Virtual Environment Installation

The easiest way for everyone to have the same environment is to use *virtual machine* that is pre-configured with all of the material for the course. By doing this, there should be minimal issues during the course with the environment. If you get stuck or confused, please watch the *installation video* which will be released prior to the workshop. If, at this point, you are still stuck, please contact one of the instructors via the *slack* channel. If you are trying to get the most out of the workshop then install this VM prior to the workshop; installing during the workshop would some time that could be better sent learning the material at HackFest.

The environment is configured with useful tools that are needed for heap exploitation:

- Pwntools
- Pwndbg
- An ideal debugging environment. This has multiple custom LibC compilations with debugging symbols and removed optimizations from Malloc. Additionally, all of the exercises are automatically loaded with GDB configured to the right version of LibC and the source code for Malloc.
- Tools for incremental steps on the exercises and the challenges.

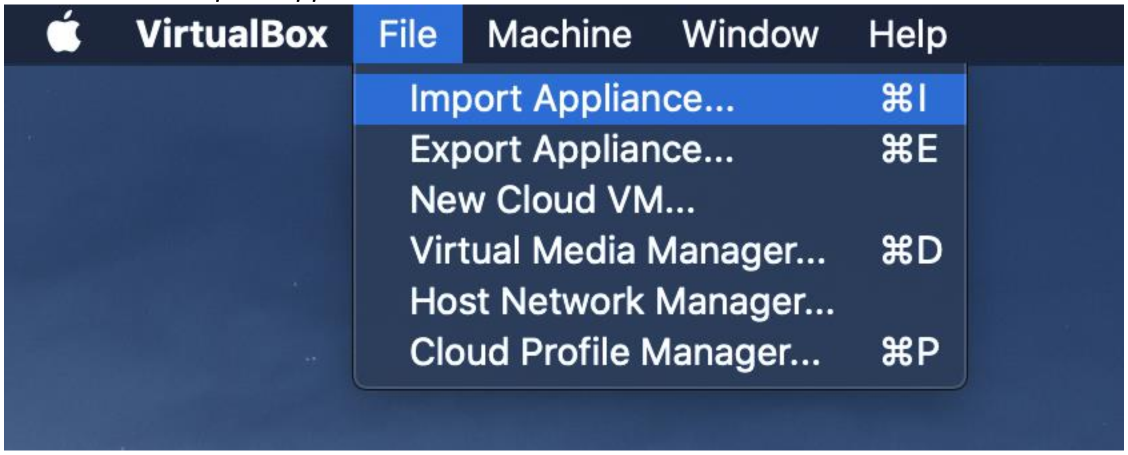
Prior to starting this tutorial there are some prerequisites:

- Virtualbox installed on your current platform. If not, install the latest version of Virtualbox.
    - NOTE: I have not personally tested this on VMWare (Elizabeth has as well) but other people have installed it with success. I recommend using Virtualbox because of guest extension pack already being within the VM for Virtualbox. None-the-less but this is still an option for you.
- Virtualbox extension pack for the **same** version of Virtualbox installed; the extension pack is tied to the current version of Virtualbox. This allows for features such as easy mouse integration and copy-paste across hosts. Please upgrade to the next version of Virtualbox for this.
- At least 5GB of free space on the device.
- Join the slack channel! This is where the slides, solution guides and questions will be answered at. Here's a link to join: https://join.slack.com/t/heapexploitatation/shared_invite/zt-1z8zc3h4y-ahf892cha7Hth2D_k_tNeA

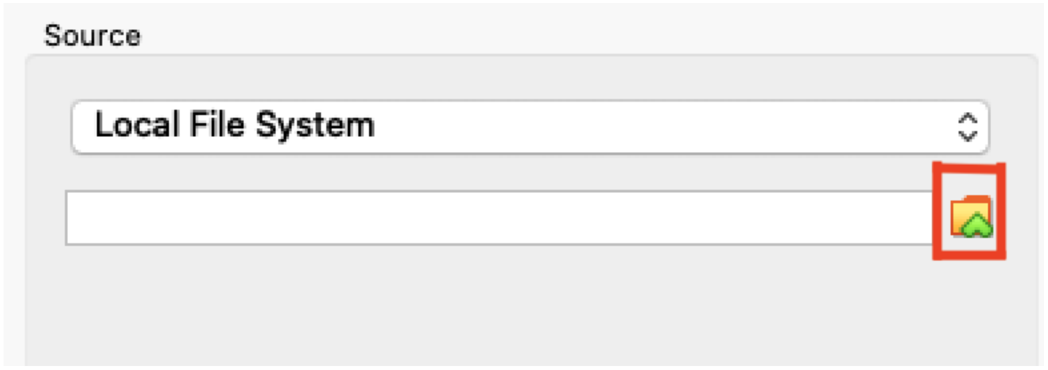
Virtual Machine Installation Steps

These instructions were written for installation on a Macbook. The steps are assumed to be the same on all OS's (Windows, Linux, etc.) unless otherwise noted below.

