**Write-Up: Debugging Memory Layout and Finding picoCTF Flag**

This write-up outlines the steps and key commands used to identify the flag from memory data in GDB. The goal was to understand how a value (0x2262c96b) is stored in memory and extract it based on its byte layout.

**1. Problem Context**

* A 32-bit constant, 0x2262c96b, is loaded into memory in the main function.
* The task is to inspect memory byte-by-byte and determine how this value is stored (using little-endian format).
* The flag format is picoCTF{0x<value\_in\_memory>}, where <value\_in\_memory> is derived from the byte order.

**2. Step-by-Step Solution**

**Step 1: Analyze Assembly Code**

* The instruction:

perl

Sao chépChỉnh sửa

movl $0x2262c96b, -0x4(%rbp)

This means:

* + The 32-bit value 0x2262c96b is stored in memory at address RBP - 4.

**Step 2: Identify the Memory Address**

* Using GDB, break at main and inspect registers:

shell

Sao chépChỉnh sửa

gdb> info registers

Find the value of the RBP register (base pointer). For example:

makefile

Sao chépChỉnh sửa

rbp = 0x7fffffffe220

* Calculate the memory address where the value is stored:

css

Sao chépChỉnh sửa

Address = RBP - 4 = 0x7fffffffe220 - 0x4 = 0x7fffffffe21c

**Step 3: Inspect Memory**

* Use the GDB x command to examine 4 bytes at the calculated address:

shell

Sao chépChỉnh sửa

gdb> x /4xb 0x7fffffffe21c

Output:

makefile

Sao chépChỉnh sửa

0x7fffffffe21c: 0x6b 0xc9 0x62 0x22

**Step 4: Interpret Byte Order**

* The value 0x2262c96b is stored in **little-endian** format, meaning:
  + **Least Significant Byte (LSB)** is stored first: 0x6b.
  + **Most Significant Byte (MSB)** is stored last: 0x22.
* Combine the bytes in reverse order:

Sao chépChỉnh sửa

0x6b 0xc9 0x62 0x22 → 0x6bc96222

**Step 5: Construct the Flag**

* The final flag is:

Sao chépChỉnh sửa

picoCTF{0x6bc96222}

**3. Key Commands**

1. **Break at main function**:

shell

Sao chépChỉnh sửa

gdb> break main

gdb> run

1. **Check register values**:

shell

Sao chépChỉnh sửa

gdb> info registers

1. **Inspect memory**:

css

Sao chépChỉnh sửa

gdb> x /4xb <address>

Replace <address> with the calculated memory address.

1. **Understand endianness**:
   * Little-endian: Bytes are stored from least significant to most significant.

**4. Key Concepts**

* **Endianness**:
  + **Little-endian**: LSB stored first, MSB stored last (used by x86/x86-64).
  + **Big-endian**: MSB stored first, LSB stored last.
* **GDB Commands**:
  + info registers: Displays current values of all registers.
  + x /Nxb <address>: Examines N bytes starting from <address>.
  + Arithmetic with registers: Subtract or add offsets to find memory addresses.
* **Memory Layout**:
  + When storing multi-byte data, understanding the system's endianness is crucial for interpreting values correctly.