10/7 Agenda

- Syscall Security
- Process Creation

Syscalls

→ voluntarily invoked

→ user provided arguments

Types of parameters

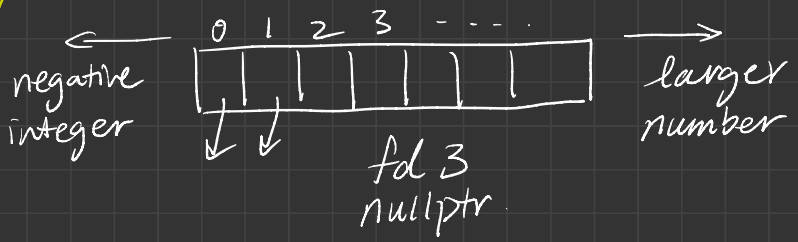
→ pointers (byte buffers, structs)

→ string (char *)

→ integers (file descriptors)

How would you attack these?

① integer [read(fd, buf, bytes)]



② String (char*)

→ invalid: no null terminator (part of the string at invalid address)

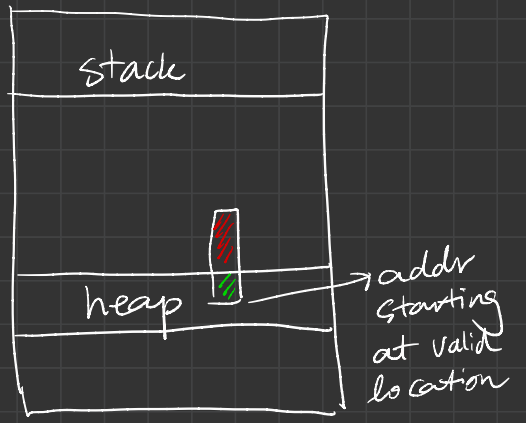
kernel address (possible if kernel is mapped into proc's address space)

- XK KERNBASE

```
44  // Fetch the nul-terminated string at addr from the current process.
45  // Doesn't actually copy the string - just sets *pp to point at it.
46  // Returns length of string, not including nul.
47  int
48  fetchstr(uint64_t addr, char **pp)
49  {
50    struct vregion *r;
51    struct vspace *v;
52    char *s, *ep;
53
54    v = &myproc()->vspace;
55    for (r = v->regions; r < &v->regions[NREGIONS]; r++) {
56      if (vregioncontains(r, addr, 0)) {
57        *pp = (char*)addr;
58        ep = (char *)VRTOP(r);
59        for(s = *pp; s < ep; s++) {
60          if(*s == 0)
61            return s - *pp;
62        }
63      }
64    }
65    return -1;
66  }
```

→ end of vregion

* checks if every byte is within the vregion.



☆ string can start valid but span over invalid area.

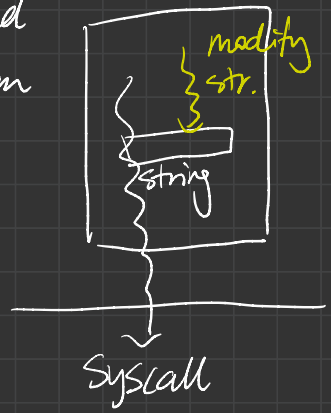
syscall.c

→ What may happen after kernel validated a string?

- string is still in user address space, can be modified by user.

to carry this out
① multithreaded process
② shared memory or memory map

☆ Kernel copies the validated string to prevent it from changing



③ pointers

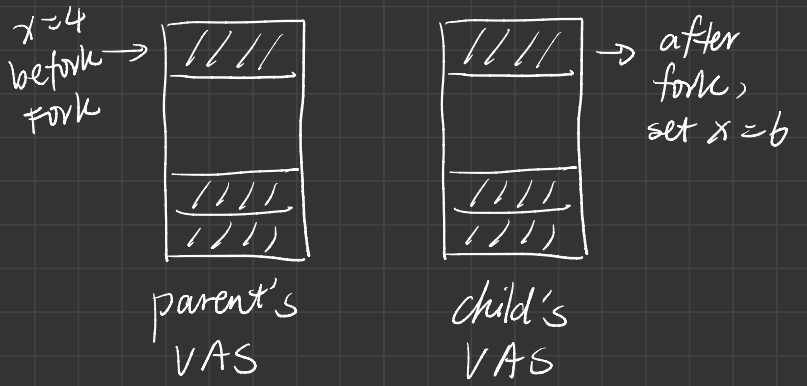
→ Similar attacks as strings but ptr argument requires a size.

→ can't pull the same trick (no moving null terminator)

↳ no changing size.

Process Creation

- Fork
  - → create a new process that's the exact copy of the calling process.



frozen in time

☆ after the copy, memory updates are private to the process.

→ What about CPU states?

sp : same address (though each in its own VAS)

pc : same address (both returning from fork)

rax : return value (pid for parent, 0 for child)

other registers stay the same.

OS Resources

PCB

```
79  // Per-process state
80  struct proc {
81    struct vspace vspace;          // Virtual address space descriptor
82    char* kstack;                  // Kernel stack
83    enum procstate state;          // Process state
84    int pid;                       // Process ID
85    struct proc *parent;           // Parent process
86    struct trap_frame *tf;         // Trap frame for current syscall
87    struct context *context;       // swtch() here to run process
88    void *chan;                    // If non-zero, sleeping on chan
89    int killed;                    // If non-zero, have been killed
90    char name[16];                 // Process name (debugging)
91  };
```

different instance, same content as its parent (same VAS same vspace)

must be different, why?

parent state?
- running

child state?
- ready

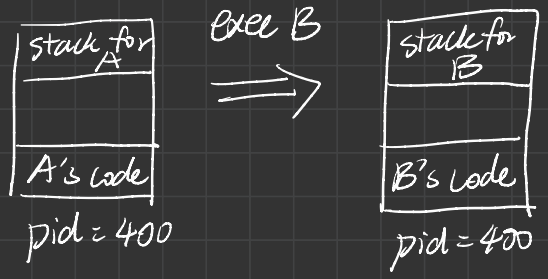
stores the CPU states, same as parent but different return value (rax)

Additional Resources
-> File Descriptors
- child inherits a copy

Exec

→ loads an executable into a process's address space.

→ doesn't create a new process



- VAS changes, vspace changes too
- trapframe changes to reflect the new execution stream (new pc & code to execute)

Fork & Exec

→ create a new process and execute a specific executable

fork

proc can change the environments (including open files)

exec

→ shell uses this to easily implement redirection

ls -l > output.txt