El mundo de las competiciones CTFs

Caliphal Hounds y Google CTF 2023

EL CONTENIDO DE ESTE TALLER SE IMPARTE ÚNICAMENTE CON

FINES EDUCATIVOS

HACED USO DE ESTE CONOCIMIENTO DE MANERA RESPONSABLE

QUIÉNES SOMOS

David Ramírez Acero

Carlos Freire Caballero

Alvaro Palacios Criado

Pablo Cáceres Gaitán

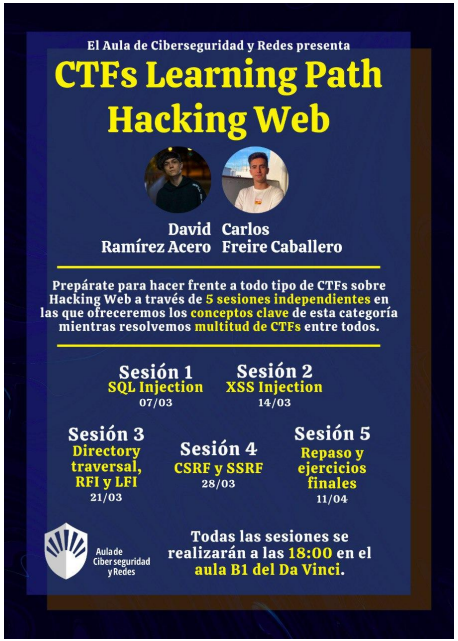
AULA

Aula de
Ciber seguridad
y Redes

¿QUÉ VAMOS A HACER?

1. Introducir nuestro equipo de CTFs: **Caliphal Hounds**.
2. Presentar los **departamentos**.
3. Introducción **Google CTF 2023**.
4. CTF de ejemplo 1: **Criptografía**.
5. CTF de ejemplo 2: **Esteganografía**.
6. CTF de ejemplo 3: **Miscelánea / PPC**.

CALIPHAL HOUNDS



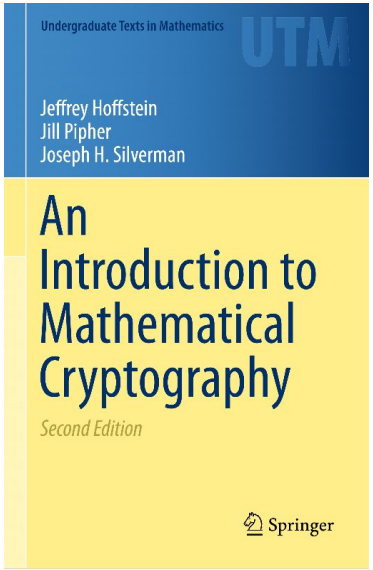
Country position	Name	Points
1	Flaggermeister	236,953
2	54e2952a4f049a4d8f11740c0ca852d6	182,684
3	base64	182,293
4	Watch4Hack	37,507
5	clock_break3rs	33,538
6	ISwearIGoogledIt	32,983
7	Ucha_CTF	32,782
8	ID-10-T	32,256
9	ICAI Cyber Team	32,046
10	LosCazones	31,655
11	Caliphal Hounds	31,612
12	Samfaina	28,083

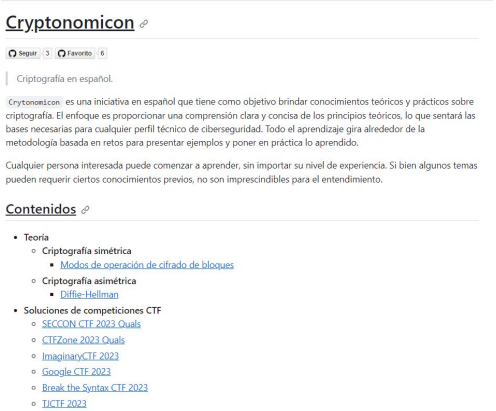
Place	Event
263	ASIS CTF Quals 2023
432	SECCON CTF 2023 Quals
352	SekaiCTF 2023
141	CTFZone 2023 Quals
131	ImaginaryCTF 2023
326	Crypto CTF 2023
171	Google Capture The Flag 2023
28	Break the Syntax CTF 2023
183	TJCTF 2023
60	Cryptoverse CTF 2023

11º de España en CTFTime

CRIPTOGRAFÍA

- Codificación
- Simétrica
- Asimétrica
- Post-cuántica
- Misc: PRNGs, SSS...





Cryptonomicon

Seguir 3 Favorito 6

> Criptografía en español.

`Crytonomicon` es una iniciativa en español que tiene como objetivo brindar conocimientos teóricos y prácticos sobre criptografía. El enfoque es proporcionar una comprensión clara y concisa de los principios teóricos, lo que sentará las bases necesarias para cualquier perfil técnico de ciberseguridad. Todo el aprendizaje gira alrededor de la metodología basada en retos para presentar ejemplos y poner en práctica lo aprendido.

