labtainer strace
```

A link to this lab manual will be displayed.

3 Environment

The lab includes two computers, one called *the-client* and another called *the-server*. You will only get a terminal on the-client.

The IP address of the-server is `10.10.0.2`, which can be reached from the-client. A service is running on the-server called *observer* and that service has been known to provide some kind of network communication using UDP.

You have a copy of the observer executable on the-client.

4 Tasks

Your objective is to use strace to dynamically analyze the observer program and determine the port number that it listens on.

4.1 Explore

- Use the ping command to confirm the-client can reach the-server (10.10.0.2).
- Use the Unix `file` command to learn a bit about the observer program.
- Try running the observer program.

4.2 Use strace

Use the strace utility to observe Linux system calls made by the observer program as it executes:

```
strace ./observer
```

Pay attention to the progress made by the program and to any issues that may cause it to fail. Don't be afraid to create files on the client if that helps you to get further with the program. And keep in mind that your goal is to find the port number being listened to. You need not get the program to run successfully.

4.3 Test the port with sendudp.py

Once you think you've found the port number, test it by using the sendudp.py script on the-client:

```
./sendudp.py <port>
```

4.4 Extra

Consider situations where they type of dynamic analysis provided by strace might be useful due to limited alternative approaches. For example, tools such a nmap can have difficulty identifying open UDP ports. Try it from the-client if you'd like, assuming you are familiar with nmap (see the nmap-discover and nmap-ssh labs). Even when you know the port number to scan, nmap may not be reliable. Try that:

```
sudo nmap -sU -p <port> 10.10.0.2
```

Static analysis tools such as Ghidra are another alternative for reverse engineering program functions. However, some programs may obfuscate their functions, complicating the process of determining things like port numbers. Your the-client computer has Ghidra installed, which you can start using:

```
./ghidra
```

If you are not familiar with Ghidra, consider trying the ghidra lab.

5 Submission

After finishing the lab, go to the terminal on your Linux system that was used to start the lab and type:

```
stoplab
```

When you stop the lab, the system will display a path to the zipped lab results on your Linux system. Provide that file to your instructor, e.g., via the Sakai site.

This lab was developed for the Labtainer framework by the Naval Postgraduate School, Center for Cybersecurity and Cyber Operations under sponsorship from the National Science Foundation Award Number 1932950. This work is in the public domain, and cannot be copyrighted.