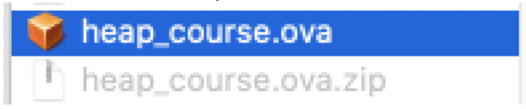
1. Download the virtual machine from the link at here . The VM is 3 gigabytes in size. Make sure to give yourself ample time to download the file.
2. Click on *File->Import Appliance*.



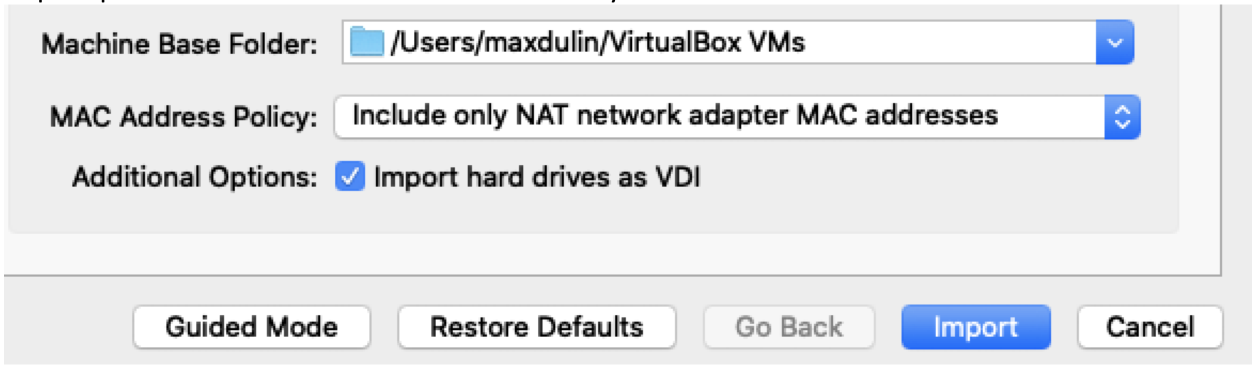
3. Click on the folder icon. This can be seen in the *red box* should below.



4. Click on the Heap Course file. This should be "*Heap Course DEFCON 31.ova*".



5. The configuration options for RAM, CPU, and everything else are the default on our systems. However, feel free to modify these as you see fit.
6. Once you are happy with the configuration options, click on *Import*. This should start the import process of the virtual machine. This may take a few minutes.

