**Introduction to PHP Exploitation: Include and POST Vulnerabilities**

By: Noah Dunn

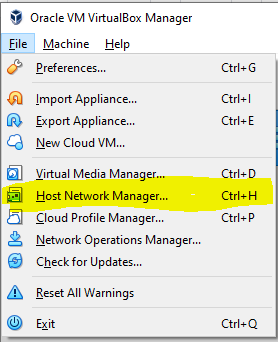
**Note:** The following lab was derived entirely using the help of the listed two resources. If at any point you get stuck, consult the following first for any necessary clarifications.

<http://www.sec-art.net/2018/03/how-to-install-web-for-pentester-vm-in.html>

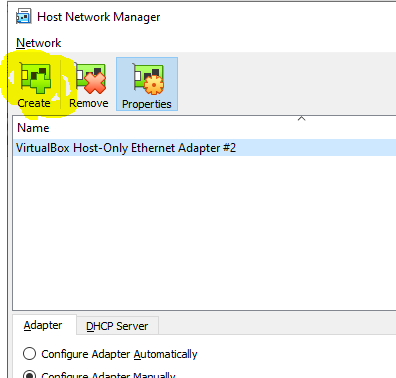
<https://pentesterlab.com/exercises/php_include_and_post_exploitation/course>

**Step 1: Setting up VirtualBox**

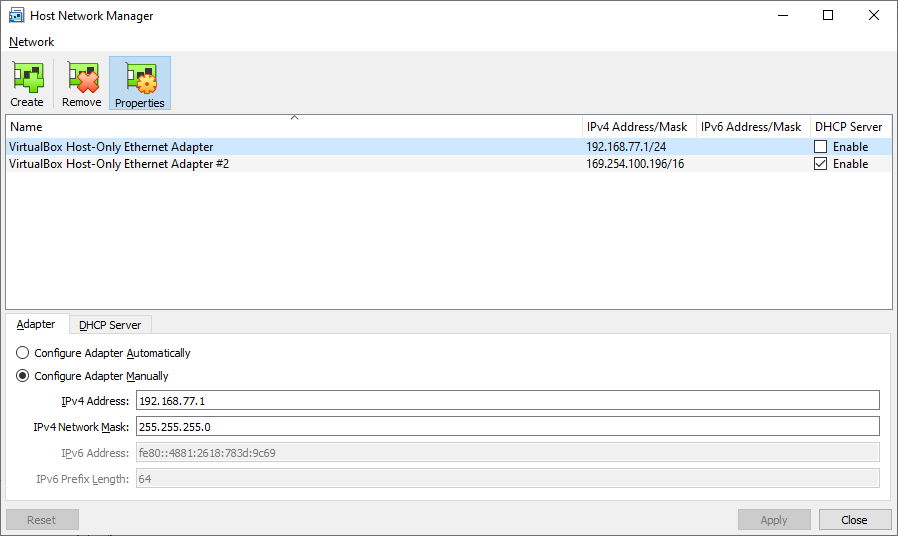
1. Download the ISO for this lab available [here](https://pentesterlab.com/exercises/php_include_and_post_exploitation/iso).
2. Download and install VirtualBox available [here](https://www.virtualbox.org/wiki/Downloads).   
   **NOTE:** You can do this through the CLI version of VirtualBox, but I recommend using the visual installer and setup tool as the instructions will be provided in this form.  
   **NOTE:** VirtualBox has a habit of acting like complete garbage dependent on system, CPU architecture, and Operating System it is run on. It is however free. Any minor issues that come up **SHOULD** be Google-able, but in the strange case of an unresolvable impasse, feel free to try mounting the ISO to any other Virtualization software that enables digital disks. (You could even do it on your host box, but I prefer to keep anything dealing with potential vulns off my system, even if I think I’m the only one on)
3. Load VirtualBox, accepting the default settings.
4. A **VERY IMPORTANT NOTE BEFORE YOU BEGIN**: This was tested on my personal network at home, where I have free range to spin up web servers on my local host as needed. If you are on public wifi, you may have to modify the IP Addresses in the following steps in order to accommodate your needs (In the case of a potential IP conflict). I believe the ‘Host only’ network adapter only configures a spoof network for your device, but I am not confident enough to leave this section out.
5. Select the **‘Host Network Manager’** option from the **File** dropdown in the top left corner of the options bar