Cualquier persona interesada puede comenzar a aprender, sin importar su nivel de experiencia. Si bien algunos temas pueden requerir ciertos conocimientos previos, no son imprescindibles para el entendimiento.

Contenidos

- Teoría
  - Criptografía simétrica
    - Modos de operación de cifrado de bloques
  - Criptografía asimétrica
    - Diffie-Hellman
- Soluciones de competiciones CTF
  - SECCON CTF 2023 Quals
  - CTFZone 2023 Quals
  - ImaginaryCTF 2023
  - Google CTF 2023
  - Break the Syntax CTF 2023
  - TJCTF 2023

HACKING WEB

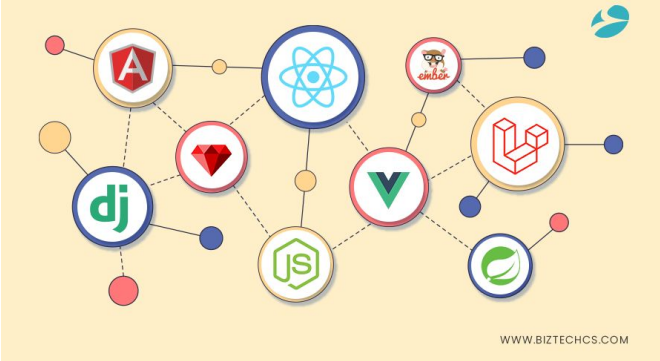
Vulnerabilidades:

- SQL/NoSQL injection
- XSS
- Directory Traversal
- CSRF y SSRF
- XXE
- ...

HACKING WEB

¿Con qué trabajamos?

- PortSwigger
- Burp Suite CE
- Docker
- Web frameworks, muchos web frameworks (Node.js, Ruby on Rails, Flask, ...)



REVERSING

¿Qué es Revering? → Ingeniería Inversa

TIPOS CTF

- Architecture
- OS
- Type:
  - Piece of assembly listing
  - Executable

HERRAMIENTAS

Windows

Linux

PLATAFORMAS

crackmes.one

Reversing.Kr

[Crackmes]

[Reversing.Kr]

[Reverse Engineering challenges]

[https://www.root-me.org]

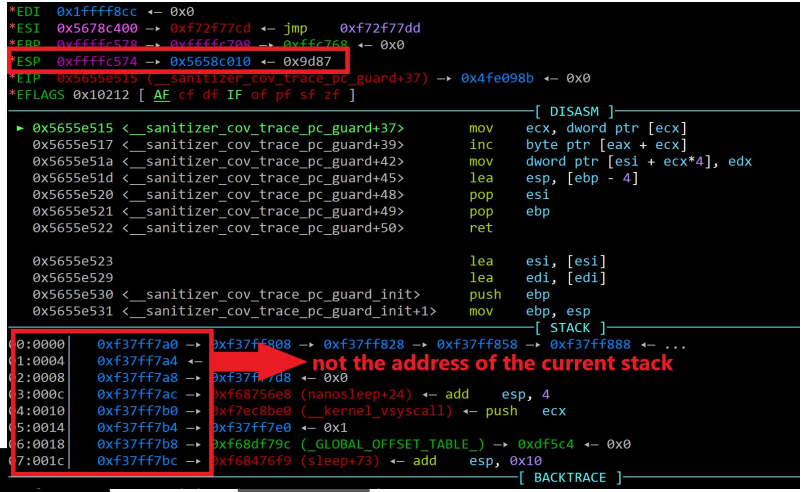
PWN - Explotación (de binarios)

- Herramientas
  - Las de Rev (ghidra)
  - Pwntools
  - Checksec
  - Gdb (gdb-pwnbg)

- Recursos
  - Cryptocat
  - pwn.college
  - pwnable.kr
  - pwnable.xyz

PWNABLE.KR

./ pwn.college



GOOGLE CTF 2023: ¿QUÉ ES?

GOOGLE CTF 2023: INTERÉS

Quals
1º: 12345$
2º: 5432$
3º: 3210$

8 equipos

Finals
1º: 7331$
2º: 3113$
3º: 1337$

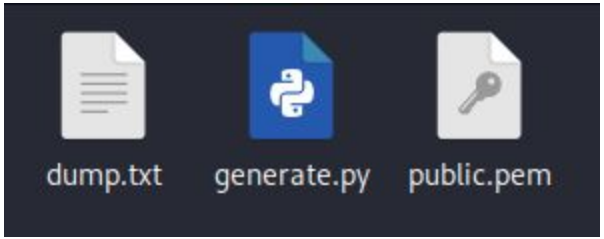
- misc
- crypto
- pwn
- reversing
- web
- sandbox

CTF DE EJEMPLO 1: CRIPTOGRAFÍA

LEAST COMMON GENOMINATOR?

