ÉCOLE CENTRALE LYON

CYBERSÉCURITÉ

Rapport BE 2 : Root-me

Étudiants :
Victor LUDVIG

Encadrants :
Charles-Edmond BICHOT

14 janvier 2024

Table des matières

I Challenges résolus

J'ai obtenu un nombre total de points de 125. Mon pseudo sur le site est v-l. Voici les 7 challenges que j'ai résolus :

TABLE 1

Nom	Niveau	Points	Mot de passe
Quick Response Code	3	40	POHeyZ6pMvgn
Second degree polynomial solver	3	25	P0lyNom35_4r3_e45y_Bruhh
Suite mathematique	3	20	lFablYE9P1
CAPTCHA me if you can	2	20	dtePZJgVAfaU
Javascript - Authentification 2	1	10	HIDDEN
Javascript - Authentification	1	5	sh.org
Bash - System I	1	5	!oPe96a/.s8d5

II Explications

Voici les explications des problèmes que j'ai résolus, dans l'ordre dans lequel je les ai résolus.

II.1 CAPTCHA me if you can

Le but est de casser un captcha simple en moins de 3 secondes. J'ai fait du web scraping en python pour trouver l'image, j'ai utilisé pytesseract pour convertir l'image en chaine de caractère, et j'ai ensuite écrit le texte automatiquement.
Pour automatiser les interactions avec le navigateur web Firefox, j'ai utilisé le driver marionette. Comme j'ai utilise le WebDriver Selenium de python, il faut traduire les commandes Selenium python pour qu'elles soient comprehensibles par Marionette. Cela est fait avec le geckodriver.

C'était mon premier challenge et j'ai cru qu'il fallait cliquer sur 'Start challenge' à chaque fois qu'on voulait retenter le challenge. En fait, on pouvait juste recharger la page du captcha. Le début de mon code clique donc sur 'Start challenge', ensuite Selenium change de tab pour pouvoir charger la page du captcha en python, et le captcha est cassé avec pytesseract. Ce module ne réussit pas à tous les coups à casser le captcha, il faut faire plusieurs essais.

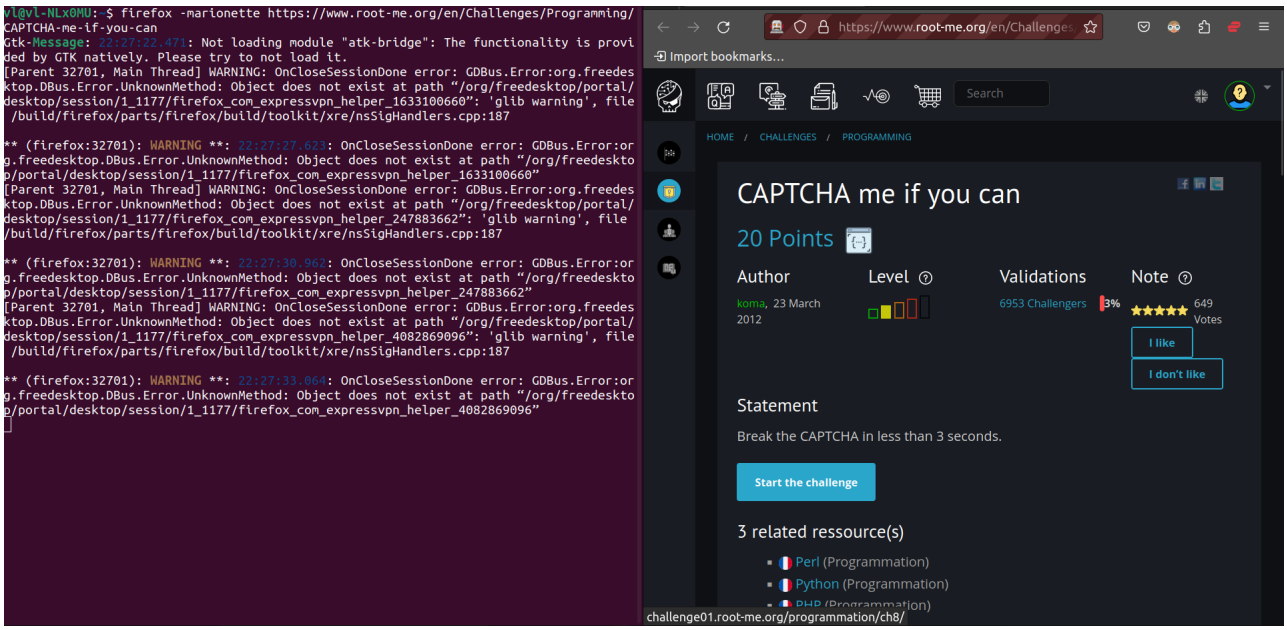


FIGURE 1 – Marionette driver Firefox. On remarque la barre de recherche rouge et le petit robot qui montre que Marionette est activé

