Steps in the attack:
1. Control-flow hijack: overwriting a code pointer e.g., RA
2. Payload: (a) injected code
   (b) existing code (e.g., return-to-libc or ROP)
3. Bypass exploit mitigation techniques e.g., Canary
   (a) Partial overwrite
   (b) Double pointer attack

Exploit Goal: Increase attacker's capability (privilege)

Starting point: no login or ability to run code on victim
   (but is able to connect and send data to victim)
   \[ \text{escalate} \]
   \[ \text{run code (as normal user)} \]
   \[ \text{run code as root} \]
Partial overwrite:

- Brute-forcing canary requires $\sim 2^{32}$ attempts.
- "Smart" search
  - overwrite the first reachable byte

1. wait for a client to connect
2. accept connection
3. fork a child process
to handle connection
Double pointer overwrite: "Skip past the canary"

```c
void parseCmd(char *cmd) {
    char * arg = malloc(4096);
    char cmdnm[128];
    int i = 0;
    while (!isspace(*cmd)) cmdnm[i++] = *cmd ++;
    cmd ++; // skip space
    // copy (rest of) cmd into arg
```
4B → RA
4B → Saved BP
4B → canary
arg = malloc'd buffer addr

128 bytes = 32 words

cmd: 60 40 50 60
A new RA injected code
not space
Space
cmd arg

Lo addr
Hi addr