Someone used this program to send me an encrypted message but I can't read it! It uses something called an LCG, do you know what it is? I dumped the first six consecutive values generated from it but what do I do with it?!

CTF DE EJEMPLO 1: CRIPTOGRAFÍA



dump.txt

6 números consecutivos

generate.py

Generación del cifrado

public.pem

Clave pública en formato PEM

CTF DE EJEMPLO 1: CRIPTOGRAFÍA

generate.py > ...

```python
from secret import config
from Crypto.PublicKey import RSA
from Crypto.Util.number import bytes_to_long, isPrime

class LCG:
    lcg_m = config.m
    lcg_c = config.c
    lcg_n = config.n

    def __init__(self, lcg_s):
        self.state = lcg_s

    def next(self):
        self.state = (self.state * self.lcg_m + self.lcg_c) % self.lcg_n
        return self.state

if __name__ == '__main__':

    assert 4096 % config.it == 0
    assert config.it == 8
    assert 4096 % config.bits == 0
    assert config.bits == 512

    # Find prime value of specified bits a specified amount of times
    seed =
    211286818345627549183608678726370412218029639873054513839005340650674982169404937862395980568550063504804783328450267566224937880641772833325018028629959635
    lcg = LCG(seed)
```

CTF DE EJEMPLO 1: CRIPTOGRAFÍA

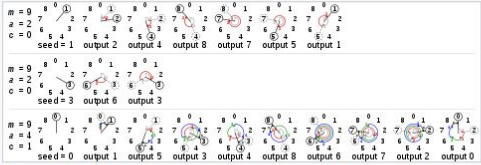
Linear congruential generator

17 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

A **linear congruential generator** (**LCG**) is an algorithm that yields a sequence of pseudo-randomized numbers calculated with a discontinuous piecewise linear equation. The method represents one of the oldest and best-known pseudorandom number generator algorithms. The theory behind them is relatively easy to understand, and they are easily implemented and fast, especially on computer hardware which can provide modular arithmetic by storage-bit truncation.



Two modulo-9 LCGs show how different parameters lead to different cycle lengths. Each row shows the state evolving until it repeats. The top row shows a generator with m = 9, a = 2, c = 0, and a seed of 1, which produces a cycle of length 6. The second row is the same generator with a seed of 3, which produces a cycle of length 2. Using a = 4 and c = 1 (bottom row) gives a cycle length of 9 with any seed in [0, 8].

The generator is defined by the recurrence relation:

$$X_{n+1} = (aX_n + c) \mod m$$

where X is the sequence of pseudo-random values, and

m, $0 < m$ — the "modulus"
a, $0 < a < m$ — the "multiplier"
c, $0 \le c < m$ — the "increment"
X_0, $0 \le X_0 < m$ — the "seed" or "start value"

are integer constants that specify the generator. If c = 0, the generator is often called a **multiplicative congruential generator** (MCG), or Lehmer RNG. If $c \neq 0$, the method is called a **mixed congruential generator**.[1]:4

When $c \neq 0$, a mathematician would call the recurrence an affine transformation, not a linear one, but the misnomer is well-established in computer science.[2]:1

$$X_{i+1} = (mX_i + c) \mod n$$

CTF DE EJEMPLO 1: CRIPTOGRAFÍA

Tenemos:

- X_0 = seed (en el código)
- $X_1, X_2, X_3, X_4, X_5, X_6$ (en dump.txt)

¿Podemos conseguir m, c y n?

$$X_{i+1} = (mX_i + c) \bmod n$$

CTF DE EJEMPLO 1: CRIPTOGRAFÍA

AVISO: SOLO VAMOS A ROMPER EL LCG

Ignoramos teoría RSA, PRNGS... Si queréis detalles:

Google CTF 2023

Seguir 3 Favorito 6

CTF PRNG clave pública aritmética modular

> 23/06/2023 20:00 CEST - 25/06/2023 20:00 CEST

Todo el código desarrollado se puede consultar en la carpeta de scripts.

Least Common Genominator?

