GDB para reversing

Hacking Ético

©Ismael Ripoll &
Hector Marco

Universidad Politècnica de València

February 1, 2022

Índice

Qué vamos a trabajar (I)

- El depurador GDB (GNU DeBugger) es una potente herramienta para depurar todo tipo de procesos.
- En esta práctica aprenderéis a utilizarlo para seguir código ensamblador. No vamos a depurar programas fuente sino que queremos "ver" el funcionamiento a nivel de código máquina.
- También veremos como se puede modificar la ejecución del proceso que depuramos.
- GDB permite trabajar con casi cualquier procesador, por lo que aprenderlo bien es muy rentable.
- La interfaz es "desagradable" (fea, cutre, viejuna, poco robusta, etc...) pero es muy potente. Así que aguanta el tipo y haz el esfuerzo.

https://undo.io/resources/presentations/cppcon-2015-greg-law-give-me-15-minutes-ill-change/

Preparación (I)

- El compilador GCC está en continua mejora. Continuamente están añadiendo nuevas optimizaciones en eficiencia y protecciones de seguridad.
  Por defecto, es recomendable utilizar los siguientes flags:

  "`-m32`" Le decimos al compilador que genere código para 32 bits. Por defecto lo genera para 64, lo que hace que las direcciones sean más largas y más difíciles de recordar.

  "`-O0`" menos o (mayúscula) cero. Esto desactiva las estrategias de optimización de código. Con lo que el código generado se puede asociar más fácilmente como el código fuente.

  "`-fno-pie -no-pie`" Genera código dependiente de la posición. Como resultado el ejecutable siempre se cargará en las mismas direcciones de memoria.

Preparación (II)

"`-D_FORTIFY_SOURCE=0`" No queremos que nos añada comprobaciones de seguridad (o corrija) si escribimos código con fallos. Y lo vamos a hacer!

"`-fno-stack-protector`" Desactivar otro mecanismo de protección. Que estudiaremos a final del curso. Pero que no necesitamos.

"`-fno-inline-small-functions`" El compilador es tan "optimizante" por defecto que tenemos que decirle que no incruste funciones dentro de otras.

"`-ggdb`" Añadimos información de depuración para poder seguir el código fuente a la vez que vemos el ensamblador.

- Puede que la versión que instalada sea todavía más moderna y tenga nuevas funcionalidades que haya que desactivar para poder tener código fácil de comprender y atacar.

Primer programa

- Escribe un programa en “C” que imprima “Hello World”.
- Compílalo y ejecútalo:

```
$ gcc -ggdb -m32 -O0 -fno-pie -no-pie hello.c -o hello
```

- No utilices el flag de compilación “-ggdb” ya que no vamos a depurar el fuente.
- Ahora ejecuta el programa desde el GDB:

```
$ gdb -args ./test
(gdb) run
```

Interfaz (I)

- Lanza el GDB con tu programa.
- Para activar el modo TUI (Text User Interface) pulsa [<Ctrl>x 2] dos veces. Pulsa control-x, se liberan las teclas y luego el número 2.
- Verás tres ventanas de texto:
  1. En la parte de arriba los registros y sus valores
  2. En el medio el código ensamblador,
  3. y abajo el consola de comandos gdb.
- Ahora ponemos un breakpoint en `main` y ejecutamos el programa "con `run`":

```
(gdb) b main
Breakpoint 1 at 0x6a4
(gdb) run
```

Interfaz (II)

```
Register group: general
eax            0xffffd1e0         -11808             ecx            0xffffd1e0         -11808
edx            0xffffd204         -11772             ebx            0x0                0
esp            0xffffd1c0         0xffffd1c0         ebp            0xffffd1c8         0xffffd1c8
esi            0xf7faa000         -134569984         edi            0xf7faa000         -134569984
eip            0x80491b7          0x80491b7 <main+19>   eflags         0x286              [ PF SF IF ]
cs             0x23               35                 ss             0x2b               43
ds             0x2b               43                 es             0x2b               43
fs             0x0                0                  gs             0x63               99
```

```
   0x80491a4 <main>        lea    0x4(%esp),%ecx
   0x80491a8 <main+4>      and    $0xfffffff0,%esp
   0x80491ab <main+7>      pushl  -0x4(%ecx)
   0x80491ae <main+10>     push   %ebp
   0x80491af <main+11>     mov    %esp,%ebp
   0x80491b1 <main+13>     push   %ecx
   0x80491b2 <main+14>     sub    $0x4,%esp
   0x80491b5 <main+17>     mov    %ecx,%eax
B+>0x80491b7 <main+19>     mov    0x4(%eax),%eax
   0x80491ba <main+22>     mov    (%eax),%eax
   0x80491bc <main+24>     sub    $0x8,%esp
```

```
native process 16916 In: main                                          L5    PC: 0x80491b7
(gdb) b main
Punto de interrupción 1 at 0x80491b7: file hello.c, line 5.
(gdb) run
Starting program: /home/iripoll/Clases/clases/Hacking-Etico/pract_02_gdb/hello

Breakpoint 1, main (argc=1, argv=0xffffd274) at hello.c:5
(gdb)
```

Interfaz (III)

- Familiarízate con los registros del procesador, los valores mostrados y el resto de numeritos.
- Aunque parezca todo son números, es más fácil de lo que parece.
- El foco del las flechas y el scroll del teclado están en la ventana del ensamblador. Para cambiar el foco entre ventanas utilizamos [<Ctrl>-x o]. Pasa el foco a la ventana de comandos.
- NOTA: una vez el foco esta en los comandos, con el TABulador podemos completar las ordenes y con las flechas podemos recuperar comandos anteriores. Igual que hace el bash.
- El comando `help` da ayuda de todo, y puede completar.

