libdebug

A Python library to debug binary executables, your own way

CYBER
CHALLENGE.IT

Gabriele Digregorio *@Io_no*
Roberto A. Bertolini *@MrIndeciso*
6 July 2024

What

- libdebug is:
  - A **dream**
  - A **Python library** to debug binary executables
  - The foundation of **your own** debugger
  - Your guardian angel during **dynamic analysis**

Why

- **Rev engineering** requires:
    - **scripted debugging**
    - **reproducible execution**
- GDB is **complex**, **slow**, and can **break** easily (e.g., corrupted ELF)
- Nobody really knows **gdbscript**
- **Python** is cool and everyone knows and uses Python

How: Debug a Process

```
from libdebug import debugger

d = debugger("chall")

d.run()

. . .

d.kill()
```

```
from libdebug import debugger

d = debugger()

d.attach(pid=12345)

. . .

d.kill()
```

How: Register Access

“If you can do it in assembly, you can do it in the script too”

- *MasterGuesser*

```
val = d.regs.rax
d.regs.rax += 1
d.regs.bh = 3
d.regs.rcx = d.regs.ebp


d.syscall_arg0 = 0x1
```

How: Memory Access

```
d.memory[d.regs.rsp]

d.memory[0x200:0x400]

d.memory["main+a1"]

d.memory["main":"main+8"]

d.memory["main", 29]

d.memory[0x1337, 10, "binary"]
```

```
d.memory[d.regs.rsp] = b"0"

d.memory[0x200:0x202] = b"1"

d.memory["main+a1"] = b"2"

d.memory["main":"main+2"] = b"3"

d.memory["main", 1] = b"4"

d.memory[0xa, 1, "binary"] = b"5"
```

How: Breakpoints and Watchpoints

```python
bp = d.breakpoint(0x374)

bp = d.breakpoint("main")

bp = d.breakpoint("main+2a")

bp = d.bp(0x738, hardware=True)

bp = d.bp("puts", file="libc")

bp = d.bp(0xa0, callback=brutino)
```

```python
wp = d.watchpoint(0x1337)

wp = d.watchpoint("main")

wp = d.watchpoint("main+2a")

wp = d.wp(0x1337, condition="rw")

wp = d.wp(0x1337, file="binary")

wp = d.wp(0xa, callback=watchino)
```

How: Other Features

- Multithreading
- Communication with the process
- Signal catching
- Syscall handling
- . . .
- More to come

```
d.threads[3].regs.rax = 0x1337


r = d.run()
r.sendlineafter(b"provola",
b"provolone")


d.catch_signal("SIGINT")


d.handle_syscall("nanosleep")
```

Challenge Time

libdebug

Thanks for your attention

libdebug

libdebug.org

io_no@libdebug.org

github.com/libdebug

mrindeciso@libdebug.org