

Proyecto:3

Barrero Olguin Adolfo Patricio
Espino Rojas Hector Daniel

Patomap

Requerimientos

Se requiere tener python3 instalado. Junto con la biblioteca termcolor, dicho modulo se puede instalar

```
pip3 install termcolor
```

Ejecucion

El programa cuenta con dos tipos de ejecuciones:

a) Mediante un pid

```
python3 patomap.py -p <pid>
```

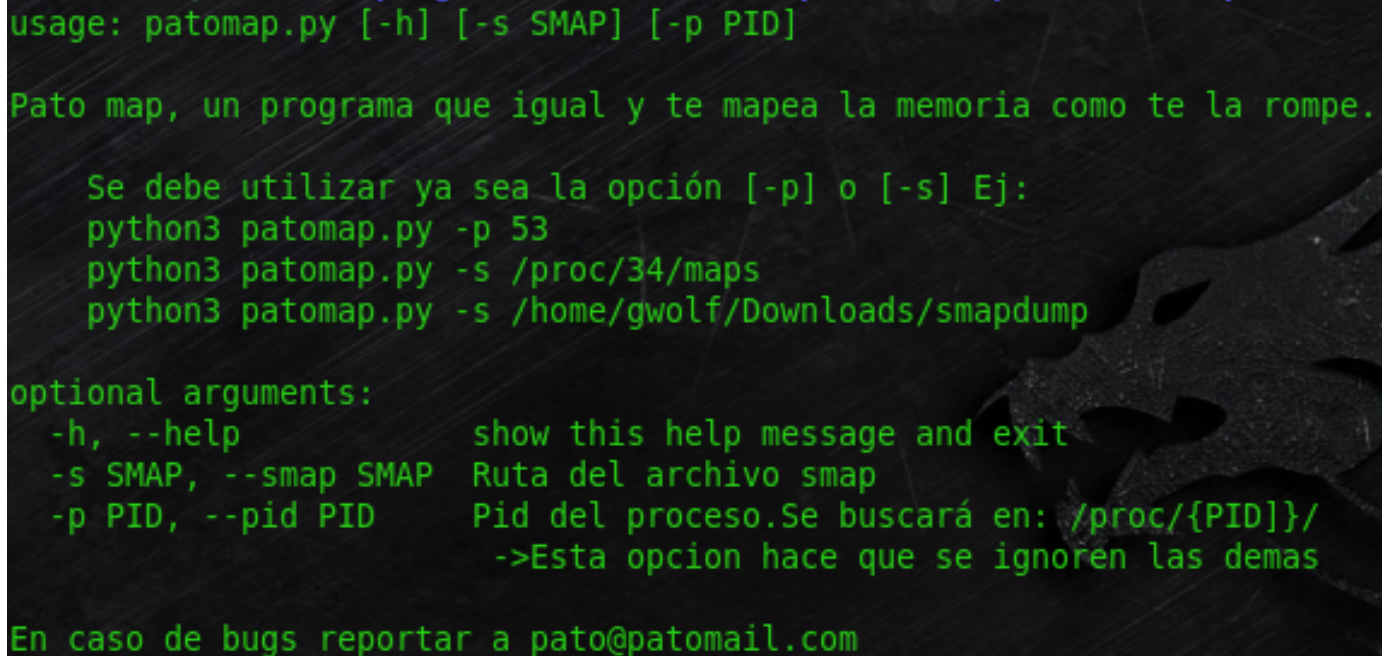
b) Mediante la ruta al archivo smaps

```
python3 patomap.py -s <ruta>
```

c) En caso de requerir ayuda:

```
python3 patomap.py -h
```

Capturas de Pantalla del Programa



```
usage: patomap.py [-h] [-s SMAP] [-p PID]

Pato map, un programa que igual y te mapea la memoria como te la rompe.

Se debe utilizar ya sea la opción [-p] o [-s] Ej:
python3 patomap.py -p 53
python3 patomap.py -s /proc/34/maps
python3 patomap.py -s /home/gwolf/Downloads/smapdump

optional arguments:
  -h, --help                show this help message and exit
  -s SMAP, --smap SMAP      Ruta del archivo smap
  -p PID, --pid PID         Pid del proceso.Se buscará en: /proc/{PID}/
                           ->Esta opcion hace que se ignoren las demas

En caso de bugs reportar a pato@patomail.com
```

