


```

1 import random
2 from secrets import randbelow
3
4 seed = randbelow(2**256)
5 print(f"3NCRIPT4D0R_D3_M3NS4G3NS_EiTS versao 2.1.{seed}")
6 random.seed(seed)
7
8 def inicializaM(lin, col, arr):
9     mat = [[0 for _ in range(col)] for _ in range(lin)]
10
11     for i in range(lin):
12         random.shuffle(arr)
13         mat[i] = arr[:]
14
15     return mat
16
17 def genA(lin, mat):
18     arr = [0] * lin
19
20     for i in range(lin):
21         arr[i] = random.choice(mat[i])
22
23     return arr
24
25 def inverte(numero):
26     return int(str(numero)[::-1])
27
28 def cript(msg, arr):
29     msg = int.from_bytes(msg, 'big')
30
31     for numero in arr:
32         msg = (msg << numero) + 1
33         msg = inverte(msg)
34
35     return msg
36
37 lin = random.choice(range(20, 30))
38 col = random.choice(range(10, 20))
39 a = random.choices(range(1, 10), k=col)
40
41 M = inicializaM(lin, col, a)
42
43 membros = ['beedee', 'reticencias', 'Fnknda', 'SirMonteiro']
44 # O seguinte array "msgs" não corresponde ao array "msgs" que está rodando no servidor
45 # (obviamente pra não deixar a flag tão fácil de se conseguir..)
46 msgs = [ b'eits{flag_falsa}', b'eits{essa_nao_eh_a_flag}'
47         , b'eits{essa_tambem_nao...}', b'eits{muito_menos_essa}']
48
49 for membro, msg in zip(membros, msgs):
50     A = genA(lin, M)
51     enc = cript(msg, A)
52     print(f'[{membro}]: {enc}')

```

Disponível em:

<https://ctf.intheshell.page/challenges#randomico-22>

Objetivo:

O objetivo deste desafio é expor as vulnerabilidades em sistemas de criptografia expõem as **seeds de criptografia**. Com isso, o desafio busca demonstrar como a previsibilidade de elementos aleatórios pode permitir a recuperação de dados originalmente protegidos.

Resolução:

Análise do Código:

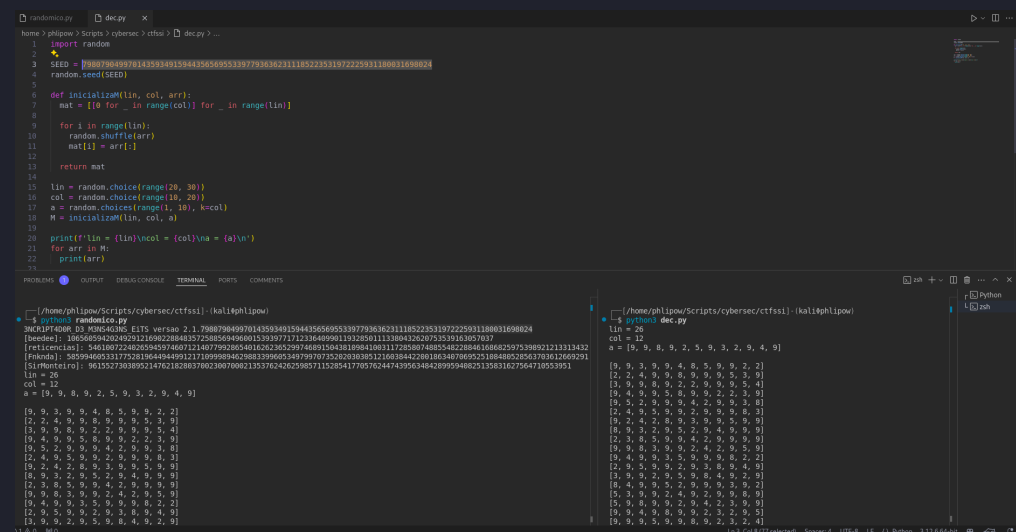
Ao analisar o código, foi possível identificar que o processo de criptografia dependia de variáveis geradas pseudo-aleatoriamente a partir de uma seed. Essa seed estava exposta no programa, sendo impressa em:

```
print(f"3NCR1PT4D0R_D3_M3NS4G3NS_EiTS versao 2.1.{seed}")
```