1. Select the **CREATE** button



1. Make sure **Properties** is selected for the adapter that you just created



1. Select **Configure Adapter Manually** if it is not already selected and type in the following values:

**IPv4 Address : 192.168.56.1**

**IPv4 Network Mask : 255.255.255.0**

1. Select the **DHCP Server** tab, click **Enable**, and input the following values

**Server Address : 192.168.56.1**

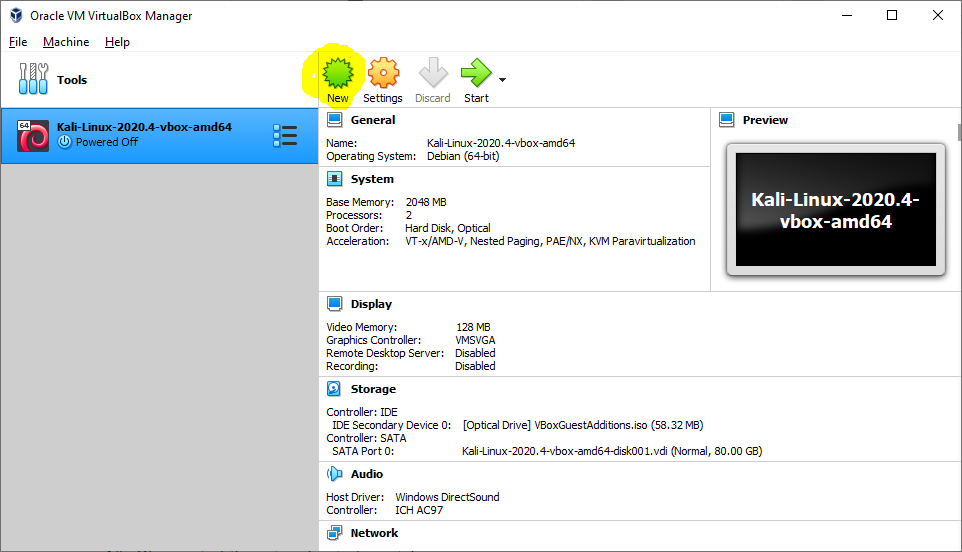
**Server mask : 255.255.255.0**

**Lower Address Bound : 192.168.56.101**

**Upper Address Bound : 192.168.56.254**

1. Close out of the network interfaces tab
2. Select the New button

**NOTE:** You will not see the Kali Instance, this is something I have installed for personal use ;)



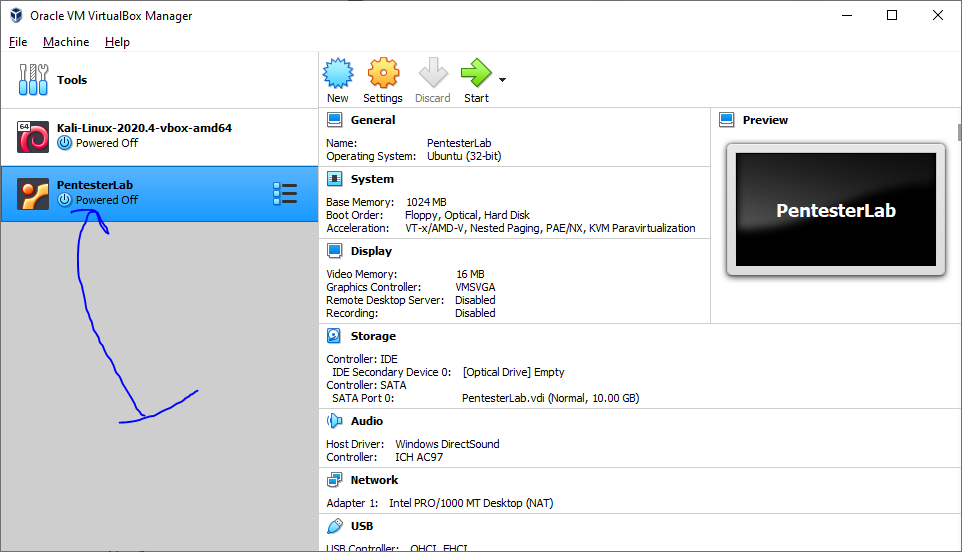
1. In the dialog box input the following values:

**Name : PentesterLab**

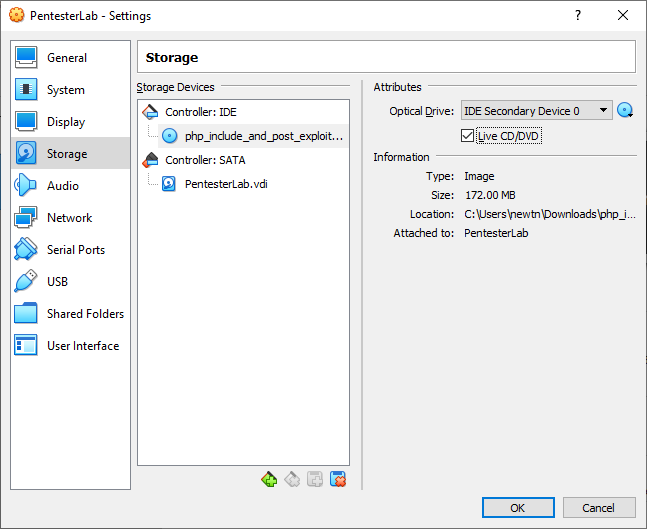
**Type : Linux**

**Version : Ubuntu (32-bit)**

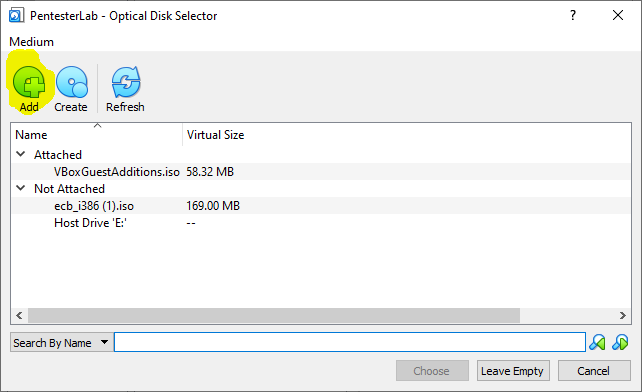
1. Click **NEXT** through all the dialogue defaults
2. You should now have an instance of PentesterLab saved



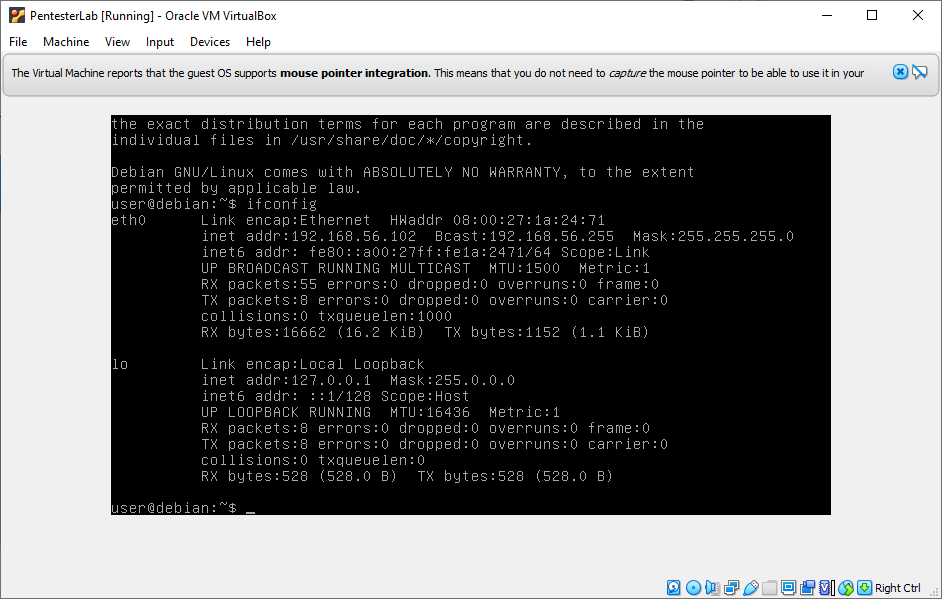
1. Right Click on **PentesterLab**, click **Settings**,and pull up the **Network** tab
2. Ensure that **Enable Network Adapter** is checked, Attached to **Host-only Adapter**, and make sure the **Name** entry matches the network adapter we just created.
3. Select the **Storage** tab. Under the **Controller: IDE** tab, click the Empty CD tab and select the ISO downloaded in Step 1 using the **Optical Drive** drop down on the right hand side, hitting **Choose/Create a Virtual Optical Disk**. Additionally, make sure **Live CD/DVD** is selected