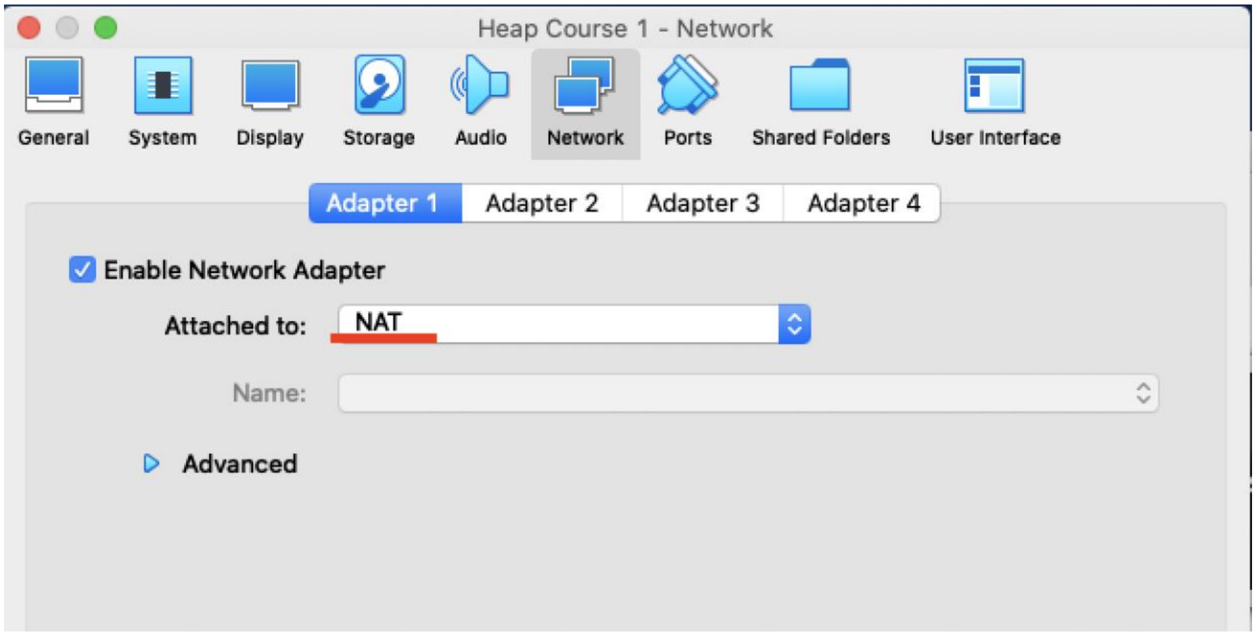


Machine Base Folder: /Users/maxdulin/VirtualBox VMs

MAC Address Policy: Include only NAT network adapter MAC addresses

Additional Options: ✓ Import hard drives as VDI

7. Once the import process has completed, we need to confirm a few *settings*. First, go to *Settings->Network*. Once here, confirm that the networking is set to NAT. This allows for the VM to have internet.

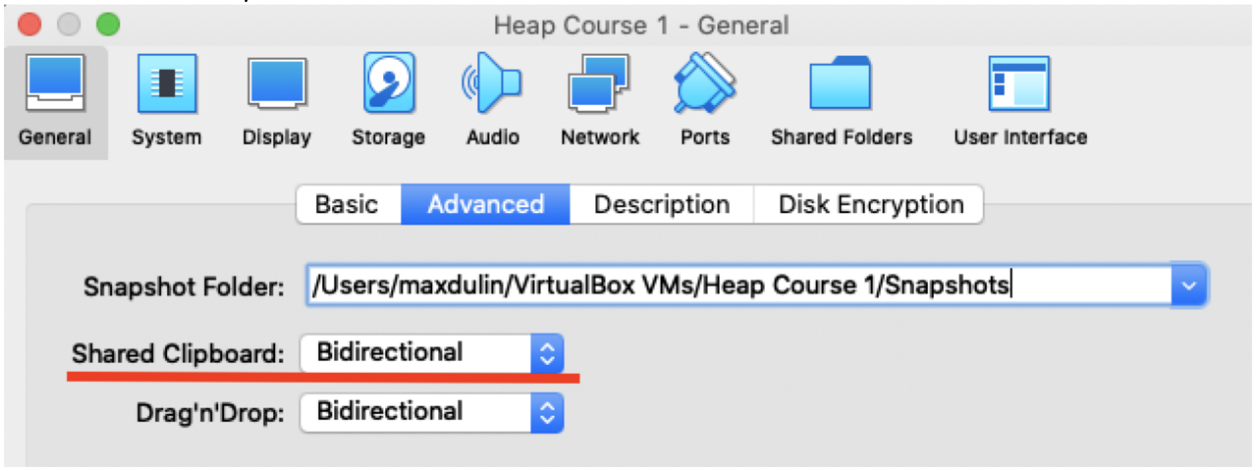


Enable Network Adapter ✓

Attached to: NAT

Name:

8. The second configuration item to check is under *Settings->General->Advanced.* Validate that the *Shared Clipboard* is set to *Bi-Directional*.