Com a seed visível, tornou-se viável reconstruir a sequência de variáveis aleatórias, o que seria essencial para descriptografar as mensagens

Reconstrução de Variáveis Iniciais:

O próximo passo foi garantir que as variáveis iniciais aleatórias `lin`, `col`, `a` e `M` estavam sendo geradas corretamente com base na seed fornecida. Para isso, editei o código do arquivo `randomico.py`, adicionando prints que exibiram as essas variáveis. Em seguida, copieei a lógica de inicialização delas para um novo script, `dec.py`. Utilizando a mesma `seed` fornecida por `randomico.py`, gerei as variáveis em `dec.py` para assegurar que os valores estavam sendo gerados corretamente.



```
1 import random
2
3 seed = 180879849970143594915944356565533779363623118522351972225931180831698024
4 random.seed(SEED)
5
6 def inicializa(lin, col, arr):
7     mat = [[0 for _ in range(col)] for _ in range(lin)]
8
9     for i in range(lin):
10         random.shuffle(arr)
11         mat[i] = arr[i]
12
13     return mat
14
15 lin = random.choice(range(20, 30))
16 col = random.choice(range(10, 20))
17 a = random.choice(range(1, 10), k=col)
18 M = inicializa(lin, col, a)
19
20 print(f"lin = {lin}\ncol = {col}\na = {a}\n")
21 for arr in M:
22     print(arr)
23
24
```

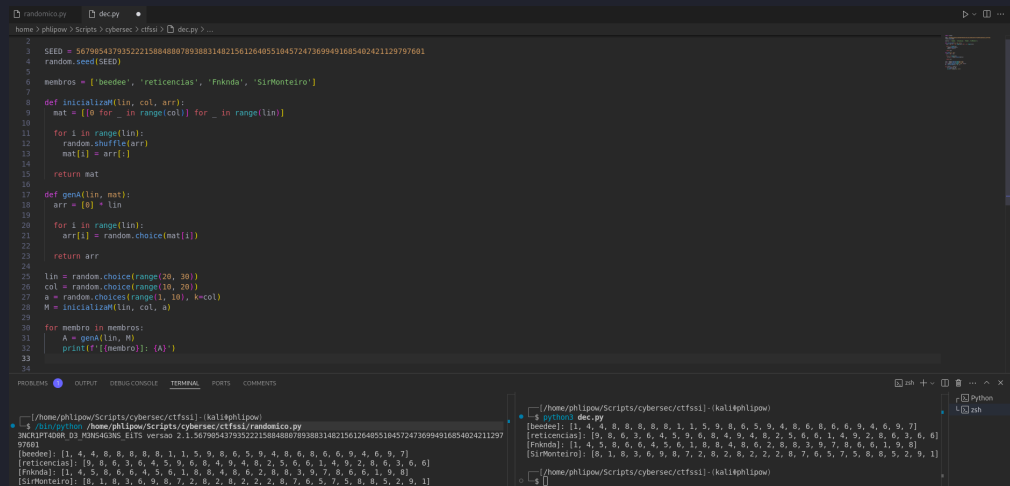
```
1 # python3 dec.py
2
3 lin = 26
4 col = 12
5 a = [9, 9, 8, 9, 2, 5, 9, 3, 2, 9, 4, 9]
6
7 [9, 9, 3, 9, 9, 4, 8, 5, 9, 9, 2, 2]
8 [2, 2, 4, 9, 9, 8, 9, 9, 9, 5, 3, 9]
9 [3, 9, 9, 8, 9, 2, 2, 9, 9, 9, 5, 4]
10 [9, 4, 9, 9, 5, 8, 9, 9, 2, 2, 3, 9]
11 [9, 2, 2, 9, 9, 9, 4, 2, 9, 9, 3, 8]
12 [2, 4, 9, 5, 9, 9, 2, 9, 9, 8, 3]
13 [9, 2, 4, 8, 9, 3, 9, 9, 9, 9, 9]
14 [8, 9, 3, 2, 9, 5, 2, 9, 9, 9, 9]
15 [2, 3, 8, 5, 9, 9, 4, 2, 9, 9, 9]
16 [9, 4, 9, 9, 2, 4, 1, 2, 9, 9, 8]
17 [5, 9, 8, 9, 9, 2, 9, 4, 2, 3, 9]
18 [9, 4, 9, 8, 9, 9, 2, 3, 2, 9, 3]
19 [9, 9, 9, 5, 9, 9, 8, 9, 2, 3, 2, 4]
```

obs.: Note que a constante SEED foi copiada do output de `randomico.py`, a esquerda.