> 352/676 soluciones | 50 puntos

Enunciado

```
Someone used this program to send me an encrypted message but I can't read it! It uses something called an LCG, do you know wha
```

Archivos

CTF DE EJEMPLO 1: CRIPTOGRAFÍA

$$s_1 \equiv (m \cdot seed + c) \bmod n$$
$$s_2 \equiv (m \cdot s_1 + c) \bmod n$$
$$s_3 \equiv (m \cdot s_2 + c) \bmod n$$
$$s_4 \equiv (m \cdot s_3 + c) \bmod n$$
$$s_5 \equiv (m \cdot s_4 + c) \bmod n$$
$$s_6 \equiv (m \cdot s_5 + c) \bmod n$$

$\longrightarrow$

$$s_2 - s_1 \equiv (s_1 - seed) \cdot m \bmod n$$
$$s_3 - s_2 \equiv (s_2 - s_1) \cdot m \bmod n$$
$$s_4 - s_3 \equiv (s_3 - s_2) \cdot m \bmod n$$
$$s_5 - s_4 \equiv (s_4 - s_3) \cdot m \bmod n$$
$$s_6 - s_5 \equiv (s_5 - s_4) \cdot m \bmod n$$

$\longrightarrow$

$$m \equiv (s_2 - s_1) \cdot (s_1 - seed)^{-1} \bmod n$$
$$m \equiv (s_3 - s_2) \cdot (s_2 - s_1)^{-1} \bmod n$$
$$m \equiv (s_4 - s_3) \cdot (s_3 - s_2)^{-1} \bmod n$$
$$m \equiv (s_5 - s_4) \cdot (s_4 - s_3)^{-1} \bmod n$$
$$m \equiv (s_6 - s_5) \cdot (s_5 - s_4)^{-1} \bmod n$$

$$(s_2 - s_1) \cdot (s_1 - seed)^{-1} \equiv (s_3 - s_2) \cdot (s_2 - s_1)^{-1} \bmod n$$
$$(s_2 - s_1)^2 \equiv (s_3 - s_2) \cdot (s_1 - seed) \bmod n$$
$$(s_2 - s_1)^2 - (s_3 - s_2) \cdot (s_1 - seed) \equiv 0 \bmod n$$

CTF DE EJEMPLO 1: CRIPTOGRAFÍA

Conociendo n...

$$m \equiv (s_2 - s_1) \cdot (s_1 - seed)^{-1} \bmod n$$

$$s_1 \equiv (m \cdot seed + c) \bmod n$$

$$c \equiv s_1 - m \cdot seed \bmod n$$

CTF DE EJEMPLO 1: CRIPTOGRAFÍA

```
seed = 211286818345627549183608678726370412218029639873054513839005340650674982169404937862395980568550063504804783328450267566
s1 = 216677167559518406933910736590837715770116448582098140999392527951219912341837403427546559000484813594667145408422073164509
s2 = 672927295046762545629845467821961309046725482467931899305229458757015342493526736497182727713752192920278362155342195853376
s3 = 223039690330235292148470412270553940320105049016464910218279805992634309651115828886730161464847151672305209276131210511777
s4 = 457884778773614375685082340716851911217526009260147681053983079265656874713660425014685811141870505413826619334816923971040
s5 = 757833297947908654663746903694848255115124009980381223594999714789287109798229301725647518950444795514739940579187539545085
s6 = 255042044327038100300787352076304283749324419761666666776839714611058930160211988483660541866446355086539902693484828908429

t0 = s1 - seed
t1 = s2 - s1
t2 = s3 - s2
t3 = s4 - s3
t4 = s5 - s4
t5 = s6 - s5

multiple1 = (t0 * t2) - pow(t1, 2)
multiple2 = (t1 * t3) - pow(t2, 2)
multiple3 = (t2 * t4) - pow(t3, 2)
multiple4 = (t3 * t5) - pow(t4, 2)
multiple5 = (t3 * t0) - (t1 * t2)
multiple6 = (t4 * t0) - (t1 * t3)
multiple7 = (t5 * t0) - (t1 * t4)
multiple8 = (t4 * t1) - (t2 * t3)
multiple9 = (t5 * t1) - (t2 * t4)
multiple10 = (t5 * t2) - (t3 * t4)

n2 = GCD(multiple1, multiple2, multiple3, multiple4, multiple5, multiple6, multiple7, multiple8, multiple9, multiple10)

m = (s2 - s1) * pow(s1 - seed, -1, n2) % n2

c = s1 - seed*m % n2

assert s1 == (seed * m + c) % n2
```

CTF{C0nGr@tz_RiV35t_5h4MiR_nD_Ad13MaN_W0ulD_b_h@pPy}

CTF DE EJEMPLO 2: ESTEGANOGRAFÍA

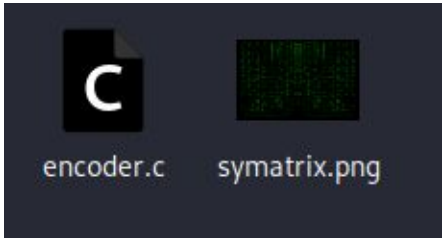
SYMATRIX

The CIA has been tracking a group of hackers who communicate using PNG files embedded with a custom steganography algorithm.

An insider spy was able to obtain the encoder, but it is not the original code.

You have been tasked with reversing the encoder file and creating a decoder as soon as possible in order to read the most recent PNG file they have sent.