Ejecucuión controlada (I)

- El comando "`si [N]`" (stepi) ejecuta la siguiente instrucción. Utilízalo varias veces y podrás ver como avanza el contador de programa (registro `pc`) y como van cambiando los registros.
- El comando "`c [N]`" (continue) continua la ejecución hasta el siguiente breakpoint o señal.
- El comando "`n [N]`" (next) continua la ejecución pero las llamadas a función (calls) no las sigue.
- Todos estos comandos aceptan un parámetro numérico, `N`, que indica las veces que el comando se ejecuta.
- Tras cada comando, si se pulsa `<Enter>` se vuelve a ejecutar el mismo.

Inspección de memoria (I)

- El comando "`x/FMT addr`" examina el contenido de una posición de memoria o registro. Formato de `FMT`:
  - **x** hexadecimal.
  - **u** decimal sin signo.
  - **t** binario.
  - **i** instrucción.
  - **s** string, cadena que acaba en cero.
- Se pueden utilizar modificadores después del tipo: b(byte), h(halfword), w(word), g(giant, 8 bytes).
- También se puede indicar cuantas posiciones se quiere ver, con un número **delante** del formato.
- La dirección puede ser el nombre de un registro con un $ delante, para diferenciarlo de posibles direcciones.

Inspección de memoria (II)

```
(gdb) x /i $pc
=> 0x80491b7 <main+19>: mov 0x4(%eax),%eax
(gdb) x /i 0x80491b7
=> 0x80491b7 <main+19>: mov 0x4(%eax),%eax
(gdb) x /5i 0x80491b7
=> 0x80491b7 <main+19>: mov 0x4(%eax),%eax
   0x80491ba <main+22>: mov (%eax),%eax
   0x80491bc <main+24>: sub $0x8,%esp
   0x80491bf <main+27>: push %eax
   0x80491c0 <main+28>: push $0x804a01e
(gdb) x /8xw $sp
0xffffd1c0: 0xf7fe45a0 0xffffd1e0 0x00000000 0xf7def751
0xffffd1d0: 0xf7faa000 0xf7faa000 0x00000000 0xf7def751
(gdb) x /2s $sp+1130
0xffffd62a:   "SESSION_DESKTOP=LXDE"
0xffffd63f:   "LOGNAME=iripoll"
```

Inspección de memoria (III)

- Fíjate que los dos primeros comandos son equivalentes (registro y dirección).
- Observa el último comando (`x /2s $sp+1130`) ¿De dónde crees que es eso de `SHELL=/bin/bash`"?
- Busca en tu programa esa cadena.
- Cada arquitectura tiene su propio juego de registros (con distintos nombres), pero como en todas existen el "contador de programa" y el "puntero de pila", GDB define los nombres `pc` y `sp` como nombres de registros independientes de arquitectura para referirse a esos registros. Por tanto, en i386 `pc` equivale al registro `eip` y el `sp` equivale al `esp`.

Datos del proceso (I)

- El comando "`i`" (info) permite listar multitud de información (help info).
- Recuerda que hasta que no esté creado el proceso (run), no se podrá listar esta información.
- El comando "`i r`" lista los registros.
- El comando "`i proc mappings`" lista los registros.

Breakpoints (I)

- El comando "`catch`" sirve para establecer puntos de captura genéricos. Los más interesantes son:
  - El comando "`catch syscall`" detiene la ejecución en cada llamada a sistema.
  - El comando "`catch signal`" cuando llega una señal (excepto SIGTRAP, que es usada por el propio GDB).
- El comando "`break`" es uno de los más usados. La sintaxis es:

```
break [MODIFIER] [LOCATION] [thread NUM] [if COND]
```

el campo LOCATION puede ser el nombre de una función o una dirección de memoria. En caso de ser una dirección se tiene que indicar con un "*" delante.

Breakpoints (II)

```
(gdb) b *0x400534
Punto de interrupción 3 at 0x400534
Continuando.
(gdb) c
Hello World
Breakpoint 3, 0x0000000000400534 in main ()
```

- El proceso se para cuando llega al breakpoint.
- Pon catch points en las llamadas al sistema y ejecuta el programa de syscall en syscall.
- Se puede relanzar el programa con el comando `run`.
- El comando "`i breakpoints`" muestra todos los puntos de ruptura instalador.
- El comando "`delete [NR]`" elimina uno o varios puntos de ruptura.

Moficar el programa (I)

- El comando "`set VAR=EXP`" permite modificar registros del procesador y la memoria.
- En el siguiente ejemplo se modifica el código del ejecutable.

```
(gdb) x /i $pc
=> 0x80491b7 <main+19>: mov 0x4(%eax),%eax
(gdb) set *(0x80491b7)=0x90909090
(gdb) x /i $pc
=> 0x80491b7 <main+19>: nop
   0x80491b8 <main+20>: nop
   0x80491b9 <main+21>: nop
   0x80491ba <main+22>: nop
```

Actividades de la práctica (I)

1. A partir del siguiente programa:

```
#include <stdio.h>
void primera(){ printf("En primera\n"); }
void segunda(){ printf("En segunda\n"); }
int main(int argc, char *argv[]){
  printf("Soy: %s\n", argv[0]);
  primera();
  segunda();
}
```

   Compílalo y ejecútalo paso a paso.

2. Vuelve a ejecutarlo paso a paso pero modifica el contador de programa (`pc`) al llegar a la llamada a primera para saltarse la llamada a esa función y que pase directamente al call a segunda.

Actividades de la práctica (II)

3. Imprime el contenido de la pila (`sp`). Trata de identificar si lo que hay en la pila son direcciones y a qué zonas de memoria apuntan.
4. Ejecuta varias veces el mismo programa y comprueba a ver si las direcciones son las mismas.