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 (you’re going to DEF CON, of course you are), then install this VM prior to the workshop; installing during the workshop would some time that could be better sent leaning the material at DEF CON.

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.
* 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.
* At least 10GB of free space on the device.

# 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](https://drive.google.com/file/d/14a38rDqBw1gIkA3tR9gi9SwYHXy79N8r/view?usp=sharing)**.** The VM is 5 gigabytes in size. Make sure to give yourself ample time to download the file.
2. Unzip the zip file to output the OVA file.   
   **NOTE**: The default MacOS unarchiving tool will unzip the zip and the **OVA** file; this is bad! If after unzipping you see a folder with an *ovf* file inside of it, OVA file was also unzipped; trying to import this will not work. In this case, use the *unzip* command via the terminal CLI or another tool to unzip the zip file properly. The command *unzip* ***heap\_course.zip*** *heap\_course* will unzip the files properly and the OVA file will be in the folder *heap\_course*.
3. Click on *File->Import Appliance*.   
   A screenshot of a computer

   Description automatically generated with medium confidence
4. Click on the folder icon. This can be seen in the *red box* should below.   
   Graphical user interface, text, application

   Description automatically generated
5. Click on the Heap Course file. This should be *heap\_course.ova*. Graphical user interface, text, application

   Description automatically generated
6. 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.
7. 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.  
   Graphical user interface, text, application, email

   Description automatically generated
8. 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.   
   Graphical user interface, application

   Description automatically generated
9. The second configuration item to check is under *Settings->General->Advanced.* Validate that the *Shared Clipboard* is set to *Bi-Directional*.   
   Graphical user interface, text, application

   Description automatically generated
10. 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.   
   Graphical user interface, text, application

   Description automatically generated
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.

Graphical user interface, application

Description automatically generated

1. Open up the *Terminal Emulator* application. This can be done by clicking on the terminal icon.  
   Graphical user interface

   Description automatically generated
2. 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.*  
   Text

   Description automatically generated
3. 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

1. Text

   Description automatically generatedAfter 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 ignored.
2. 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.  
   Text

   Description automatically generated
3. 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  
     
   Text

   Description automatically generated
4. 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*). This information can be found in the *preparation* slack channel.