CTF DE EJEMPLO 2: ESTEGANOGRAFÍA



encoder.c

Código que añade el mensaje

symatrix.png

Imágen con mensaje oculto

CTF DE EJEMPLO 2: ESTEGANOGRAFÍA

¿Qué veis? ¿A qué os recuerda?

CTF DE EJEMPLO 2: ESTEGANOGRAFÍA

encoder.c

aba60aa2e9c806187f88279742c2ced243dd73b142c5c5bac1327956975e4d3add04afad77cfd823dd4f00847f6334b294ab058308639f8cb52897e8f1be769e > encode

```c
17  */
18
19  #ifndef PY_SSIZE_T_CLEAN
20  #define PY_SSIZE_T_CLEAN
21  #endif /* PY_SSIZE_T_CLEAN */
22  #include "Python.h"
23  #ifndef Py_PYTHON_H
24      #error Python headers needed to compile C extensions, please install development version of Python.
25  #elif PY_VERSION_HEX < 0x02060000 || (0x03000000 <= PY_VERSION_HEX && PY_VERSION_HEX < 0x03030000)
26      #error Cython requires Python 2.6+ or Python 3.3+.
27  #else
28  #define CYTHON_ABI "0_29_33"
29  #define CYTHON_HEX_VERSION 0x001D21F0
30  #define CYTHON_FUTURE_DIVISION 0
31  #include <stddef.h>
32  #ifndef offsetof
33    #define offsetof(type, member) ( (size_t) & ((type*)0) -> member )
34  #endif
35  #if !defined(WIN32) && !defined(MS_WINDOWS)
36    #ifndef __stdcall
37      #define __stdcall
38    #endif
39    #ifndef __cdecl
40      #define __cdecl
41    #endif
42    #ifndef __fastcall
43      #define __fastcall
44    #endif
45  #endif
46  #ifndef DL_IMPORT
```

Código en C generado por Cython

CTF DE EJEMPLO 2: ESTEGANOGRAFÍA

```
1637  /* "encoder.py":7
1638 * def hexstr_to_binstr(hexstr):
1639 *     n = int(hexstr, 16)
1640 *     bstr = ''             # <<<<<<<<<<<<<<
1641 *     while n > 0:
1642 *         bstr = str(n % 2) + bstr
1643 */
1644  __Pyx_INCREF(__pyx_kp_s_);
1645  __pyx_v_bstr = __pyx_kp_s_;
1646
1647  /* "encoder.py":8
1648 *     n = int(hexstr, 16)
1649 *     bstr = ''
1650 *     while n > 0:             # <<<<<<<<<<<<<<
1651 *         bstr = str(n % 2) + bstr
1652 *         n = n >> 1
1653 */
1654  while (1) {
1655    __pyx_t_2 = PyObject_RichCompare(__pyx_v_n, __pyx_int_0, Py_GT); __Pyx_XGOTREF(__pyx_t_2); if (unlikely(!__pyx_t_2)) __PYX_ERR
      (0, 8, __pyx_L1_error)
1656    __pyx_t_3 = __Pyx_PyObject_IsTrue(__pyx_t_2); if (unlikely(__pyx_t_3 < 0)) __PYX_ERR(0, 8, __pyx_L1_error)
1657    __Pyx_DECREF(__pyx_t_2); __pyx_t_2 = 0;
1658    if (!__pyx_t_3) break;
```

CTF DE EJEMPLO 2: ESTEGANOGRAFÍA

encoder.c | encoder.py

encoder.py > hexstr_to_binstr

```python
from PIL import Image
from random import randint
import binascii


def hexstr_to_binstr(hexstr):
    n = int(hexstr, 16)
    bstr = ''
    while n > 0:
        bstr = str(n % 2) + bstr
        n = n >> 1
    if len(bstr) % 8 != 0:
        bstr = '0' + bstr
    return bstr

def pixel_bit(b):
    return tuple((0, 1, b))

def embed(t1, t2):
    return tuple((t1[0] + t2[0], t1[1] + t2[1], t1[2] + t2[2]))

def full_pixel(pixel):
    return pixel[1] == 255 or pixel[2] == 255



print("Embedding file...")

bin_data = open("./flag.txt", 'rb').read()
data_to_hide = binascii.hexlify(bin_data).decode('utf-8')
```

```python
base_image = Image.open("./original.png")

x_len, y_len = base_image.size
nx_len = x_len * 2
new_image = Image.new("RGB", (nx_len, y_len))

base_matrix = base_image.load()
new_matrix = new_image.load()

binary_string = hexstr_to_binstr(data_to_hide)
remaining_bits = len(binary_string)
nx_len = nx_len - 1
next_position = 0


for i in range(0, y_len):
    for j in range(0, x_len):
        pixel = new_matrix[j, i] = base_matrix[j, i]
        if remaining_bits > 0 and next_position <= 0 and not full_pixel(pixel):
            new_matrix[nx_len - j, i] = embed(pixel_bit(int(binary_string[0])),pixel)
            next_position = randint(1, 17)
            binary_string = binary_string[1:]
            remaining_bits -= 1
        else:
            new_matrix[nx_len - j, i] = pixel
            next_position -= 1

new_image.save("./symatrix.png")
new_image.close()
base_image.close()
print("Work done!")
exit(1)
```