Snapshot Folder: /Users/maxdulin/VirtualBox VMs/Heap Course 1/Snapshots

Shared Clipboard: Bidirectional

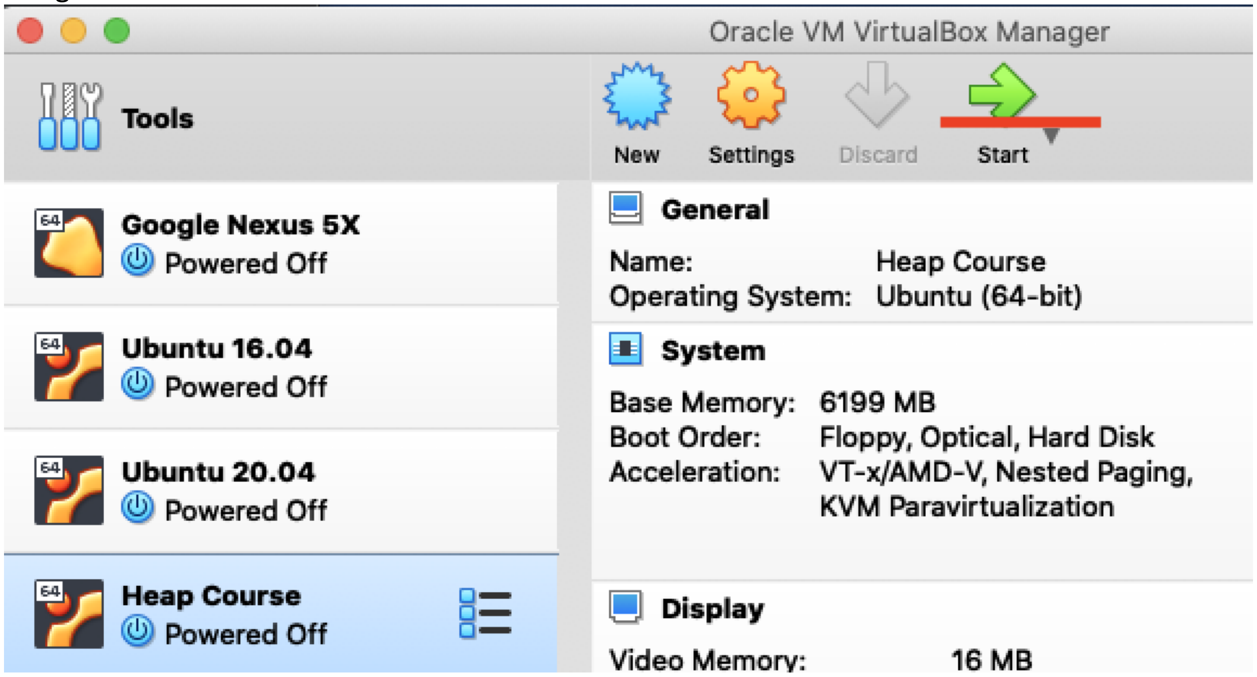
Drag'n'Drop: Bidirectional

9. Optionally, the CPU, Base Memory and other settings can be changed to fit the system. By default, they are relatively low. In order to change this, go to the *Settings->System* section for these settings. To make the VM run as fast as possible for the course, it is recommended that as many CPUs and memory are given to the system as possible.

Virtual Machine Testing

Congratulations! The VM is now installed and ready to go. Now, it is time to give it a try.

1. Turn on the VM. This can be done by clicking on the VM (Heap Course) and clicking on the green start arrow.

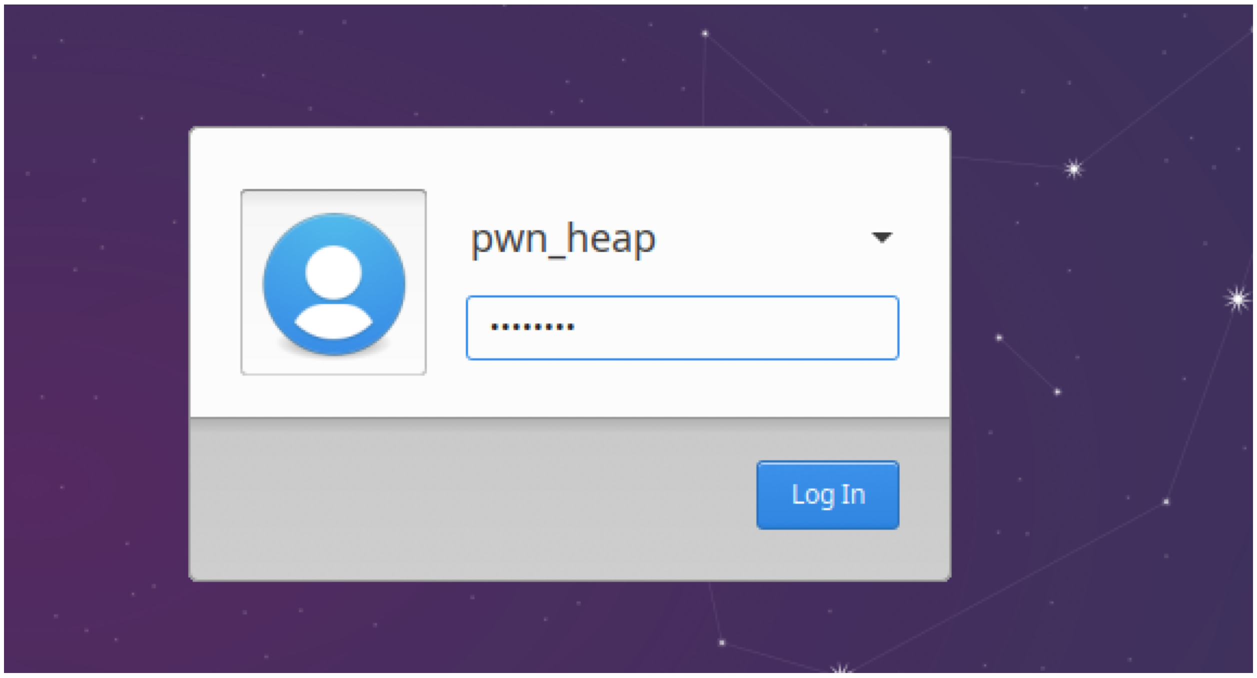


Name: Heap Course
Operating System: Ubuntu (64-bit)

