Raj Kane

राज काणे

BLOG
ABOUT
CONTACT

Debugging Stripped Binaries in GDB

2022-12-11

Debugging executables is all well and good when they are compiled with GCC's `-g` flag to retain debugging information, but hackers have to deal with **stripped** binaries. By removing symbolic information unessential for correct execution, stripping not only saves disk space and potentially improves performance, but—in the context of security—serves as one level of obfuscation against prying eyes.

We examine the effects of stripping using GCC's `-s` flag on the following toy program that prints out a UID.

```
➜ cat getuid.c
#include <stdio.h>
#include <unistd.h>

int main() {
    printf("UID: %d\n", geteuid());
}
➜  gcc getuid.c -o getuid
➜  ./getuid
UID: 1000
```

After compiling the program without any stripping, we can open the executable with GDB and readily place a breakpoint at a symbol. Shoutout to the extremely handy GDB enhancer, GEF.

```
➜ gdb getuid
Reading symbols from getuid...
gef➤  b main
Breakpoint 1 at 0x114d
```

But when we compile and strip the binary, we no longer have this luxury.

```
➜ gcc -s getuid.c -o getuid
➜ gdb getuid
```

```
Reading symbols from getuid...
Debuginfod has been disabled.
(No debugging symbols found in getuid)
gef➤  b main
Function "main" not defined.
```

Not to worry! We can still find our way in this seemingly hopeless scenario by finding the program's **entry point** offset, then its **start address**, and finally our destination.

Within GDB, we find the entry point with the command `info files`.

```
gef➤  info files
Symbols from "/home/raj/getuid".
Local exec file:
        `/home/raj/getuid', file type elf64-x86-64.
        Entry point: 0x1050
        0x0000000000000318 - 0x0000000000000334 is .interp
        0x0000000000000338 - 0x0000000000000378 is .note.gnu.property
        0x0000000000000378 - 0x000000000000039c is .note.gnu.build-id
        0x000000000000039c - 0x00000000000003bc is .note.ABI-tag
        0x00000000000003c0 - 0x00000000000003dc is .gnu.hash
        0x00000000000003e0 - 0x00000000000004a0 is .dynsym
        0x00000000000004a0 - 0x0000000000000537 is .dynstr
        0x0000000000000538 - 0x0000000000000548 is .gnu.version
        0x0000000000000548 - 0x0000000000000578 is .gnu.version_r
        0x0000000000000578 - 0x0000000000000638 is .rela.dyn
        0x0000000000000638 - 0x0000000000000668 is .rela.plt
        0x0000000000001000 - 0x000000000000101b is .init
        0x0000000000001020 - 0x0000000000001050 is .plt
        0x0000000000001050 - 0x000000000000116f is .text
        0x0000000000001170 - 0x000000000000117d is .fini
        0x0000000000002000 - 0x000000000000200d is .rodata
        0x0000000000002010 - 0x0000000000002034 is .eh_frame_hdr
        0x0000000000002038 - 0x00000000000020b4 is .eh_frame
        0x0000000000003dd0 - 0x0000000000003dd8 is .init_array
        0x0000000000003dd8 - 0x0000000000003de0 is .fini_array
        0x0000000000003de0 - 0x0000000000003fc0 is .dynamic
        0x0000000000003fc0 - 0x0000000000003fe8 is .got
        0x0000000000003fe8 - 0x0000000000004010 is .got.plt
        0x0000000000004010 - 0x0000000000004020 is .data
        0x0000000000004020 - 0x0000000000004028 is .bss
```