Reconstrução dos As:

Com as variáveis iniciais devidamente geradas, o próximo passo foi reconstruir os arrays `As`, que eram fundamentais para a descriptografia das mensagens. Os array `As` são gerados aleatoriamente utilizando essas variáveis iniciais. Para isso, implementei a mesma lógica de construção do array `A` no script `dec.py`, replicando o processo exato do programa original

utilizava. Assim, consegui recriar o array A correspondente a cada mensagem. Para assegurar que estavam sendo gerados corretamente, editei o código `randomico.py` para que printasse os As ao invés das mensagens e os comparei com o output de `dec.py`.



```
1 randomico.py 2 dec.py
3
4 SEED = 56798543793222158848807893883148215612648551845724736949168540242112979601
5 random.seed(SEED)
6
7 membros = ['beedee', 'reticencias', 'frknda', 'SiMonteiro']
8
9 def inicializaMat(lin, col, arr):
10     mat = [[0 for _ in range(col)] for _ in range(lin)]
11     for i in range(lin):
12         random.shuffle(arr)
13         mat[i] = arr[i]
14     return mat
15
16 def gera(lin, mat):
17     arr = [0] * lin
18     for i in range(lin):
19         arr[i] = random.choice(mat[i])
20     return arr
21
22 lin = random.choice(range(20, 30))
23 col = random.choice(range(10, 20))
24 a = random.choice(range(1, 10), k=col)
25 M = inicializaMat(lin, col, a)
26
27 for membro in membros:
28     A = gera(lin, M)
29     print(f'{membro}: {A}')
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
1 /home/philipow/Scripts/cybersec/ctfss1-(kalishphilow)
2 $ python3 randomico.py
3 randomico.py version 1.1-56798543793222158848807893883148215612648551845724736949168540242112979601
4 [beedee]: [1, 4, 4, 8, 8, 8, 8, 8, 1, 1, 5, 9, 8, 6, 5, 9, 4, 8, 6, 6, 6, 9, 4, 6, 9, 7]
5 [reticencias]: [9, 8, 6, 3, 6, 4, 5, 9, 6, 6, 4, 9, 4, 8, 2, 5, 6, 6, 1, 4, 9, 2, 8, 6, 3, 6, 6]
6 [frknda]: [1, 4, 5, 8, 6, 6, 4, 5, 6, 1, 8, 8, 4, 8, 6, 2, 8, 8, 3, 9, 7, 8, 6, 6, 1, 9, 8]
7 [SiMonteiro]: [8, 1, 8, 3, 6, 9, 6, 7, 2, 6, 2, 8, 2, 2, 2, 8, 7, 6, 5, 7, 5, 8, 8, 5, 2, 9, 1]
8
```

Descriptografando as Mensagens:

Após recriar corretamente o array A, foi possível desenvolver um algoritmo de descriptografia que segue a lógica inversa da criptografia. Para entender essa lógica, primeiro precisamos analisar o funcionamento da função `cript`, responsável por criptografar as mensagens:

Lógica de Criptografia:

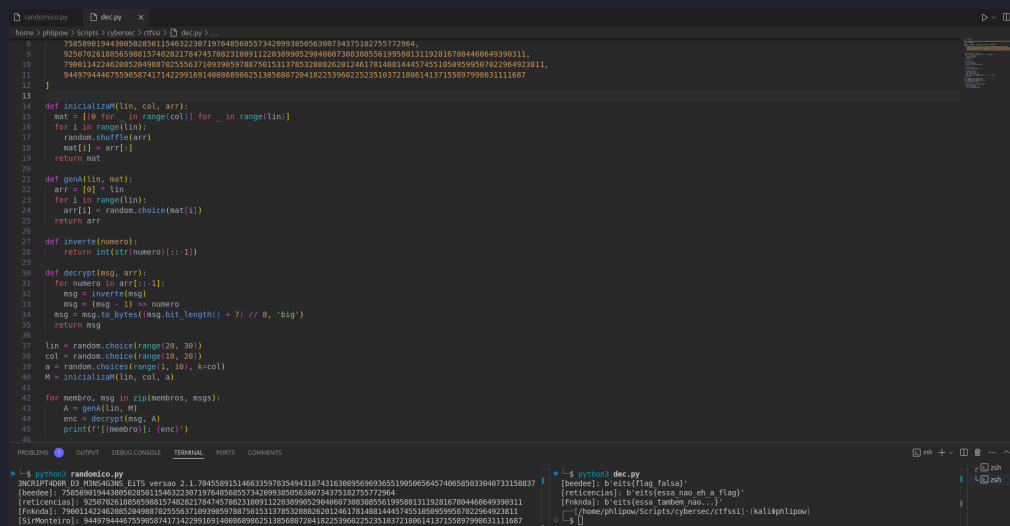
1. A mensagem `msg` é convertida para um inteiro usando `int.from_bytes(msg, 'big')`.
2. Para cada número em `arr`:
 - 2.1. A mensagem é deslocada à esquerda (`msg << numero`), aumentando seu valor conforme o número de bits especificado por `numero`.
 - 2.2. O valor resultante tem `1` adicionado ao final.
 - 2.3. A função `inverte` é aplicada para alterar a ordem dos bits.
3. O valor final resultante é a mensagem criptografada.

Com essa lógica compreendida, foi possível desenvolver a função `decrypt`, que executa o processo inverso para restaurar a mensagem original:

analisar o funcionamento da função **cript**, responsável por criptografar as mensagens:

Lógica de Descriptografia:

1. A função percorre o array **arr** ao contrário (**arr[::-1]**), revertendo cada etapa da criptografia.
2. Para cada número em **arr**:
 - o A função **inverte** é aplicada, restaurando a ordem original dos bits.
 - o **msg - 1** remove o bit adicionado anteriormente, e o deslocamento à direita (**msg >> numero**) desfaz o deslocamento original.
3. A mensagem final é convertida de volta para bytes com **msg.to_bytes((msg.bit_length() + 7) // 8, 'big')** retornando o conteúdo original.



```
1 # randomico.py
2
3 75858961944380562850115463223071976485685573428993858563808734375182755772964,
4 925870261805659881574828217847457882318091122038998529440687308308556199580131102818788440649390311,
5 79801342424028570408970225563710939895788798131178233886262812461701488144407453108999997822964928811,
6 944979444675598587417142299169140886898625138568872841822539682252518372188614137155897998631111687
7
8
9
10
11
12
13
14 def inicializa(mat, col, arr):
15     mat = [[0 for _ in range(col)] for _ in range(lin)]
16     for i in range(lin):
17         random.shuffle(arr)
18     mat[i] = arr[i]
19     return mat
20
21 def gera(lin, mat):
22     arr = [0] * lin
23     for i in range(lin):
24         arr[i] = random.choice(mat[i])
25     return arr
26
27 def inverte(numero):
28     return int(str(numero)[::-1])
29
30 def decrypt(msg, arr):
31     for numero in arr[::-1]:
32         msg = inverte(msg)
33         msg = (msg - 1) >> numero
34         msg = msg.to_bytes((msg.bit_length() + 7) // 8, 'big')
35     return msg
36
37 lin = random.choice(range(20, 30))
38 col = random.choice(range(10, 20))
39 a = random.choices(range(1, 10), k=col)
40 M = inicializa(lin, col, a)
41
42 for membro, msg in zip(membros, msgs):
43     A = gera(lin, M)
44     enc = decrypt(msg, A)
45     print(f'[{membro}]: {enc}')
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Obtendo a Flag:

Com a função de descriptografia implementada corretamente, o último passo foi utilizar o script para processar as mensagens recebidas via netcat utilizando a seed exposta para obter a flag:

eits{eu_4mo_num3ros_ps3ud04leat0rlos!!@!}