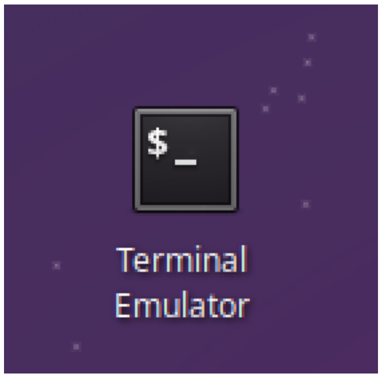
Base Memory: 6199 MB
Boot Order: Floppy, Optical, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, KVM Paravirtualization

Video Memory: 16 MB

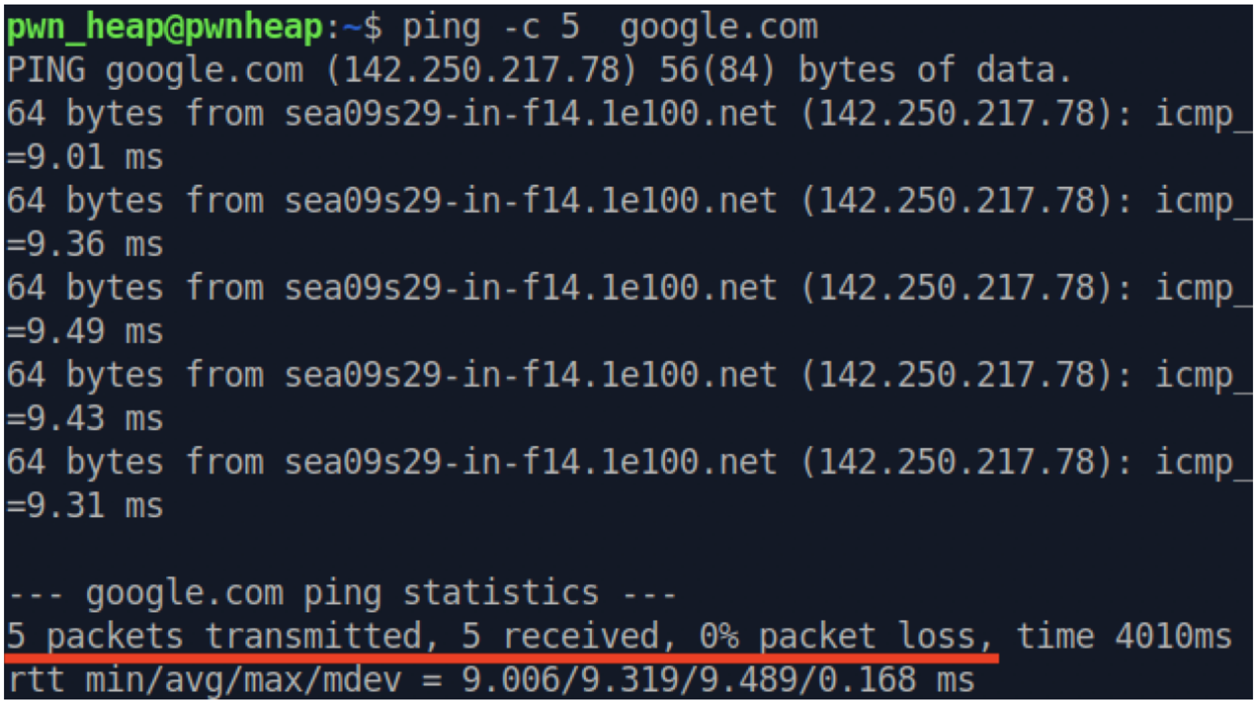
2. Once the virtual machine has finished booting, a login screen should appear. Use the username *pwn_heap* and the password *pwn_heap* to log into the VM. If all of this works, you should see a nice Linux desktop environment.



3. Open up the *Terminal Emulator* application. This can be done by clicking on the terminal icon.



4. Run *ping -c 5 google.com* in the terminal. This is to confirm internet access. If there is internet access, then the result should a *0% packet loss.*