Voici le code, de ma méthode. D'autres méthodes n'ont pas recours à Marionette et font directement des requêtes WEB sans passer par un navigateur ce qui a le mérite d'être plus concis.

```
from PIL import Image
import base64
from io import BytesIO
import pytesseract
from bs4 import BeautifulSoup
from selenium.webdriver.firefox.firefox_binary import FirefoxBinary
from selenium.webdriver.firefox.options import Options
from selenium.webdriver.firefox.service import Service
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.firefox.options import Options

options = Options()
options.binary = FirefoxBinary(r'/bin/firefox')
firefox_services =
    Service(executable_path='/home/vl/drivers/geckodriver-v0.34.0-linux64/geckodriver',
    service_args=['--marionette-port', '2828', '--connect-existing'])
driver = webdriver.Firefox(service=firefox_services, options=options)

driver.find_element(By.LINK_TEXT, 'Start the challenge').click()
# Changement de tab
for w in driver.window_handles:
    if w != p:
```

```
        driver.switch_to.window(w)
        break

# Parsing de la page qui contient le captcha
soup = BeautifulSoup(driver.page_source, 'html.parser')
img_tag = soup.find('img')
img_src = img_tag.get('src')
b64 = img_src.split(',')[1]
image_bytes = base64.b64decode(b64)
image_buffer = BytesIO(image_bytes)
image = Image.open(image_buffer)

# Cassage du captcha
captcha = pytesseract.image_to_string(image).replace('\n',
    '').replace('\x0c', '')
input_response = driver.find_element(By.CSS_SELECTOR,
    'input[type="text"][name="cametu"]')
for c in captcha:
    input_response.send_keys(c)
driver.find_element(By.CSS_SELECTOR,
    'input[type="submit"][value="Try"]').click()
```

II.2 Suite mathematique

Le but est de calculer le n-ème terme d'une suite, pour laquelle la relation de récurrence et l'initialisation sont affichées sur une page web. Il s'agit, comme pour l'exercice précédent, de web scraping. Cette fois-ci, je n'ai pas utilisé de navigateur web, j'ai directement fait des requêtes HTTPS en Python. La suite est toujours de la forme :

$$U_{n+1} = [n_1 + U_n] \pm [n \times n_2] \tag{1}$$

Donc, il faut trois expressions régulières pour récupérer n_1, n_2, et le signe de l'opérateur du milieu. Deux autres expressions régulières permettent de récupérer la valeur initiale U_0 et l'indice souhaité pour la sortie.

```
import requests
import re
from bs4 import BeautifulSoup

s = requests.Session()
html_page = s.get('http://challenge01.root-me.org/programmation/ch1/')

soup = BeautifulSoup(html_page.text, 'html.parser')
txt = soup.find('body').text
split = txt.split('\n')
```

