Assignment 3

Buffer Overflow Attacks

Basic Setup

Exploiting buffer overflows requires **precise** control over the execution environment. A small change in the compiler, environment variables, or the way the program is executed can result in slightly different **memory layout** and **code structure**, thus requiring a different exploit. For this reason, this assignment uses a Virtual Machine (VM) to run the **vulnerable** web server code.

To start working on this assignment, you'll need **Software (Hypervisor)** that lets you run a virtual machine. For Linux users, we recommend running the **VM** over Kernel-based Virtual Machine (KVM), which is built into the Linux kernel. KVM should be available through your Ubuntu distribution, try `sudo apt-get install qemu-kvm`. KVM requires **hardware virtualization** support in your CPU, and you must enable this support in your BIOS (which is often, but not always, the default). If you have another virtual machine monitor installed on your machine (e.g., VMware), that virtual machine monitor may grab the hardware virtualization support exclusively and **prevent** KVM from working.

On Linux without KVM, use VMware Workstation from IS&T or, Oracle VirtualBox.

If you are using a computer with a **non-x86** CPU (such as a Mac with an **ARM M1 or M2** processor), running the VM locally on your computer may be **prohibitively slow**, because your VM will have to **emulate** x86 instructions instead of running them natively. Hence, it is strictly recommended for everyone to rent a Ubuntu machine from Baadal VM for this assignment, or perform a **local** setup on a x86 machine (preferrably **Ubuntu 22.04** or higher).

Setup IITD Proxy to make all private IP addresses accessible to you. Once you have hypervisor installed on your machine, you should download the VM image using the following commands.

```
wget --no-check-certificate  http://10.237.27.193/2402-SIL765.sh
wget --no-check-certificate  http://10.237.27.193/2402-SIL765.vmdk
```

This virtual machine contains an installation of **Ubuntu 22.04** Linux.

To start the VM using **VMware**, import `2402-SIL765.vmdk`. Go to File > New select "create a custom virtual machine", choose Linux > Debian 9.x 64-bit, choose Legacy BIOS, and use an existing virtual disk (and select the `2402-SIL765.vmdk` file, choosing the "Take this disk away" option). Finally, click Finish to complete the setup.

To start the VM with **KVM**, first make the shell script executable using the command `sudo chmod a+x 2402-SIL765.sh` and then run `sudo ./2402-SIL765.sh` from a terminal (`Ctrl+A x` to force quit). To prevent writing **sudo** while executing the script, try adding yourself to the **kvm group**.

You will use the **student** account in the VM for your work. The password for the student account is **student**. You can also get access to the **root** account in the VM using **sudo**; for example, you can install new software packages using `sudo apt-get install <pkgname>`.

You can either log into the virtual machine using its **console**, or use Secure Shell (SSH) to log into the virtual machine over the (virtual) network. The latter also lets you easily copy files into and out of the virtual machine with Secure Copy Protocol (SCP) or Remote Sync (rsync). How you access the virtual machine over the network depends on how you're running it. If you're using **VMWare**, you'll first have to find the virtual machine's **IP address**. To do so, log in on the console, run `ip addr show dev eth0`, and note the IP address listed beside **inet**.

With **KVM**, you can use **localhost** as the IP address for **SSH** and **HTTP**. You can now log in with ssh by running the following command from your host machine: `ssh -p 2222 student@<IPADDRESS>`.

For security reasons, SSH **does not** allow logging in over the network using a password (and, in this specific case, the password is known to everyone). To log in via SSH, you will need to set up an SSH Key.

You may also find it helpful to create a host alias for your VM in your **~/.ssh/config file**, so that you can simply run, for example, `ssh lab` or `scp -r ./lab/ lab:~`. To do this, add the following lines to your **~/.ssh/config file**:

```
Host lab
Hostname localhost
User student
Port 2222
```

Getting started

The files you will need for this lab are distributed using the Git version control system. You can also use Git to keep track of any changes you make to the initial source code. Here's an overview of Git and the Git user's manual, which you may find useful.