1. Select **Add** and pick the ISO from before. Hit **OK**.



1. Double click PentesterLab to load it. After everything is finished, type ifconfig, and grab the IP address available (In this case, **192.168.56.102**).



1. This virtual machine needs to remain running in the background for the duration of the lab. On your host machine, type in the IP address in your browser (In my case: [**http://192.168.56.102/**](http://192.168.56.102/))
2. If this does not pop up immediately, quickly, like I said before, VirtualBox can be janky. Sometimes a restart of VirtualBox or a redo of the steps on a new instance will do the trick. In my case.
3. As soon as you can view this page, you are ready to move on to Step 2

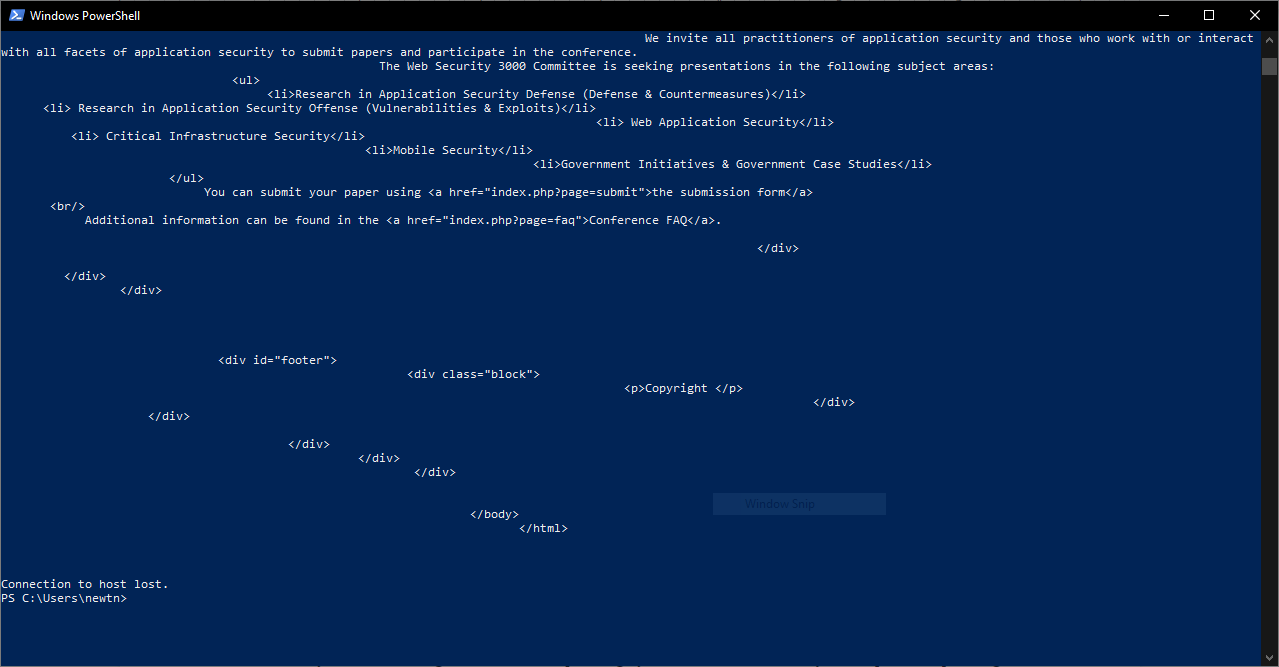


**Step 2: Fingerprinting Web Applications**

**Fingerprinting** as defined in Cyber + Network security refers to gathering as much information as possible about a given system (Operating System, Applications at Use, etc..) as provided by the system itself. This stage is normally early on in any cyber related investigation/attack, as it enables the investigator/hacker to determine what tools are at their disposal.