USO	DE PAG.	A PAG.	TAMAÑO	NUM. PAG.	PERMISOS	MAPEO
Datos	5572ded17000	5572ded44000	180 kB	45.0	r--	/usr/bin/bash
Texto	5572ded44000	5572dedf2000	696 kB	174.0	r-x	/usr/bin/bash
Datos	5572dedf2000	5572dee28000	216 kB	54.0	r--	/usr/bin/bash
Datos	5572dee28000	5572dee2c000	12 kB	3.0	r--	/usr/bin/bash
Datos	5572dee2c000	5572dee35000	36 kB	9.0	rw-	/usr/bin/bash
vacio	5572dee35000	5572dee3f000	40 kB	10.0	rw-	vacio
Heap	5572e0516000	5572e0599000	524 kB	131.0	rw-	[heap]
Bib-Datos	7fc065724000	7fc065747000	140 kB	35.0	r--	/usr/share/locale/es/LC_MESSAGES/libc.mo
Bib-Datos	7fc065747000	7fc065a2c000	2964 kB	741.0	r--	/usr/lib/locale/locale-archive
vacio	7fc065a2c000	7fc065a2f000	12 kB	3.0	rw-	vacio
Bib-Datos	7fc065a2f000	7fc065a51000	136 kB	34.0	r--	/usr/lib/x86_64-linux-gnu/libc-2.28.so
Bib-Texto	7fc065a51000	7fc065b99000	1312 kB	328.0	r-x	/usr/lib/x86_64-linux-gnu/libc-2.28.so
Bib-Datos	7fc065b99000	7fc065be5000	304 kB	76.0	r--	/usr/lib/x86_64-linux-gnu/libc-2.28.so
???	7fc065be5000	7fc065be6000	4 kB	1.0	---	/usr/lib/x86_64-linux-gnu/libc-2.28.so
Bib-Datos	7fc065be6000	7fc065bea000	16 kB	4.0	r--	/usr/lib/x86_64-linux-gnu/libc-2.28.so
Bib-Datos	7fc065bea000	7fc065bec000	8 kB	2.0	rw-	/usr/lib/x86_64-linux-gnu/libc-2.28.so
vacio	7fc065bec000	7fc065bf0000	16 kB	4.0	rw-	vacio
Bib-Datos	7fc065bf0000	7fc065bf1000	4 kB	1.0	r--	/usr/lib/x86_64-linux-gnu/libdl-2.28.so
Bib-Texto	7fc065bf1000	7fc065bf2000	4 kB	1.0	r-x	/usr/lib/x86_64-linux-gnu/libdl-2.28.so
Bib-Datos	7fc065bf2000	7fc065bf3000	4 kB	1.0	r--	/usr/lib/x86_64-linux-gnu/libdl-2.28.so
Bib-Datos	7fc065bf3000	7fc065bf4000	4 kB	1.0	r--	/usr/lib/x86_64-linux-gnu/libdl-2.28.so
Bib-Datos	7fc065bf4000	7fc065bf5000	4 kB	1.0	rw-	/usr/lib/x86_64-linux-gnu/libdl-2.28.so
Bib-Datos	7fc065bf5000	7fc065c03000	56 kB	14.0	r--	/usr/lib/x86_64-linux-gnu/libtinfo.so.6.1
Bib-Texto	7fc065c03000	7fc065c11000	56 kB	14.0	r-x	/usr/lib/x86_64-linux-gnu/libtinfo.so.6.1
Bib-Datos	7fc065c11000	7fc065c1e000	52 kB	13.0	r--	/usr/lib/x86_64-linux-gnu/libtinfo.so.6.1
Bib-Datos	7fc065c1e000	7fc065c22000	16 kB	4.0	r--	/usr/lib/x86_64-linux-gnu/libtinfo.so.6.1
Bib-Datos	7fc065c22000	7fc065c23000	4 kB	1.0	rw-	/usr/lib/x86_64-linux-gnu/libtinfo.so.6.1
vacio	7fc065c23000	7fc065c25000	8 kB	2.0	rw-	vacio
Bib-Datos	7fc065c25000	7fc065c28000	12 kB	3.0	r--	/usr/lib/x86_64-linux-gnu/libnss_files-2.28.so
Bib-Texto	7fc065c28000	7fc065c2f000	28 kB	7.0	r-x	/usr/lib/x86_64-linux-gnu/libnss_files-2.28.so
Bib-Datos	7fc065c2f000	7fc065c31000	8 kB	2.0	r--	/usr/lib/x86_64-linux-gnu/libnss_files-2.28.so
???	7fc065c31000	7fc065c32000	4 kB	1.0	---	/usr/lib/x86_64-linux-gnu/libnss_files-2.28.so
Bib-Datos	7fc065c32000	7fc065c33000	4 kB	1.0	r--	/usr/lib/x86_64-linux-gnu/libnss_files-2.28.so
Bib-Datos	7fc065c33000	7fc065c34000	4 kB	1.0	rw-	/usr/lib/x86_64-linux-gnu/libnss_files-2.28.so
vacio	7fc065c34000	7fc065c3a000	24 kB	6.0	rw-	vacio

