Group A

Guilherme Pereira

picoCTF 2022 - file-run1 & file-run2

```
Both challenges are ELF files:
    - writeups/run1: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), corru
pted program header size, corrupted section header size
    - writeups/run2: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), corrup
ted program header size, corrupted section header size
Thus, by simply running the command ´strings run´ the flag is returned in the outpu
t:
    - strings run1 > picoCTF{U51N6_Y0Ur_F1r57_F113_9bc52b6b}
    - strings run2 > picoCTF{F1r57_4rgum3n7_be0714da}
```

WordPress Login Brute Force

```
    For the WordPress login brute force my initial approach was to run the "wpscan"
tool. From the scan I was able to verify that the "Login" action could
be carried out in two ways, either by sending POST requests to the xmlrpc.php endpo
int (E10) or via the wp-login.php endpoint. The downfall of
using the xmlrpc.php endpoint was that the "lockdown" timer, triggered when two log
in requests are sent, couldn't be viewed. Thus, there would
be no way of knowing when the timer has started. With the wp-login.php endpoint, th
e timer could be viewed by parsing the resulting HTML from
the login request (div with id=login_error). This allowed for a dynamic brute force
attack where delays could be acknowledged with minimization
of wastely login requests.
```

In [ ]:

```python
# E10: xmlrpc.php request
def xml_req(s, pwd, user="think"):
    xml = f"""<methodCall>
    <methodName>wp.getUsersBlogs</methodName>
    <params>
    <param><value>{user}</value></param>
    <param><value>{pwd}</value></param>
    </params>
    </methodCall>"""
    headers = {
        'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:55.0) Gecko/2010
0101 Firefox/55.0',
    }
    r = s.post("http://localhost/cybersec/xmlrpc.php", headers=headers, data=xml)
```

```
    The next step was to develop a tool that could be used to exploit the faulty wp
-login.php page. Since two requests would result in a 10 minute login
  lockdown (independently of the timeout between these two requests), any brute for
ce tool would be limited by this measure.

    The tool that I developed has two modes of operation, a sequential loop (singl
e process) that attempts to authenticate a user, in this case either
  "admin" or "think", with the candidate password or a "divide and conquer" approac
```

"admin" or "think", with the candidate password of a "divide and conquer" approac
h (multi process) that splits the total number of candidate passwords
 by the the number of available CPU cores and assigns a process for each chunk to attempt authentication, relevant timeout delays and orchestration is
 done using a shared mutual exclusion lock and a shared dictionary object. The code also has function specific documentation, bellow I will display
 the help menu (with default values) and relevant commands to run the tool.

```
usage: main.py [-h] [-p PAYLOAD_SIZE] [-u USERNAME] [-ps PASSWORD_PATH] [--url URL]
[--redirect REDIRECT] [-t | --threads | --no-threads]

options:
  -h, --help            show this help message and exit
  -p PAYLOAD_SIZE, --size PAYLOAD_SIZE
                        2
  -u USERNAME, --username USERNAME
                        admin
  -ps PASSWORD_PATH, --password PASSWORD_PATH
                        passwd/
  --url URL             /wp-login.php
  --redirect REDIRECT   /wp-admin
  -t, --threads, --no-threads
```

> **pip install -r requirements.txt**
>
> **python3 main.py**

After an overly excessive amount of trials I was able to conclude that "admin"s password is "#1mama" and "think"s password is "123".

References:

- https://servebolt.com/articles/xmlrpc-php/#bruteforce
- https://stackoverflow.com/questions/2332765/what-is-the-difference-between-lock-mutex-and-semaphore/45567101#45567101
- https://stackoverflow.com/questions/31508574/semaphores-on-python