1. Locate the IP Address that was used to connect to the server in the browser in the previous section, (In my case, it was **192.168.56.102**).
2. If you are not using a Linux based operating system, install Telnet for your respective operating system available [here](https://support.code42.com/CrashPlan/6/Troubleshooting/Test_your_network_connection#Install_Telnet).
3. Powershell will work for this on Windows. On Mac, you should be able to use the built in terminal, in Linux you can just use the terminal.
4. Execute the command

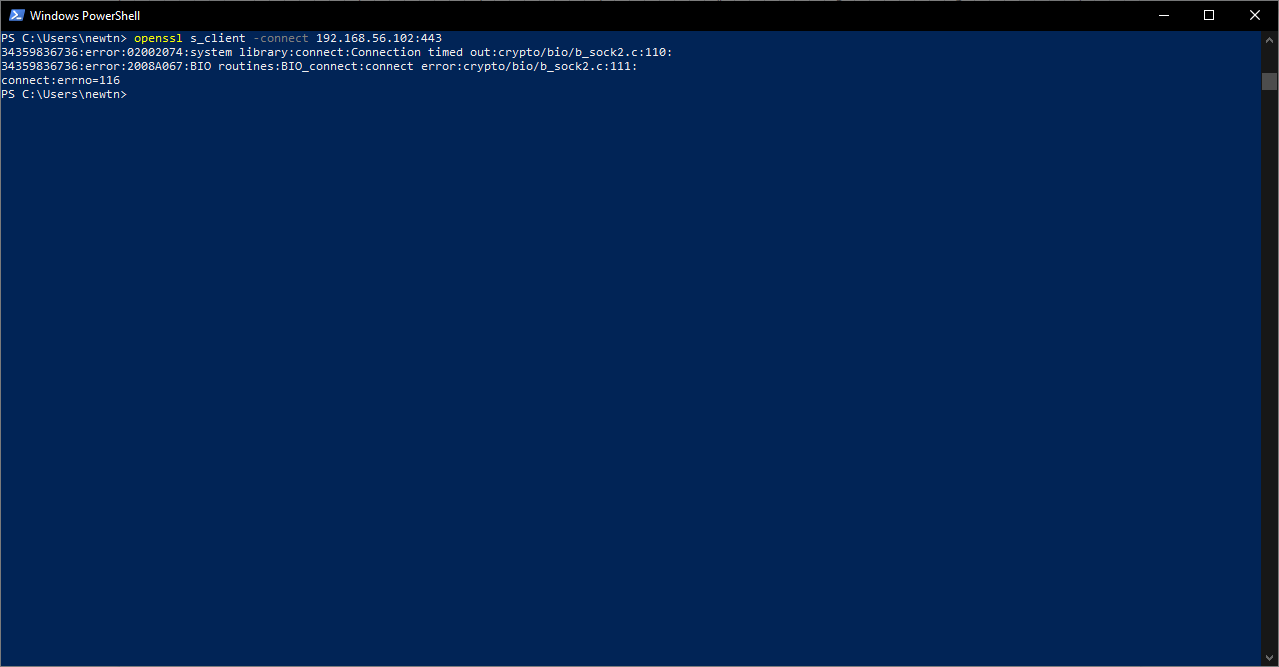
*telnet YOUR\_IP\_HERE 80*

**NOTE**: On Windows Powershell, you also need to perform a **CTRL+C** followed by an **ENTER** in order to get the text to pop up for some reason. This may be the same on other operating systems  


1. The system should respond with the HTML. This means the Web Server serving the **HTTP** traffic is located on port **80**, which is very common. If telnet would have replied with nothing, this would mean all traffic was being routed over **HTTPS**, or, the web server is hosted on a port that is not 80.
2. Execute the command

*openssh s\_client -connect YOUR\_IP\_HERE:443*

You should receive a timeout upon execution of this command. This will indicate that nothing is running on Port **443**. For now, we can assume that the web server does not have any traffic running using **HTTPS**.