Lógica del Programa

El programa sigue la siguiente logica

1. El programa detecta si el usuario ingreso el pid o la direccion, si ingreso el pid obtiene la direccion del archivo smaps.
2. Lee el contenido del archivo.
3. Del archivo leído obtiene los puntos importantes del segmento de memoria
4. Con los datos obtenidos infiere a que región de memoria hace referencia
5. Muestra en pantalla el resultado obtenido

Reconocimiento de las Regiones de Memoria

Realiza el reconocimiento mediante las siguientes reglas:

Stack: El segmento contiene [stack] en el archivo smaps
Heap: El segmento contiene [heap] en el archivo smaps
Mapeo Anonimo: El segmento contiene [anon] en el archivo smaps
Llamada al Sistema: El segmento contiene [vdso], [syscalls], o [vectors] en el archivo smaps
Var Kernel: El segmento contiene [vvar] en el archivo smaps
Bib→Texto: El segmento se puede ejecutar y así referencia a algún "lib"
Bib→Datos: El segmento se puede leer y así referencia a algún "lib"
Texto: El segmento se puede ejecutar y así referencia a "/usr/bin"
Datos: El segmento se puede leer y así referencia a algún "/usr/bin"

Analisis:

archivo C

Analisis e identificación de regiones:

Primero comenzamos realizando un analisis estatico con el codigo fuente:
Identificamos las distintas areas del source y damos una hipotesis de que encontraríamos:

Texto: Instrucciones a ser ejecutadas

Codigo a ejecutar
Seccion de datos:##variables globales y cadenas de caracteres.
cadena1, "Yo solo se que no se nada"
cadena_total
tamano = 30

Posiblemente aquí esten las demas cadenas embebidas en las funciones

Heap:

Dependiendo el momento en que se haya realizado el dump,
puede que encuentre aquí cadena 2,3 y4.
Ademas de cadena total

Stack: Funciones, variables locales

Podria estar la referencia a main, y los apuntadores a las cadenas en el heap. Depende el momento en que se realizara el dump

Se utilizan las siguientes librerias:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
```

ejecutable

Procedemos a analizar el ejecutable:

```
file donde_en_la_memoria
donde_en_la_memoria: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically link
ed, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=23f11f4ff28b90
27d3bf54ab2d3c9a387d2e2c37, not stripped
```

Al parecer el archivo no era un x86_32 como pudo haber parecido a priori, y vemos algo sobre dynamically linked interpreter. Esto nos da a entender que se encuentra ligado de forma dinamica y no veremos las bibliotecas de terceros en nuestro ejecutable.

Acorde a: <https://stackoverflow.com/questions/311882/what-do-statically-linked-and-dynamically-linked-mean>

Podemos esperar encontrar las bibliotecas en el core, ya que en memoria estas si son traídas.

interpreter /lib64/ld-linux-x86-64.so.2

Es un archivo ejecutable, que al correrlo obtenemos los siguientes datos:

```
Usage: ld.so [OPTION]... EXECUTABLE-FILE [ARGS-FOR-PROGRAM...]
You have invoked `ld.so', the helper program for shared library executables.
This program usually lives in the file `/lib/ld.so', and special directives
in executable files using ELF shared libraries tell the system's program
loader to load the helper program from this file. This helper program loads
the shared libraries needed by the program executable, prepares the program
to run, and runs it. You may invoke this helper program directly from the
command line to load and run an ELF executable file; this is like executing
that file itself, but always uses this helper program from the file you
specified, instead of the helper program file specified in the executable
file you run. This is mostly of use for maintainers to test new versions
of this helper program; chances are you did not intend to run this program.
```

Pareciera entonces que este interprete se encargará de avisar al cargador al programa ayudante. El programa ayudante será encargado de cargar las librerias que necesita el ejecutable.

Cambiando de estrategia:

Utilizando:

```
strings donde_en_la_memoria | sort
nada
que nada s
que no s
usted!
;*3$"
8}.,7
[]\A\A]A^A_
.bss
__bss_start
cadena1
cadena_total
.comment
completed.7325
construye_final
crtstuff.c
__cxa_finalize
__cxa_finalize@@GLIBC_2.2.5
.data
__data_start
deregister_tm_clones
__do_global_dtors_aux
__do_global_dtors_aux_fini_array_entry
donde_en_la_memoria.c
__dso_handle
.dynamic
_DYNAMIC
.dynstr
.dynsym
.edata
.eh_frame
.eh_frame_hdr
.fini
.fini_array
frame_dummy
__frame_dummy_init_array_entry
__FRAME_END__
free
free@@GLIBC_2.2.5
GCC: (Debian 8.3.0-6) 8.3.0
getc
getc@@GLIBC_2.2.5
getpid
getpid@@GLIBC_2.2.5
GLIBC_2.2.5
_GLOBAL_OFFSET_TABLE_
__gmon_start__
__gmon_start__
__GNU_EH_FRAME_HDR
.gnu.hash
.gnu.version
.gnu.version_r
.got.plt
.init
.init_array
__init_array_end
__init_array_start
.interp
_IO_stdin_used
_ITM_deregisterTMCloneTable
_ITM_deregisterTMCloneTable
_ITM_registerTMCloneTable
_ITM_registerTMCloneTable
/lib64/ld-linux-x86-64.so.2
__libc_csu_fini
__libc_csu_init
libc.so.6
__libc_start_main
```

```

__libc_start_main@@GLIBC_2.2.5
lo s
main
malloc
malloc@@GLIBC_2.2.5
mgUa
.note.ABI-tag
.note.gnu.build-id
Pero si alguien sabe menos
.plt.got
Proceso fil
puede ser
puts
puts@@GLIBC_2.2.5
.rela.dyn
.rela.plt
.rodata
.shstrtab
siempr
snprintf
snprintf@@GLIBC_2.2.5
sofo, PID %d
stdin
stdin@@GLIBC_2.2.5
strncat
strncat@@GLIBC_2.2.5
strncpy
strncpy@@GLIBC_2.2.5
.strtab
.symtab
tamano
.text
__TMC_END__
u/UH
Yo s
Yo solo s

```

Intentamos observar repeticiones de las cadenas, y notamos que se encuentran ahí las distintas cadenas que se encuentran en el archivo, nombres de funciones que llama -interesante mencionar estas son traídas por las bibliotecas pero estas no son mostradas, ej: string y strncpy-, secciones del archivo ELF(.text por ejemplo) y algunos nombres de variables.

core

Utilizando el archivo core:

Vemos el tipo de archivo:
file donde_en_la_memoria.core

Y obtenemos:

```

donde_en_la_memoria.core: ELF 64-bit LSB core file, x86_64, version 1 (SYSV), SVR4-style, from
'donde_en_la_memo./donde_en_la_memoria', real uid: 1000, effective uid: 1000, real gid: 1000,
effective gid: 1000, execfn: './donde_en_la_memoria', platform: 'x86_64'

```

uid 1000? ¿Será ese el usuario gworld del profesor?

Encontramos también que se trata de un ELF 64 bits LSB core file
Elf significa “executable and linking format” ELF

A partir del LSB core file encontramos el siguiente recurso:

<https://www.ibm.com/support/pages/debugging-core-files-02-how-prepare-core-file-analysis>

Dentro del cual se nos menciona que las regiones de texto no se encontrarán dentro del core, tampoco la tabla de símbolos. Por lo cual no tendrá todo lo que podríamos observar dentro del archivo core todo lo que realmente está en memoria. Salvo que utilizemos recursos además del core, como tener un ejecutable.

Desafortunadamente el siguiente recurso: https://web.eecs.umich.edu/~sugih/pointers/gdb_core.html

Nos dice lo siguiente: **“A menos que el programa sea compilado con información de depuración, tendremos información bastante ilegible -cryptic-”**

Por lo que esta aproximación puede ser no muy adecuada para encontrar algo interesante.

Ejecutando un analisis estatico obtenemos lo siguiente:

```
strings donde_en_la_memoria.core | sort
```

(Resultado en el archivo analisisCore.strings)

Encontramos varias repeticiones de las cadenas y que el usuario efectivamente es gwolf con uid 1000, el correo y algunas variables de entorno:

Vemos que el profesor tiene ssh-agent, eso explica porque nunca utiliza pass o especifica una llave privada.

En caso de no ordenar las salidas: Obtenemos hasta el final las secciones: .text, .comment, etc

Y podemos observar que las cadenas tienen algunos patrones de repeticion:

```
Yo s
lo s
que nada s
Pero si alguien sabe menos
puede ser
;*3$"
Yo solo s
que no s
nada
Pero si alguien sabe menos
```

Analisis con gdb y utilizando el recurso de ibm:

-Buscando los módulos cargados en tiempo de ejecución:

```
(gdb) info sharedlibrary
From          To          Syms Read  Shared Object Library
0x00007fd3a61ec320 0x00007fd3a633239b Yes        /lib/x86_64-linux-gnu/libc.so.6
0x00007fd3a63d1090 0x00007fd3a63eeb20 Yes        /lib64/ld-linux-x86-64.so.2
```

//Libc y ld, bastante standard

Analizando el backtrace (Lo cual nos muestra el stack del programa):