Veamos el código

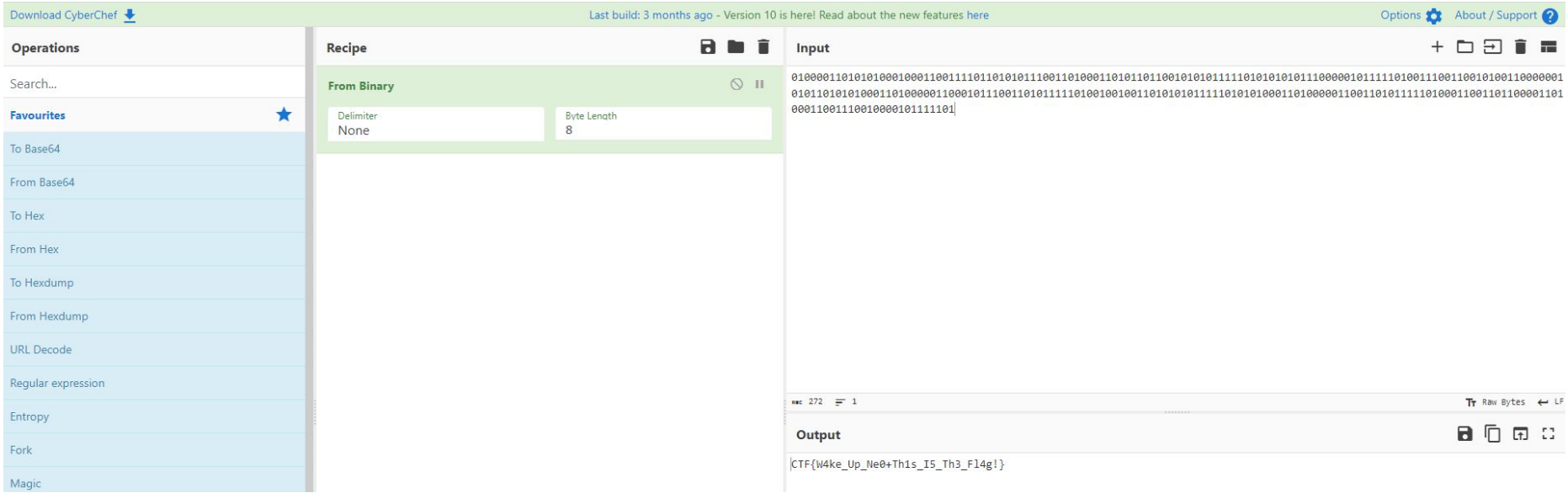
CTF DE EJEMPLO 2: ESTEGANOGRAFÍA

solver.py

solver.py > ...

```python
from PIL import Image

image = Image.open("./symatrix.png")

nx_len, y_len = image.size
matrix = image.load()
x_len = int(nx_len / 2)

nx_len -= 1

m = ""
for i in range(0, y_len):
    for j in range(0, x_len):
        pixel = matrix[j, i]

        if pixel != matrix[nx_len - j, i]:
            m += str(matrix[nx_len - j, i][2])

print(m)
```

CTF DE EJEMPLO 2: ESTEGANOGRAFÍA



CTF{W4ke_Up_Ne0+Th1s_I5_Th3_Fl4g!}

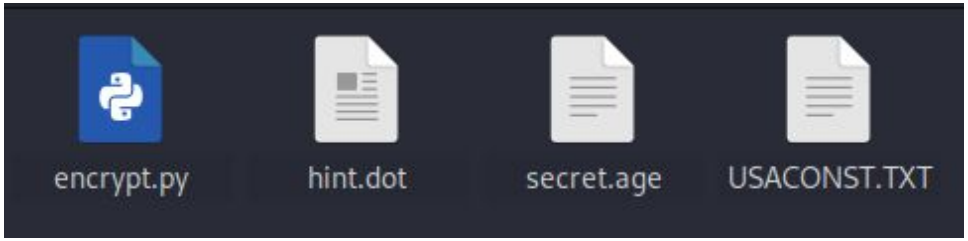
CTF DE EJEMPLO 3: MISCELÁNEA / PPC

NPC

A friend handed me this map and told me that it will lead me to the flag.

It is confusing me and I don't know how to read it can you help me out?