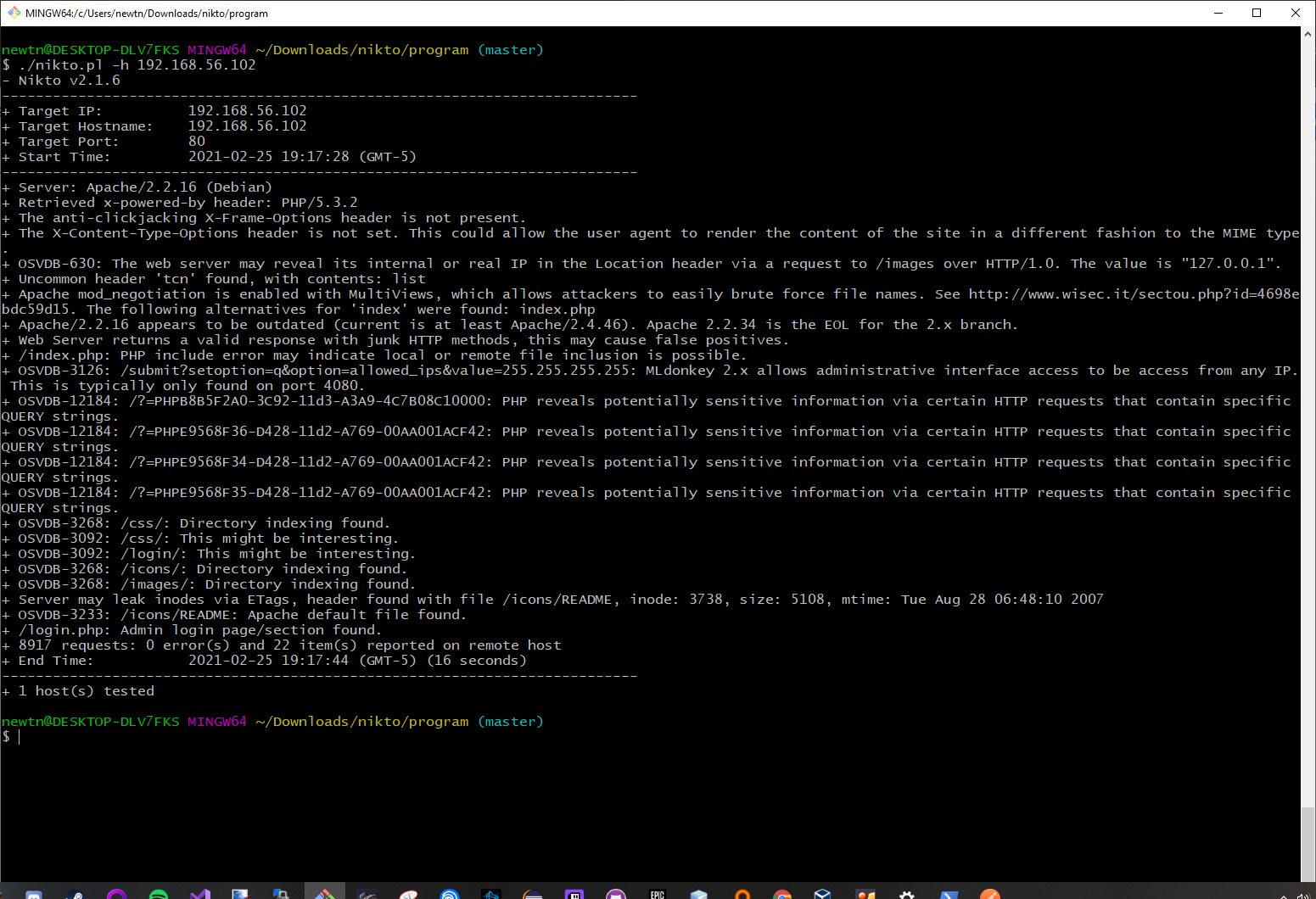


1. **Nikito** is a web server scanner that started development in 2001, and is still in development today. It is useful for discovering and enumerating services that are being used on a remote host. Additionally, it points out any known vulnerabilities associated with anything it finds.