```

(gdb) bt
#0  0x00007fd3a62b4461 in __GI__libc_read (fd=0, buf=0x557c8fb4f750, nbytes=1024)
    at ../sysdeps/unix/sysv/linux/read.c:26
#1  0x00007fd3a6246670 in _IO_new_file_underflow (fp=0x7fd3a6385a00 <_IO_2_1_stdin_>)
    at libioP.h:839
#2  0x00007fd3a62477b2 in __GI__IO_default_uflow (fp=0x7fd3a6385a00 <_IO_2_1_stdin_>)
    at libioP.h:839
#3  0x0000557c8e4642ae in main ()
(gdb) bt full
#0  0x00007fd3a62b4461 in __GI__libc_read (fd=0, buf=0x557c8fb4f750, nbytes=1024)
    at ../sysdeps/unix/sysv/linux/read.c:26
        resultvar = 18446744073709551104
        sc_ret = <optimized out>
        sc_ret = <optimized out>
        resultvar = <optimized out>
        resultvar = <optimized out>
        __arg3 = <optimized out>
        __arg2 = <optimized out>
        __arg1 = <optimized out>
        _a3 = <optimized out>
        _a2 = <optimized out>
        _a1 = <optimized out>
        sc_cancel_oldtype = <optimized out>
        resultvar = <optimized out>
        resultvar = <optimized out>
        __arg3 = <optimized out>
        __arg2 = <optimized out>
        __arg1 = <optimized out>
        _a3 = <optimized out>
        _a2 = <optimized out>
        _a1 = <optimized out>
#1  0x00007fd3a6246670 in _IO_new_file_underflow (fp=0x7fd3a6385a00 <_IO_2_1_stdin_>)
    at libioP.h:839
        count = <optimized out>
#2  0x00007fd3a62477b2 in __GI__IO_default_uflow (fp=0x7fd3a6385a00 <_IO_2_1_stdin_>)
    at libioP.h:839
        ch = <optimized out>
#3  0x0000557c8e4642ae in main ()
No symbol table info available.

```

Encontrando esto:

Podemos ver que la tabla de simbolos no nos será disponible, al menos mediante este método.

Podemos ver que se encontraba dentro de main al ser tomado el dumpfile. Como hipotesis podemos establecer que main llamo a IO, posiblemente se encontraba en la espera delgetc,

ya que se encontraba utilizando __GI__libc_read

El stack esta de este modo

main-> __GI__IO -> _IO_new_file -> __GI__libc_read

“No symbol table info available” -> Problemas, no podremos obtener información de no tener la tabla de simbolos, posiblemente debido a la explicación de IBM.

Algunas funciones no serán encontradas por lo mismo (Hipotesis de los alumnos)

Al realizar up varias veces en aras de encontrar información, de las variables locales nos encontramos con lo siguiente, por lo que no encontramos nada dentro de las funciones en el frame de main:

```

(gdb) info locals
No symbol table info available.

```

Encontramos información de los siguientes registros:

```
(gdb) info registers
rax      0xfffffffffffffe00    -512
rbx      0x0                    0
rcx      0x7fd3a62b4461        140547002680417
rdx      0x400                 1024
rsi      0x557c8fb4f750        93993475307344
rdi      0x0                    0
rbp      0x7ffc05564eb0        0x7ffc05564eb0
rsp      0x7ffc05564e90        0x7ffc05564e90
r8       0x7fd3a63878c0        140547003545792
r9       0x7fd3a638c500        140547003565312
r10      0x557c8fb4f010        93993475305488
r11      0x246                 582
r12      0x557c8e4640d0        93993451274448
r13      0x7ffc05564f90        140720398028688
r14      0x0                    0
r15      0x0                    0
rip      0x557c8e4642ae        0x557c8e4642ae <main+249>
eflags   0x246                 [ PF ZF IF ]
cs       0x33                 51
ss       0x2b                 43
ds       0x0                    0
es       0x0                    0
fs       0x0                    0
gs       0x0                    0
```

Con “info functions”, parece que no obtenemos otra cosa que las funciones de las bibliotecas llamadas por el archivo: strings,malloc,etc.

Al estudiar info variables, encontramos:

```
0x0000557c8e467070  cadena1
```

Parece ser que se encuentran algunos simbolos de cadena.