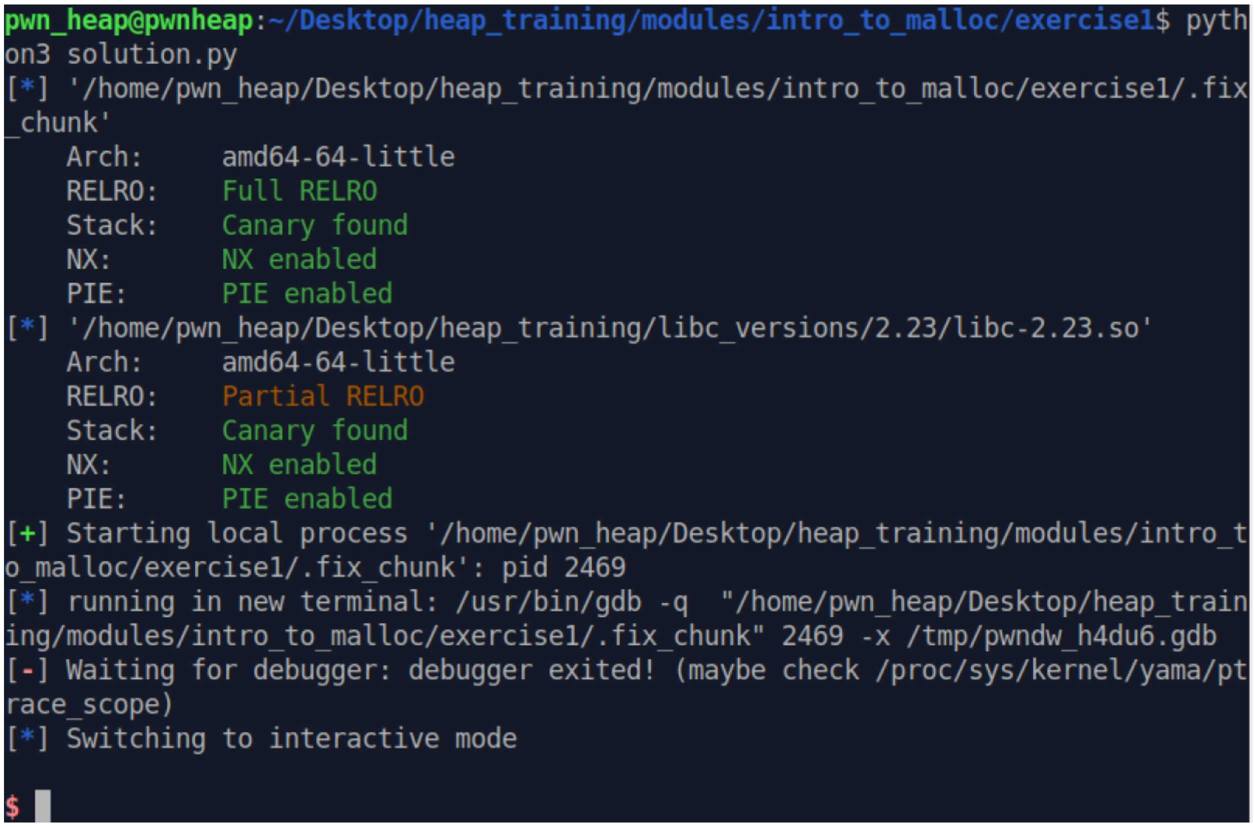
```
pwn_heap@pwnheap:~$ ping -c 5  google.com
PING google.com (142.250.217.78) 56(84) bytes of data.
64 bytes from sea09s29-in-f14.1e100.net (142.250.217.78): icmp_
=9.01 ms
64 bytes from sea09s29-in-f14.1e100.net (142.250.217.78): icmp_
=9.36 ms
64 bytes from sea09s29-in-f14.1e100.net (142.250.217.78): icmp_
=9.49 ms
64 bytes from sea09s29-in-f14.1e100.net (142.250.217.78): icmp_
=9.43 ms
64 bytes from sea09s29-in-f14.1e100.net (142.250.217.78): icmp_
=9.31 ms

--- google.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4010ms
rtt min/avg/max/mdev = 9.006/9.319/9.489/0.168 ms
```

5. Next, we are going to test to ensure that the environment for debugging is setup properly. Move to the *heap_training* directory by executing the command below.

```
cd ~/Desktop/heap_training/modules/intro_to_malloc/exercise1
```

6. After the command in step 5, the directory is in the first exercise of the workshop. Now, run `python3 solution.py`. This will start up the *solution* for the challenge. All of the templates in the modules automatically load the binary with the right version of LibC, debugging symbols and source code linked to *malloc*. The screenshot below is the script starting up and loading the binary in order to interact with it. NOTE: Even though this claims to have an error with GDB, everything loads fine; this error message should be