Execute the following:

git clone https://github.com/sullo/nikto

cd nikto/program

./nikto.pl -h *YOUR\_IP\_HERE  
*

1. This can be an overwhelming amount of information, but there are some key figures to point out here.
   1. Apache is running version **2.2.16**
   2. PHP is running version **5.3.2**
   3. A potential PHP include vulnerability, denoted by the line ‘PHP include error may….’
   4. Directories we did not know about before (**css, login, icons, images**)
   5. A login page located at **login.php**
   6. A list of potential vulnerabilities (all the OSVDBs). If we investigated further, we would determine these are all actually false positives thrown by Nikito.
2. With all this information, we have enough to get started down the exploitation rabbit hole

**Step 3: Detection and Exploitation of PHP Include**

Old School PHP was a maintainability nightmare. People used to keep files of necessary code for all their individual pages, and just copy-paste huge swaths of repeated code all over the place. Modern PHP makes use of two conventions to import code from other files: **include** and **require**. As discovered in the previous example, there may be something problematic with a PHP include error.

In particular, exploitation involving the require and include statements occur when the programmer of a PHP application enables **some user control over path location**, as opposed to a **static, predisposed path**. This is very bad coding practice, but if programmers did not make bad coding mistakes, exploitation would be much more difficult. A very trivial version of a PHP path exploit would be the following:

<?php

include("header.php");

include($\_GET["page"]);

?>

This is trivial because this enables the user to grab a PHP file from anywhere on the web and execute the code located there, simply by providing a GET request with the correct header. Notice this code does no pre-processing or validation checking.

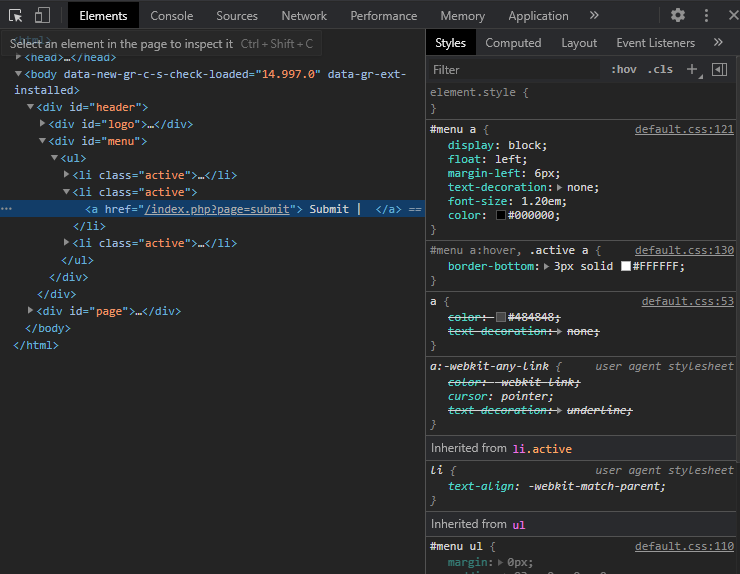
The above example is a case of **remote file include**, which means files can be pulled from anywhere on the web. The second type is known as **local file include**, which means we can only make use of files on the host web-server’s filesystem.

Normal targets for path exploitation tend to come up by using external tools like **Nikito, Linpeas, or GoBuster**. We already are aware from the Nikito output that a PHP include vulnerability may be lurking, so instead of trying random paths, let’s see if we can find some query strings for page redirection in the HTML.