```
0x0000557c8e467070  cadena1
```

```
0x0000557c8e4670a0  cadena_total
```

```
/*mas otras que no vimos por el sueño*/
```

Por lo que pudimos encontrar:

```
(gdb) info symbol 0x0000557c8e467070
cadena1 in section .data of /home/usuario/proyectos/3/donde en la memoria/donde en la memoria
(gdb) info symbol 0x0000557c8e4670a0
cadena_total in section .bss of /home/usuario/proyectos/3/donde en la memoria/donde en la memoria
```

Aquí vemos que algunas variables y en que area se encuentran.

Cambiando a otra estrategia, se hacen busquedas directas por las variables, se obtiene lo siguiente:

Variables y cadenas encontradas:

Acorde a lo esperado, estas se encuentran en la definición del codigo y no en el heap ni stack

Buscando la cadena1: Encontramos su dirección: 0x557c8e467070

```
(gdb) info address cadena1
Symbol "cadena1" is at 0x557c8e467070 in a file compiled without debugging.
(gdb) p &cadena1
$7 = (<data variable, no debug info> *) 0x557c8e467070 <cadena1>
```

Y la imprimimos:

```
(gdb) x /s 0x557c8e467070
0x557c8e467070 <cadena1>: "Yo solo sé que no sé nada"
```

La variable cadena1 se encuentra en el area de datos(permisos rw) en la pagina:

```
557c8e467000-557c8e468000 rw-p 00003000 fe:01 11289373 /home/gwolfo/vcs/sistop-2020-2/proyectos/3/ejemplos/donde_en_la_memoria
```

```
(gdb) info symbol 0x557c8e467070
cadena1 in section .data of /home/hectorhmx/programacion/sistemas0perativos/sistop-2020-2/proyectos/3/donde_en_la_memoria/donde_en_la_m
```

Buscando el resto de las cadenas utilizaremos la siguiente aproximación:

find [dir ini],+tamano,"string"

Podemos encontrar lo siguiente:

```
(gdb) find 0x557c8e466000,+1000,"Pero si alguien sabe menos"
0x557c8e46603a
1 pattern found.
(gdb) x/s 0x557c8e46603a
0x557c8e46603a: "Pero si alguien sabe menos"
```

```
(gdb) find 0x557c8e466000,+1000,"Yo sólo sé que nada sé"
0x557c8e466020
1 pattern found.
(gdb) x/s 0x557c8e466020
0x557c8e466020: "Yo sólo sé que nada sé"
```

```
(gdb) find 0x557c8e466000,+1000,"puede ser"
0x557c8e466062
1 pattern found.
(gdb) x/s 0x557c8e466062
0x557c8e466062: "puede ser"
```

Las dos cadenas anteriores se encuentran en la pagina: que es el area de datos y se puede comprobar por los permisos de lectura

```
557c8e466000-557c8e467000 r--p 00002000 fe:01 11289373 /home/gwolfo/vcs/sistop-2020-2/proyectos/3/ejemplos/donde_en_la_memoria
```

Encontramos donde estan las funciones y variables:

Funciones:

```
(gdb) info address main
Symbol "main" is at 0x557c8e4641b5 in a file compiled without debugging.
(gdb) info address construye_final
Symbol "construye_final" is at 0x557c8e4642dc in a file compiled without debugging.
```

main tiene la dirección: 557c8e4641b5
e info tiene la dirección:557c8e4642dc

Encontrándose en la siguiente pagina: y dentro del area de texto

```
557c8e464000-557c8e465000 r-xp 00001000 fe:01 11289373 /home/gwolfo/vcs/sistop-2020-2/proyectos/3/ejemplos/donde_en_la_memoria
```

Variables:

```
(gdb) info address cadena_total
Symbol "cadena_total" is at 0x557c8e4670a0 in a file compiled without debugging.
```

Se encuentra en la siguiente pagina cadena_total y es en el area de .bss:

-> La diferencia entre .bss y .data es que .bss tiene objetos no inicializados aún.