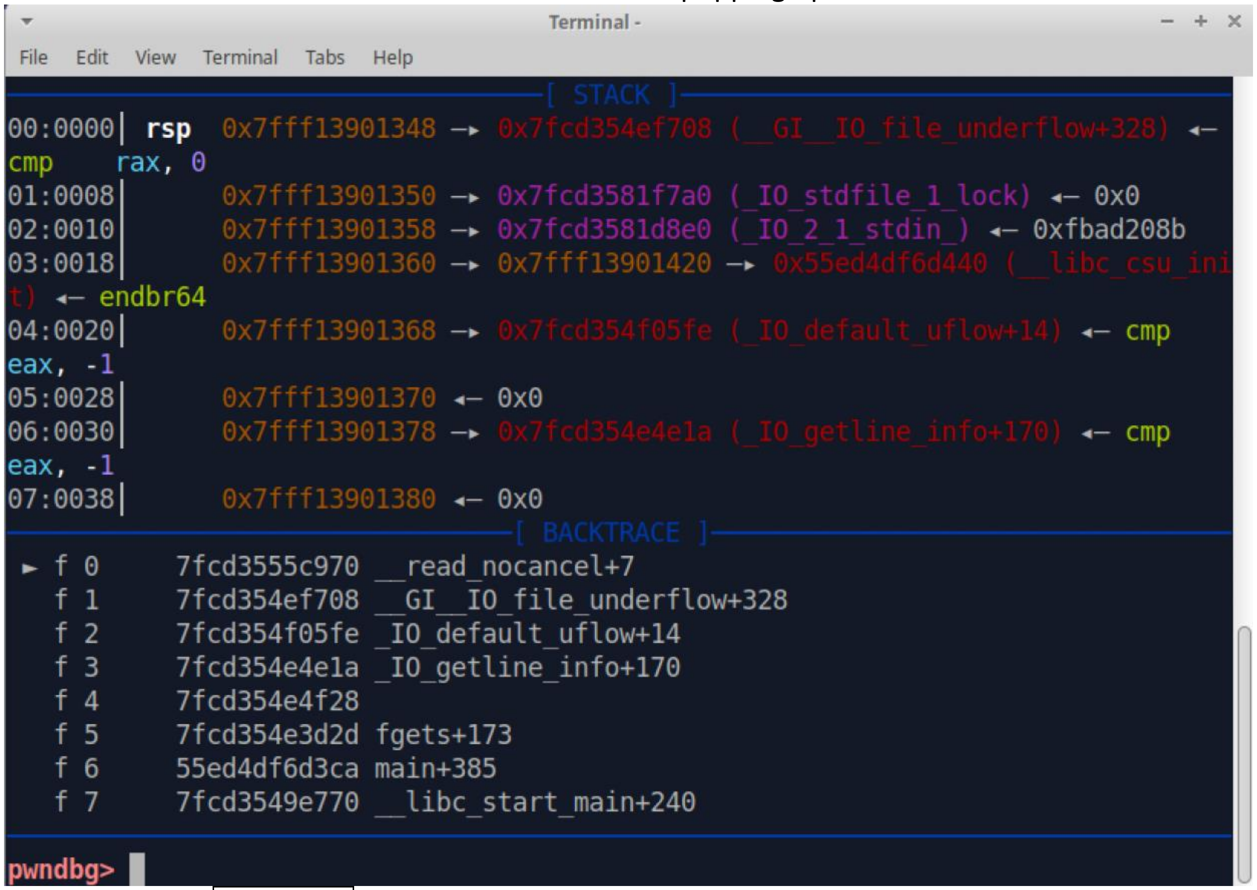
```
pwn_heap@pwnheap:~/Desktop/heap_training/modules/intro_to_malloc/exercise1$ pyth
on3 solution.py
[*] '/home/pwn_heap/Desktop/heap_training/modules/intro_to_malloc/exercise1/.fix
_chunk'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[*] '/home/pwn_heap/Desktop/heap_training/libc_versions/2.23/libc-2.23.so'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[+] Starting local process '/home/pwn_heap/Desktop/heap_training/modules/intro_t
o_malloc/exercise1/.fix_chunk': pid 2469
[*] running in new terminal: /usr/bin/gdb -q  "/home/pwn_heap/Desktop/heap_train
ing/modules/intro_to_malloc/exercise1/.fix_chunk" 2469 -x /tmp/pwndw_h4du6.gdb
[-] Waiting for debugger: debugger exited! (maybe check /proc/sys/kernel/yama/pt
race_scope)
[*] Switching to interactive mode

$ 
```

ignored.