CTF DE EJEMPLO 3: MISCELÁNEA / PPC



encrypt.py

Código que cifra el mensaje

hint.dot

“Mapa” para encontrar la flag

secret.age

Mensaje a descifrar

USACONST.TXT

Constitución de Estados Unidos

CTF DE EJEMPLO 3: MISCELÁNEA / PPC

Sabemos:

- Una contraseña formada por 28 caracteres (hay 28 nodos en el grafo).
- La contraseña está formada por n palabras de la constitución de EEUU.

Veamos la estructura de la pista

CTF DE EJEMPLO 3: MISCELÁNEA / PPC

```python
def generate_hint(password):
  random = secrets.SystemRandom()
  id_gen = IdGen()
  graph = Graph([],[])
  for letter in password:
    graph.nodes.append(Node(letter, id_gen.generate_id()))
  for a, b in zip(graph.nodes, graph.nodes[1:]):
    graph.edges.append(Edge(a, b))
  for _ in range(int(len(password)**1.3)):
    a, b = random.sample(graph.nodes, 2)
    graph.edges.append(Edge(a, b))
  random.shuffle(graph.nodes)
  random.shuffle(graph.edges)
  for edge in graph.edges:
    if random.random() % 2:
      edge.a, edge.b = edge.b, edge.a
  return graph
```

```
graph {
    1051081353 [label=a];
    66849241 [label=a];
    53342583 [label=n];
    213493562 [label=d];
    4385267 [label=i];
    261138725 [label=o];
    51574206 [label=t];
    565468867 [label=e];
    647082638 [label=r];
    177014844 [label=d];
    894978618 [label=e];
    948544779 [label=n];
    572570465 [label=n];
    582531406 [label=r];
    264939475 [label=a];
    415170621 [label=s];
    532012257 [label=t];
    151901859 [label=v];
    346347468 [label=g];
    148496047 [label=g];
    125615053 [label=s];
    723039811 [label=e];
```

```
    53342583 -- 565468867;
    582531406 -- 76544105;
    125615053 -- 120330115;
    264939475 -- 572570465;
    53342583 -- 565468867;
    532012257 -- 264939475;
    346347468 -- 532012257;
    125615053 -- 582531406;
    177014844 -- 120330115;
```

$28^{1.33}$ = 84 aristas erroneas + 28 originales=112 totales | 50% revertir

CTF DE EJEMPLO 3: MISCELÁNEA / PPC

¿Ideas?

CTF DE EJEMPLO 3: MISCELÁNEA / PPC

Con los nodos, podemos descartar:

- Cualquier palabra que tenga una letra que **no está en el grafo**. (292 palabras restantes)
- Cualquier palabra que tenga **un número mayor de una letra que el que hay en el grafo**. (259 palabras restantes)

CTF DE EJEMPLO 3: MISCELÁNEA / PPC

Con las aristas, podemos descartar:

- Cualquier palabra que tenga una sucesión de **2 letras no presente** en el grafo. (86 palabras restantes)

¿Y ahora qué?

CTF DE EJEMPLO 3: MISCELÁNEA / PPC

Hay unas letras que solo se **repiten una vez**. Se coge la letra que **menos palabras candidatas** tiene.

En este caso, la “c” solo se repite una vez en toda la contraseña, y solo **7 palabras** tienen “c”:

['chosen', 'consist', 'consent', 'charged', 'encreased', 'congress', 'created']

Una palabra de estas **debe** estar en la contraseña.

CTF DE EJEMPLO 3: MISCELÁNEA / PPC

Problema del **número n de palabras** de la contraseña.

En base a prueba y error, con 28 caracteres y las palabras candidatas, tenía que tener entre **4 y 5 palabras**.

Tomar **86 palabras candidatas de 5 en 5** era una locura computacionalmente.

Tomarlas de **4 en 4** y añadir las **palabras con "c"** no.

CTF DE EJEMPLO 3: MISCELÁNEA / PPC

Combinando las palabras solo existen **6 posibles combinaciones**.

```
solutions = [["consist", "and", "whatever", "sign", "danger"],
["consist", "whatever", "grand", "an", "signed"],
["chosen", "standard", "giving", "was", "enter"],
["chosen", "standard", "given", "water", "sign"],
["congress", "standard", "nine", "give", "what"],
["consent", "was", "derived", "granting", "has"]]
```