```
(gdb) info symbol 0x557c8e4670a0
cadena_total in section .bss of /home/hectorhmx/programacion/sistemasoperativos/sistop-2020-2/proyectos/3/donde_en_la_memoria/donde_en_la_memoria
```

```
557c8e467000-557c8e468000 rw-p 00003000 fe:01 11289373 /home/gwolfo/vcs/sistop-2020-2/proyectos/3/ejemplos/donde_en_la_memoria
```

¿Porque no puedo encontrar algunas?

Habrà variables que no serán encontradas: .vavar (accesibles solo nivel kernel) y estas tienen la bandera bb en smaps

Variable no cadena de texto:

tamano is at 0x557c8e46708c y tiene el tamaño de 30 (area de datos)

```
(gdb) i proc mapping
Mapped address spaces:

      Start Addr      End Addr      Size      Offset objfile
0x557c8e466000      0x557c8e467000      0x1000      0x2000 /home/gwolfo/vcs/sistop-2020-2/proyectos/3/ejemplos/donde_en_la_memoria
0x557c8e467000      0x557c8e468000      0x1000      0x3000 /home/gwolfo/vcs/sistop-2020-2/proyectos/3/ejemplos/donde_en_la_memoria
0x7fd3a6381000      0x7fd3a6385000      0x4000      0x1b6000 /lib/x86_64-linux-gnu/libc-2.28.so
0x7fd3a6385000      0x7fd3a6387000      0x2000      0x1ba000 /lib/x86_64-linux-gnu/libc-2.28.so
0x7fd3a63f7000      0x7fd3a63f8000      0x1000      0x260000 /lib/x86_64-linux-gnu/ld-2.28.so

(gdb) info address tamano
Symbol "tamano" is at 0x557c8e46708c in a file compiled without debugging.
(gdb) print (int) tamano
$3 = 30
```

Elementos validos que no corresponden al codigo fuente del profesor:

Encontramos num_ifs, al filtrar las busqueda en info variables

```
0x00007fd3a6385428 num_ifs
```

Esta variable (apuntador a entero) es parte de una función de glibc acorde a estos mirros (unofficial)de glibc

<https://github.com/bminor/glibc/blob/master/sysdeps/unix/ifreq.c>

<https://code.woboq.org/userspace/glibc/sysdeps/unix/sysv/linux/ifreq.c.html>

Encontramos las siguientes secciones mediante el uso de elf y lef:

```
(gdb) info files
Symbols from "/home/gwolfo/vcs/sistop-2020-2/proyectos/3/donde_en_la_memoria/donde_en_la_memoria".
Local core dump file: "/home/gwolfo/vcs/sistop-2020-2/proyectos/3/donde_en_la_memoria/donde_en_la_memoria.core",
file type elf64-x86-64.
0x0000557c8e466000 - 0x0000557c8e467000 is load1
0x0000557c8e467000 - 0x0000557c8e468000 is load2
0x0000557c8fb4f000 - 0x0000557c8fb70000 is load3
0x00007fd3a6381000 - 0x00007fd3a6385000 is load4
0x00007fd3a6385000 - 0x00007fd3a6387000 is load5
0x00007fd3a6387000 - 0x00007fd3a638d000 is load6
0x00007fd3a63f7000 - 0x00007fd3a63f8000 is load7
0x00007fd3a63f8000 - 0x00007fd3a63f9000 is load8
0x00007fd3a63f9000 - 0x00007fd3a63fa000 is load9
0x00007ffc05546000 - 0x00007ffc05567000 is load10
0x00007ffc05581000 - 0x00007ffc05583000 is load11
```

bibliografia

<https://stackoverflow.com/questions/19859019/what-vectors-mean-in-smaps>
<https://github.com/bminor/glibc/blob/master/sysdeps/unix/ifreq.c>
<https://code.woboq.org/userspace/glibc/sysdeps/unix/sysv/linux/ifreq.c.html>
<https://stackoverflow.com/questions/311882/what-do-statically-linked-and-dynamically-linked-mean>
<https://www.ibm.com/support/pages/debugging-core-files-02-how-prepare-core-file-analysis>
https://web.eecs.umich.edu/~sugih/pointers/gdb_core.html