Then, we find the start address by doing `set stop-on-solib-events 1`, running the program, and doing `info proc map`.

```
gef➤  set stop-on-solib-events 1
gef➤  r
Starting program: /home/raj/getuid
Stopped due to shared library event (no libraries added or removed)
gef➤  info proc map
process 59703
Mapped address spaces:

          Start Addr           End Addr       Size     Offset  Perms  objfile
      0x555555554000     0x555555555000     0x1000        0x0  r--p   /home/raj/getuid
      0x555555555000     0x555555556000     0x1000     0x1000  r-xp   /home/raj/getuid
      0x555555556000     0x555555557000     0x1000     0x2000  r--p   /home/raj/getuid
      0x555555557000     0x555555559000     0x2000     0x2000  rw-p   /home/raj/getuid
      0x7ffff7fc4000     0x7ffff7fc8000     0x4000        0x0  r--p   [vvar]
      0x7ffff7fc8000     0x7ffff7fca000     0x2000        0x0  r-xp   [vdso]
      0x7ffff7fca000     0x7ffff7fcb000     0x1000        0x0  r--p   /usr/lib/ld-linux-x86-64.so.2
      0x7ffff7fcb000     0x7ffff7ff1000    0x26000     0x1000  r-xp   /usr/lib/ld-linux-x86-64.so.2
      0x7ffff7ff1000     0x7ffff7ffb000     0xa000    0x27000  r--p   /usr/lib/ld-linux-x86-64.so.2
      0x7ffff7ffb000     0x7ffff7fff000     0x4000    0x31000  rw-p   /usr/lib/ld-linux-x86-64.so.2
      0x7ffffffdd000     0x7ffffffff000    0x22000        0x0  rw-p   [stack]
  0xffffffffff600000 0xffffffffff601000     0x1000        0x0  --xp   [vsyscall]
```

So, we find the two crucial pieces of information that we need: the entry point offset `0x1050` and the start address `0x555555554000`. We set a breakpoint at their sum, and then continue twice to break at the entry point.

```
gef➤  b *(0x555555554000 + 0x1050)
Breakpoint 1 at 0x555555555050
gef➤  c
Continuing.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/usr/lib/libthread_db.so.1".
Stopped due to shared library event:
  Inferior loaded /usr/lib/libc.so.6
gef➤  c
Continuing.

Breakpoint 1, 0x0000555555555050 in ?? ()
```

We are now at the entry point, but not yet at `main()`. So, we keep pushing and examine the next few instructions.

```
gef➤  x /15i $rip
=>  0x555555555050:      endbr64
    0x555555555054:      xor    ebp,ebp
    0x555555555056:      mov    r9,rdx
    0x555555555059:      pop    rsi
    0x55555555505a:      mov    rdx,rsp
    0x55555555505d:      and    rsp,0xfffffffffffffff0
    0x555555555061:      push   rax
    0x555555555062:      push   rsp
    0x555555555063:      xor    r8d,r8d
    0x555555555066:      xor    ecx,ecx
    0x555555555068:      lea    rdi,[rip+0xda]        # 0x555555555149
    0x55555555506f:      call   QWORD PTR [rip+0x2f4b]        # 0x555555557fc0
    0x555555555075:      hlt
    0x555555555076:      cs nop WORD PTR [rax+rax*1+0x0]
    0x555555555080:      lea    rdi,[rip+0x2f99]        # 0x555555558020
```

At instruction `0x55555555506f` we identify the call to `__libc_start_main`, which is located at the address loaded into register `rdi` in the previous instruction. So, we break at `0x5555555551f9` and continue execution.

```
gef➤  b *0x555555555149
Breakpoint 2 at 0x555555555149
gef➤  c
```

```
Continuing.

Breakpoint 2, 0x0000555555555149 in ?? ()
```

Examining the next few instructions, we see the familiar function prologue and epilogue, as well as markers for the functions in our program.

```
gef➤  x/15i $rip
=>  0x555555555149:      push   rbp
    0x55555555514a:      mov    rbp,rsp
    0x55555555514d:      call   0x555555555040 <geteuid@plt>
    0x555555555152:      mov    esi,eax
    0x555555555154:      lea    rax,[rip+0xea9]        # 0x555555556004
    0x55555555515b:      mov    rdi,rax
    0x55555555515e:      mov    eax,0x0
    0x555555555163:      call   0x555555555030 <printf@plt>
    0x555555555168:      mov    eax,0x0
    0x55555555516d:      pop    rbp
    0x55555555516e:      ret
    0x55555555516f:      add    bl,dh
    0x555555555171:      nop    edx
    0x555555555174:      sub    rsp,0x8
    0x555555555178:      add    rsp,0x8
```

Terrific! We finally made it to our `main()` function, and can now begin *actually* reversing (although there isn't much to reverse in our toy UID example).