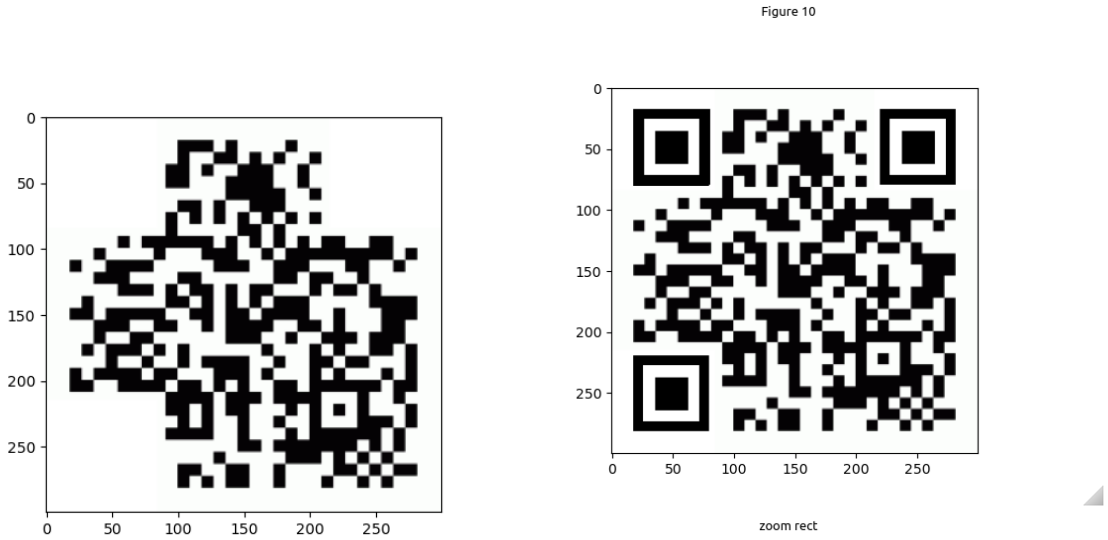
```
n1 = int(re.search(r'\[ (.*) \+ Un \]', split[0]).groups(0)[0])
n2 = int(re.search(r'\[ n \* (.*) \]', split[0]).groups(0)[0])
sign = re.search(r'\] (.*) \[', split[0]).groups(0)[0]
u = int(re.search(r'U0 = (.*)', split[1]).groups(0)[0])
index = int(re.search(r'You must find U(.*)You have only 2 seconds to ',
    split[2]).groups(0)[0])

for n in range(index):
    if sign == '+':
        u = n1+u+(n*n2)
    else:
        u = n1+u-(n*n2)

response =
    s.post(f'http://challenge01.root-me.org/programmation/ch1/ep1_v.php?result={u}')
response.text.replace('\n', '')
```

II.3 Quick Response Code

Il s'agit de décoder un QR code pour lequel les 3 *pattern finders* sont manquants. Il faut donc les rajouter. J'ai utilisé numpy pour rajouter les 3 pattern finders. J'ai repéré les positions et taille des carrés en utilisant l'affichage interactif matplotlib (il faut mettre le mot clé *%matplotlib widget* apres l'import de matplotlib dans le jupyter notebook). Voici quelques informations utiles pour tracer les pattern finders :

- Dimensions d'un carré dans l'image du challenge : 9x9 pixel.
- Dimensions des espaces blancs dans lesquels il faut mettre les pattern finders : 9x9 carres.
- Dimensions d'un pattern finder dans un QR code : 7x7 pour le bordure exterieure, 3x3 pour le carré intérieur.

J'ai codé rapidement une petite fonction qui trace un pattern finder quand on lui donne le coin haut gauche.



(a) Image du challenge **(b)** Image apres ajout des 3 pattern finders

FIGURE 2

J'ai ensuite utilise la bibliotheque pyzbar pour decoder le QR code, ce qui conduit a un message avec le flag, qu'on recupere avec une expression reguliere.

```
import requests
from bs4 import BeautifulSoup
import numpy as np
from pyzbar.pyzbar import decode
import re
import base64
from io import BytesIO
from PIL import Image

s = requests.Session()

html_page = s.get('http://challenge01.root-me.org/programmation/ch7/')
soup = BeautifulSoup(html_page.text, 'html.parser')
img_tag = soup.find('img')
img_src = img_tag.get('src')

b64 = img_src.split(',')[1]
image_bytes = base64.b64decode(b64)
image_buffer = BytesIO(image_bytes)

image = Image.open(image_buffer)
img = np.array(image)

def draw_marker(left, top, m):
    """
    Draw the QR-code 7x7 square marker, with top left corner at (left, top).
```

```
    *******
    PARAMS:
    m: image, represented as 2d numpy array
    """
    square_size = 9
    m[left:left+square_size-1, top:top+7*square_size-1] = [0,0,0]
    m[left:left+7*square_size-1, top:top+square_size-1] = [0,0,0]
    m[left+6*square_size:left+7*square_size-1, top:top+7*square_size-1] =
        [0,0,0]
    m[left:left+7*square_size-1, top+6*square_size:top+7*square_size-1] =
        [0,0,0]
    m[left+2*square_size:left+5*square_size,
        top+2*square_size:top+5*square_size] = [0,0,0]
    return m

left, right, top, bottom = 18, 281, 18, 281

img = draw_marker(left, top, img)
img = draw_marker(left, 220, img)
img = draw_marker(220, top, img)

decoded_objects = decode(img)
key = re.search(r'The key is (.*)',
    decoded_objects[0].data.decode('utf-8')).groups(0)[0]

response = s.post('http://challenge01.root-me.org/programmation/ch7/',
    data={"metu":key})
response.text
```