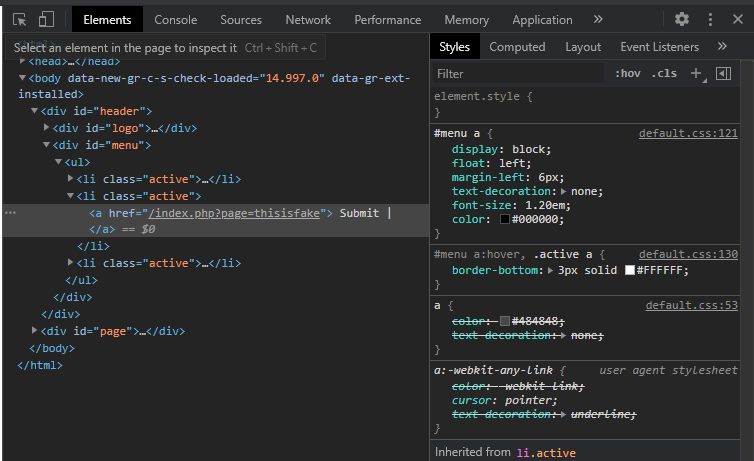
1. Execute the following command  
   curl YOUR\_IP\_HERE
2. A quick investigation of the HTML yields the following line:

<a href="/index.php?page=submit"> Submit | </a>

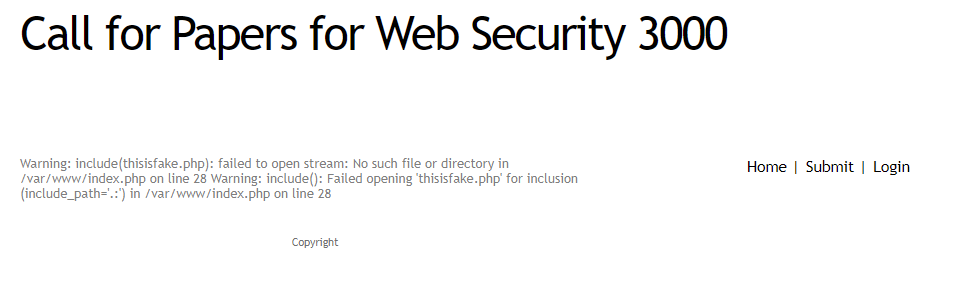
1. Bingo, we now know what kind of URL querying that the PHP might be using in it’s PATH variables. Let’s try a slight modification.
2. In your browser, Right-Click and inspect the Submit button.



1. Double Left-Click the href tag and modify the value to something simple, like ‘index.php?page=thisisfake’.



1. Click the Submit button

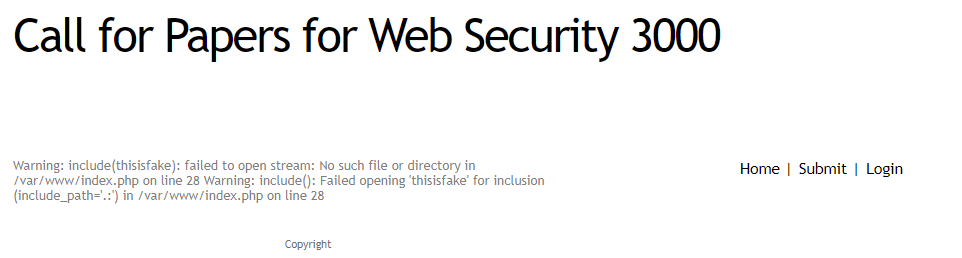


1. Sweet. We have confirmed that we have the ability to access local file systems. Now let's determine if this application is foolish enough to have their php configuration file set up to enable remote include statements.
2. We do not have external internet connections configured for this particular lab, if we did, we could use the string **‘index.php?page=http://www.google.com/?’.** This would actually embed the html for the google homepage directly in the page.



Since we do not have access to remote for this lab, let’s go back to our local example. What happens when we want to access files that don’t have a **.php** tacked on the end? Lucky for us, there is an exploit for this particular version of PHP that lets us manipulate that part of the code as well

1. Try running steps 4-6 again but this time, use the string **‘index.php?page=thisisfake%00’**



1. Look at that! No more **.php**. The %00 is the Null Byte, which cuts off the rest of the code line execution, enabling us to cut off everything after the string we put in the code. Let’s try one last exercise.
2. Use the string **‘index.php?page=../../../../../etc/passwd%00’**

**NOTE:** The number of ‘..’ to use to trace our way back up the path can really only be determined through experimentation.



1. Just like that, we have access to the password file.

**NEXT TIME**

Remote Code Execution