

# Cyber Security Lab 3

## Due

Feb 28 at class.

## Goal

Explore worm building techniques.

## Things you need to know

You will be given code for a trivial Linux server and client (vulserver and vulclient on the class web site). Vulserver listens on TCP/10001 and responds to connections with a very simplistic "magic eightball" response. Vulserver also has a very obvious buffer overflow error.

You must make sure that the linux machines have stack randomization turned off. The ones you need to change on a FC system are:

```
/proc/sys/kernel/exec-shield
```

```
/proc/sys/kernel/exec-shield-randomize
```

They both default to 1, but you won't get any buffer overflows to work unless you set them to 0. As with the IP-forwarding, you can add lines to /etc/sysctl.conf like:

```
kernel.exec-shield = 0
```

```
kernel.exec-shield-randomize = 0
```

You can use the shellcode used in `intestsc2.c` of the phrack writeup to exercise the overflow by creating a root shell on the server machine assuming the server program is running as root (either directly or by setting the program `setuid root`). The phrack shell code is already typed into the `eggshell` and `testsc` programs that come with this assignment.

You are provided an `eggshell` program to help you figure out the offset to create the appropriate return value in your attack packet. The `eggshell` program returns the attack packet in the `$EGG` environment variable in the returned shell. You can use that value directly in the `vulclient` program to try the packet, e.g., `vulclient 192.168.1.107 "$EGG"`.

Once you get a root shell to appear on the server machine, you will want to tie the `stdin`, `stdout`, `stderr` of the shell to the client's TCP socket. The `shelldupasm.c` code augments the basic shell attack with the `dup` logic. It guesses that the socket was the last generated file descriptor. It `dup`'s to create a new fd, and assumes that the socket is that new file descriptor minus one. Compile the `shelldupasm.c` code with the `"-stack"` argument. Then use `"objdump -d shelldupasm"` to extract the shellcode. You may also use Steve's `odfhex.c` program from <http://vividmachines.com/shellcode/shellcode.html>.

At this point you will change the default `vulclient`. Since the shell is non-interactive, you will get no response, so your client will need to send a command before waiting for

responses (via the select and read). Have the client sleep for a couple seconds before sending the command, so the command does not get absorbed into the attack request.

Create an attack client that:

- Checks for previous infection
- Fetches password files
- Installs a place marker to prove infection, e.g. something like kilroy was here.
- Downloads itself
- Launches attacks on other machines in the lab. Since we are constraining our attack to the lab, you can use a very simplistic target address function.

Use a unique name for your files, so the attacks do not collide.

## **Hand-in Items**

- Design notes. How does your worm propagate itself? What strategy would you use to really find potential victim addresses.
- Source code for attack client and any other changed code.
- Outline a design that you could use if the stack itself was marked non-executable.