The course Git repository is available here. To get the lab code, log into the VM using the student account (by running `./2402-SIL765.sh` in a terminal and running the following commands in separate terminal). Setup IITD Proxy and GitHub Proxy in your remote machine and clone the source code for this assignment as follows:

```
student@65660-v23:~$ git clone https://github.com/prateekbhaisora/SIL765-
    Assignment-3.git lab
Cloning into 'lab'...
student@65660-v23:~$ cd lab
student@65660-v23:~/lab$
```

It's important that you clone the repository into the lab directory, because the length of pathnames will matter in this assignment. Alternatively, you can clone the repo in the **host** machine and scp the file/folder contents to **remote** machine, as mentioned above.

Before you proceed with this assignment, remove the old binaries (if any) and make sure you can compile the **zookws** web server:

```
student@65660-v23:~/lab$ sudo rm zookd zookd-exstack zookd-nxstack
student@65660-v23:~/lab$ make
cc zookd.c -c -o zookd.o -m64 -g -std=c99 -Wall -Wno-format-overflow -
    D_GNU_SOURCE -static -fno-stack-protector
cc http.c -c -o http.o -m64 -g -std=c99 -Wall -Wno-format-overflow -
    D_GNU_SOURCE -static -fno-stack-protector
cc -m64  zookd.o http.o   -o zookd
cc -m64 zookd.o http.o   -o zookd-exstack -z execstack
cc -m64 zookd.o http.o   -o zookd-nxstack
cc zookd.c -c -o zookd-withssp.o -m64 -g -std=c99 -Wall -Wno-format-
    overflow -D_GNU_SOURCE -static
cc http.c -c -o http-withssp.o -m64 -g -std=c99 -Wall -Wno-format-
    overflow -D_GNU_SOURCE -static
cc -m64  zookd-withssp.o http-withssp.o   -o zookd-withssp
cc -m64   -c -o shellcode.o shellcode.S
objcopy -S -O binary -j .text shellcode.o shellcode.bin
cc run-shellcode.c -c -o run-shellcode.o -m64 -g -std=c99 -Wall -Wno-
    format-overflow -D_GNU_SOURCE -static -fno-stack-protector
cc -m64  run-shellcode.o   -o run-shellcode
student@65660-v23:~/lab$
```

The component of **zookws** that receives HTTP requests is **zookd**. It is written in **C** and serves static files and executes dynamic scripts. For this assignment, you don't have to understand the dynamic scripts; they are written in Python and the exploits in this assignment apply only to C code. The HTTP-related code is in **http.c**. Here is a tutorial about the HTTP protocol.

There are two versions of zookd you will be using:

- zookd-exstack
- zookd-nxstack

zookd-exstack has an **executable** stack, which makes it easy to inject executable code given a **stack buffer overflow vulnerability**. **zookd-nxstack** has a **non-executable stack**, and requires a more sophisticated attack to exploit stack buffer overflows. In order to run the web server in a **predictable** fashion — so that its stack and memory layout is the **same** every time — you will use the clean-env.sh script. This is the same way in which we will run the web server during grading, so make sure all of your exploits work on this configuration!

The reference binaries of **zookd** are provided in `bin.tar.gz`, which we will use for grading. Make sure your exploits work on those binaries. The `sudo sudo make check-lab1` command will always use both clean-env.sh and `bin.tar.gz` to check your submission.
Now, make sure you can run the **zookws** web server and access the **zoobar** web application from a browser running on your machine, as follows:

```
student@65660-v23:~/lab$ ./clean-env.sh ./zookd 8080
```

The clean-env.sh commands starts **zookd** on port **8080**. To open the zoobar application, open your browser and point it at the URL `http://<IPADDRESS>:8080/`, where <IPADDRESS> is the VM's IP address we found above.