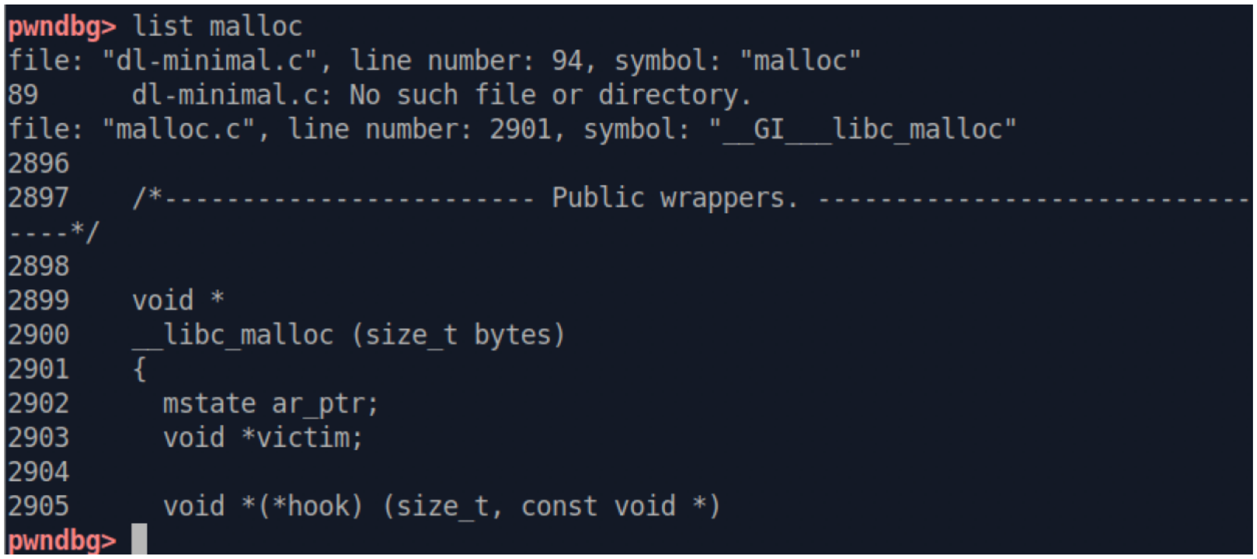
7. The previous command should have started the binary. Alongside this, a GDB window should appear. The program is automatically stopped once the GDB window is

attached. The screenshot below is the GDB window popping up.



```
─────────────────────────────────[ STACK ]─────────────────────────────────
00:0000│ rsp  0x7fff13901348 —▸ 0x7fcd354ef708 (__GI__IO_file_underflow+328) ◂—
cmp     rax, 0
01:0008│      0x7fff13901350 —▸ 0x7fcd3581f7a0 (_IO_stdfile_1_lock) ◂— 0x0
02:0010│      0x7fff13901358 —▸ 0x7fcd3581d8e0 (_IO_2_1_stdin_) ◂— 0xfbad208b
03:0018│      0x7fff13901360 —▸ 0x7fff13901420 —▸ 0x55ed4df6d440 (__libc_csu_ini
t) ◂— endbr64
04:0020│      0x7fff13901368 —▸ 0x7fcd354f05fe (_IO_default_uflow+14) ◂— cmp
eax, -1
05:0028│      0x7fff13901370 ◂— 0x0
06:0030│      0x7fff13901378 —▸ 0x7fcd354e4e1a (_IO_getline_info+170) ◂— cmp
eax, -1
07:0038│      0x7fff13901380 ◂— 0x0
───────────────────────────────[ BACKTRACE ]───────────────────────────────
 ► f 0     7fcd3555c970 __read_nocancel+7
   f 1     7fcd354ef708 __GI__IO_file_underflow+328
   f 2     7fcd354f05fe _IO_default_uflow+14
   f 3     7fcd354e4e1a _IO_getline_info+170
   f 4     7fcd354e4f28
   f 5     7fcd354e3d2d fgets+173
   f 6     55ed4df6d3ca main+385
   f 7     7fcd3549e770 __libc_start_main+240
───────────────────────────────────────────────────────────────────────────
pwndbg>
```

8. In GDB, use the `continue` command to allow the rest of the program to work.
9. Now, we will check to see if the LibC source code has been loaded in properly. This can be done by running the command `list malloc` in GDB. If the *malloc* source code appears, then the debugging environment is setup properly



```
pwndbg> list malloc
file: "dl-minimal.c", line number: 94, symbol: "malloc"
89      dl-minimal.c: No such file or directory.
file: "malloc.c", line number: 2901, symbol: "__GI___libc_malloc"
2896
2897    /*------------------------ Public wrappers. ----------------------------
----*/
2898
2899    void *
2900    __libc_malloc (size_t bytes)
2901    {
2902      mstate ar_ptr;
2903      void *victim;
2904
2905      void *(*hook) (size_t, const void *)
pwndbg>
```

10. If you have gotten this far, you are good to go! The debugging environment is setup, the VM works and you are ready to tackle this new challenge. If you wanted to prepare for the course more, go review the basics of C programming and Python (particularly *pwntools*). Email me at mdulin@securityinnovation.com or join the Slack channel that is posted to ask questions.