II.4 Second degree polynomial solver

Il s'agit de résoudre 25 équations du second degré. Contrairement aux challenges précédents, la connexion se fait directement par TCP. Comme pour les précédents challenges, j'ai utilisé une expression régulière pour trouver les coefficients de l'équation. Ensuite, une petite fonction résout le polynôme et renvoie les sorties attendues. Il faut faire attention à bien mettre le retour à la ligne à la fin de chaque requête, sinon le serveur ne comprend pas qu'on attend une réponse. Dans mon code, il faut mettre des exposants dans l'expression régulière (x^2, x^1), mais LaTeX les affiche mal. Ma solution est bien plus concise que les deux proposées sur le site, il faudrait peut-être que je la publie.

```
import socket
from math import sqrt

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
s.connect(('challenge01.root-me.org', 52018))

def solve_quadratic(a, b, c):
    d = b**2 - 4*a*c
    if d < 0:
        return 'Not possible\n'
    elif d == 0:
        return f'x: {round(-b/2*a,3)}\n'
    else:
        x1, x2 = (-b + sqrt(d)) / (2 * a), (-b - sqrt(d)) / (2 * a)
        return f'x1: {round(x1,3)} ; x2: {round(x2,3)}\n'

for _ in range(25):
    m = re.search(r'Solve this equation please: (.*).x^2 ([-+] .*).x^1 ([-+]
        .*) = ([+-]?.*)', s.recv(1024).decode('utf-8'))
    a,b,c = eval((g:=m.groups(0))[0]), eval(g[1]), eval(g[2])-eval(g[3])
    s.sendall(bytes(solve_quadratic(a,b,c), 'utf-8'))

s.recv(1024)
```

II.5 Bash - System I

Il s'agit de se connecter à un serveur en SSH et d'afficher un mot de passe pour lequel on n'a pas les droits de lecture. Un script C, ch11.c, est disponible dans le répertoire home et liste le répertoire dans lequel se trouve le mot de passe. Une requête ls -la permet de voir qu'en exécutant le script, celui-ci a les mêmes droits d'accès que root. L'idée est de transformer le ls du script en cat pour afficher le mot de passe. Pour ce faire, on remarque qu'on a les droits d'écriture dans /tmp. On crée un script ls.sh qui exécute cat dans tmp. Ensuite, on réinitialise la variable $HOME avec pour unique valeur /tmp/ls. Ce faisant, lorsqu'on exécute ch11, le ls est remplacé par cat et le mot de passe est affiché.

II.6 Javascript - Authentification 2

Il s'agit de trouver l'identifiant et le mot de passe pour se connecter au site. Il suffit de sauvegarder la page et d'ouvrir le fichier login.js. Les identifiants sont alors visibles.

```
function connexion(){
    var username = prompt("Username :", "");
    var password = prompt("Password :", "");
    var TheLists = ["GOD:HIDDEN"];
    for (i = 0; i < TheLists.length; i++)
    {
        if (TheLists[i].indexOf(username) == 0)
        {
            var TheSplit = TheLists[i].split(":");
            var TheUsername = TheSplit[0];
            var ThePassword = TheSplit[1];
            if (username == TheUsername && password == ThePassword)
            {
                alert("Vous pouvez utiliser ce mot de passe pour valider ce
challenge (en majuscules) / You can use this password to validate this challenge
(uppercase)");
            }
        }
        else
        {
            alert("Nope, you're a naughty hacker.")
        }
    }
}
```

FIGURE 3 – Code javascript, voir la variable TheLists

II.7 Javacripts - Authentification

C'est le même principe que l'exercice précédent.

```
login.js                                                    login.js
/* <![CDATA[ */

function Login(){
        var pseudo=document.login.pseudo.value;
        var username=pseudo.toLowerCase();
        var password=document.login.password.value;
        password=password.toLowerCase();
        if (pseudo=="4dm1n" && password=="sh.org") {
            alert("Password accepté, vous pouvez valider le challenge avec ce mot
passe.\nYou an validate the challenge using this password.");
        } else {
            alert("Mauvais mot de passe / wrong password");
        }
}
/* ]]> */
```

FIGURE 4 – Code javascript, voir la variable TheLists