Hell of kernel Debugging

Clément Rouault

LSE, EPITA Systems Lab.
http://lse.epita.fr/

Table of Contents

why ?

- k instructor
- want to help students
- want to help myself (more important)

How ?

- A python module for GDB
- Provide command
- Provide hooks and automatic helpers

Tools

- GDB-Python
- zgeg

GdbPython

- Python API for GDB
- Create new commands / functions
- Create new types of breakpoint
- Execute any GDB command

zgeg

- LSE Project by Franck Michea
- Simple data mapping over structs
- First real use by a non-dev
- Discover the lack of documentation

First step

- Asm injection

Why asm injection ?

- Being able to work on simple gdb stub
- Going to do almost everything by ourself

Keeping it simple

- Use nasm to compile code
- Write code in process memory
- Call it: gdb `call` do everything

```
1 gdb.inferiors()[0].write_memory(int(base), code)
2 gdb.execute("call {0}()".format(base), to_string=True)
```

Does it work ? - QEMU

```
(gdb) exec_asm 32
32
>>mov eax, 11
>>ret
>>EOF
Exec result = 11

(gdb) exec_asm 32
32
>>mov eax,42
>>ret
>>EOF
Exec result = 42
```

Does it work ? - Bochs

```
(gdb) exec_asm 32
32
>>mov eax,11
>>ret
>>EOF
Exec result = 11

(gdb) exec_asm 32
32
>>mov eax,42
>>ret
>>EOF
Exec result = 11

(gdb) x/i 0x80000
   0x80000:     mov    eax,0x2a
```

Worst quick-patch ever!

- During debug: All exec in bochs do the same thing!
- call works on any other address
- `if bochs: base_call += 1`

Get the GDT

- get `GDTR.base`
- get `GDTR.limit`
- get the GDT and parse it

Get the GDTR limit

```
1  xor eax,eax
2  sub esp,6
3  sgdt [esp]
4  mov ax, [esp]
5  add esp, 6
6  ret
```

GDT with zgeg 1/2

```
1   class GDT_entry(zm.Struct):
2       limit_0 = zf.IntField(size=zf.IntField.Size.INT16)
3       base_0 = zf.IntField(size=zf.IntField.Size.INT16)
4       base_1 = zf.IntField(size=zf.IntField.Size.INT8)
5       typee = zf.BitField(4)
6       s = zf.BitField(1)
7       dpl = zf.BitField(2)
8       p = zf.BitField(1)
9       limit_1 = zf.BitField(4)
10      avl = zf.BitField(1)
11      l = zf.BitField(1)
12      d_b = zf.BitField(1)
13      g = zf.BitField(1)
14      base_2 = zf.IntField(size=zf.IntField.Size.INT8)
```

GDT with zgeg 2/2

```
1       def new_gdt(self, nb_entry, data):
2           class GDT(zm.Struct):
3               entries = zf.ArrayField(nb_entry,
4                       zf.SuperField(self.GDT_entry))
5           return GDT(zd.Data(data), 0)
```

segment selectors

- Control access to data, stack and code
- Index to a GDT's entry
- `cs`: code
- `ss`: stack
- `ds`: data

Fun with segments selectors

```
1  mov ebx, 0xdeadbabe
2  mov edx, [ebx]
3
4  mov ebp, 0xdeadbabde
5  mov edx, [ebp]
```

EBP != EBX

SS != DS

More fun with segments selectors

```
1  ; DS : Base=0 , limit = 0X1000, g=0
2  mov edx, 0x4000
3  mov edx, [edx]
```

- Bochs: GP (good)
- QEMU: No problem (bad)

Modification of segments

```
1  mov edx, 0x12345
2
3  mov [edx], 0x1000
4  mov [edx], 0x1000
```

- take 0x12345 = `ds.segment.limit_low`
- Same address: different destinations

Return and CS

- Imagine: 4 functions (`func1-4`)
- `func1` calls `func2`, etc
- `ret1` is the return addr into `func1`
- `func4` calls the `trigger` function

Return and CS

```
1  trigger:
2      movw  $secretf4 - ret4, GDT + 8 + 2
3      pop %eax
4      push $0x8
5      push %eax
6      retf
7
8  secret4:
9      push $str4
10     call print
11     movw  $secret3 - ret3, GDT + 8 + 2
12     pop %eax
13     pop %eax
14     push $0x8
15     push %eax
16     retf
```

Return and CS: GDB Fail

- Display code
- Breakpoint

More on CS

- Fully bufferized
- Refresh on `lret / ljmp/` ...
- Processor can work with "non-existant" GDT Entry

IDT

- Interrupt Descriptor Table
- More or less a big Array of function pointers

Debugging Exceptions

- Put breakpoints on the 32 first entries
- Get exception information from the stack (The good one)
- Info on the stack are: `EIP / CS / Eflags (ESP / SS)`
- Give the error cause to user

Breakpoints in GDBPython

```
1  class IDTBreakpoint(gdb.Breakpoint):
2
3      def __init__(self, nb_entry, entry):
4          location = "*" + str(entry.offset)
5          self.nb = nb_entry
6          self.entry = entry
7          super(IDTBreakpoint, self).__init__(location)
8
9      def stop(self):
10         return self.do_the_life()
```

Next steps

- IDT injection for early crash
  - no IDT
  - triple fault
  - reboot

Asm injection

- Previous asm injection does not work anymore
- Need to take care of `CS`
- Breakpoints have the same problem as previously

TODO: not now

- Pagination
- Handling task ?

Questions ?

- Questions ?

Contact

- Clement Rouault
- hakril@lse.epita.fr
- twitter: @hakril