¿Pero y el orden? 5! * 6 = 720 combinaciones

CTF DE EJEMPLO 3: MISCELÁNEA / PPC

```
['consistandwhateversigndanger', 'consistandwhateverdangersign', 'consistandsignwhateverdanger', 'consistandsigndangerwhatever', 'consistanddangerwhateversign',
'consistanddangersignwhatever', 'consistwhateverandsigndanger', 'consistwhateveranddangersign', 'consistwhateversignanddanger', 'consistwhateversigndangerand',
'consistwhateverdangerandsign', 'consistwhateverdangersignand', 'consistsignandwhateverdanger', 'consistsignanddangerwhatever', 'consistsignwhateveranddanger',
'consistsignwhateverdangerand', 'consistsigndangerandwhatever', 'consistsigndangerwhateverand', 'consistdangerandwhateversign', 'consistdangerandsignwhatever',
'consistdangerwhateverandsign', 'consistdangerwhateversignand', 'consistdangersignandwhatever', 'consistdangersignwhateverand', 'andconsistwhateversigndanger',
'andconsistwhateverdangersign', 'andconsistsignwhateverdanger', 'andconsistsigndangerwhatever', 'andconsistdangerwhateversign', 'andconsistdangersignwhatever',
'andwhateverconsistsigndanger', 'andwhateverconsistdangersign', 'andwhateversignconsistdanger', 'andwhateversigndangerconsist', 'andwhateverdangerconsistsign',
'andwhateverdangersignconsist', 'andsignconsistwhateverdanger', 'andsignconsistdangerwhatever', 'andsignwhateverconsistdanger', 'andsignwhateverdangerconsist',
'andsigndangerconsistwhatever', 'andsigndangerwhateverconsist', 'anddangerconsistwhateversign', 'anddangerconsistsignwhatever', 'anddangerwhateverconsistsign',
'anddangerwhateversignconsist', 'anddangersignconsistwhatever', 'anddangersignwhateverconsist', 'whateverconsistandsigndanger', 'whateverconsistanddangersign',
'whateverconsistsignanddanger', 'whateverconsistsigndangerand', 'whateverconsistdangerandsign', 'whateverconsistdangersignand', 'whateverandconsistsigndanger',
'whateverandconsistdangersign', 'whateverandsignconsistdanger', 'whateverandsigndangerconsist', 'whateveranddangerconsistsign', 'whateveranddangersignconsist',
'whateversignconsistanddanger', 'whateversignconsistdangerand', 'whateversignandconsistdanger', 'whateversignanddangerconsist', 'whateversigndangerconsistand',
'whateversigndangerandconsist', 'whateverdangerconsistandsign', 'whateverdangerconsistsignand', 'whateverdangerandconsistsign', 'whateverdangerandsignconsist',
'whateverdangersignconsistand', 'whateverdangersignandconsist', 'signconsistandwhateverdanger', 'signconsistanddangerwhatever', 'signconsistwhateveranddanger',
'signconsistwhateverdangerand', 'signconsistdangerandwhatever', 'signconsistdangerwhateverand', 'signandconsistwhateverdanger', 'signandconsistdangerwhatever',
'signandwhateverconsistdanger', 'signandwhateverdangerconsist', 'signanddangerconsistwhatever', 'signanddangerwhateverconsist', 'signwhateverconsistanddanger',
'signwhateverconsistdangerand', 'signwhateverandconsistdanger', 'signwhateveranddangerconsist', 'signwhateverdangerconsistand', 'signwhateverdangerandconsist',
'signdangerconsistandwhatever', 'signdangerconsistwhateverand', 'signdangerandconsistwhatever', 'signdangerandwhateverconsist', 'signdangerwhateverconsistand',
'signdangerwhateverandconsist', 'dangerconsistandwhateversign', 'dangerconsistandsignwhatever', 'dangerconsistwhateverandsign', 'dangerconsistwhateversignand',
'dangerconsistsignandwhatever', 'dangerconsistsignwhateverand', 'dangerandconsistwhateversign', 'dangerandconsistsignwhatever', 'dangerandwhateverconsistsign',
'dangerandwhateversignconsist', 'dangerandsignconsistwhatever', 'dangerandsignwhateverconsist', 'dangerwhateverconsistandsign', 'dangerwhateverconsistsignand',
'dangerwhateverandconsistsign', 'dangerwhateverandsignconsist', 'dangerwhateversignconsistand', 'dangerwhateversignandconsist', 'dangersignconsistandwhatever',
'dangersignconsistwhateverand', 'dangersignandconsistwhatever', 'dangersignandwhateverconsist', 'dangersignwhateverconsistand', 'dangersignwhateverandconsist',
'consistwhatevergrandansigned', 'consistwhatevergrandsignedan', 'consistwhateverangrandsigned', 'consistwhateveransignedgrand', 'consistwhateversignedgrandan',
```

CTF{S3vEn_bRldg35_0f_K0eNlg5BeRg}

REFERENCIAS

- CTFTime: https://ctftime.org/
- Google CTF 2023: https://capturetheflag.withgoogle.com/
- Cryptonomicon: https://github.com/Daysapro/cryptonomicon
- HackTricks: HackTricks - HackTricks

Muchas gracias

Caliphal Hounds