Computer and Internet architecture, relevant to threat models, security and privacy

Feb.7:

a) §1.5–§1.6.1 [threat models/modeling, e.g., attack trees, STRIDE]
   + Fig.10.10 (p300) of 10.5 [network stack]
   + §10.6 [networking and TCP/IP background]
   + Fig.11.9 (p329-330) of 11.6 [TCP 3-way handshake]
   The above are from *Tools and Jewels* (URLs are given in the online course page)

b) How the Internet works—network stack model/flow diagram:
   `https://people.scs.carleton.ca/~paulv/slide1-76-v8.1-KuroseRoss.x.pdf`

Notes for Feb.9 class:

OS, kernel, virtual memory spaces, how an OS works, how unauthorized software enters/runs

a) **User space vs. kernel space** — see Fig.7.4 (p.197) in §7.5:
`https://people.scs.carleton.ca/~paulv/toolsjewels/TJrev1/ch7-rev1.pdf`
b) **syscalls, shells, creating processes** — see pp.176-177 of §6.8:
`https://people.scs.carleton.ca/~paulv/toolsjewels/TJrev1/ch6-rev1.pdf`
c) Class notes related to the above: see below. Other students will made have their own notes.

The OS provides memory-resident, core software functionality to host application programs and other software on general-purpose computing platforms. The OS manages allocation and separation of computer resources across programs. For example, the login program (used to verify username + password), and software to access files (including filesystem drivers), are both parts of the OS.

A low-level subset of OS functionality is found in a component called the OS **kernel**. The kernel handles process management (CPU scheduling and program loading), memory management (allocating distinct regions of virtual memory to processes), low-level network communications, and access to hardware peripherals (i/o buses, physical devices, various filesystems).

Kernel memory is protected and distinct from so-called **user space memory** that is allocated to processes for programs. User space includes library software such as the standard C library (libC). On Unix systems, a program running with **root privileges** is still in user space (i.e., does not have direct access to kernel memory).

A set of kernel services (and thereby, kernel resources) are accessed by user space programs through **system calls**, which thus provides an API to kernel services. Most system calls are actually made through so-called **wrapper programs** in shared libraries. System calls result in a status change in a CPU hardware register, from **user mode** to **supervisor mode** (kernel mode). Supervisor mode enables privileges not available in user mode, such as access to special hardware registers and restricted CPU instructions (e.g., one halts the CPU). Supervisor mode software can also read and write to kernel memory, including special memory addresses used to communicate with hardware devices. In so-called *monolithic kernels* such as Unix, **device drivers** (which are necessary to interface with peripheral devices such as printers) are part of the kernel itself.

Word processors and productivity software (spreadsheets, presentation software) are application programs, distinct from the OS. Browsers are often installed when an OS is, and may be viewed as application software, but now contain much of the functionality of (the non-kernel part of) an OS.

Compilers, text editors, and other tools used in software development are often installed with an OS (especially on desktop development PCs). These are best viewed as separate from the OS, albeit considered "standard tools".

Neither **command shells** (programs that provide a command shell interface to users), nor common OS utility programs (such as popular Unix executables called from shell command lines) are part of the kernel, but they are commonly viewed as part of the OS.

Privilege level of programs. When you log in to your computer (e.g., regular user account), the programs (processes) you run have privileges based on the account's numeric UID. The processes will have access to files that this UID has access to (e.g., files created through/owned by that UID).

If you also know the **root** password for the machine (Unix/Linux), and switch to a root user account, any programs that you start up will then run with root privileges. Any attachment to an email that you click will run with those same privileges.

Any software program (including malware) that manages to start running on your machine, was somehow *launched* or *spawned* or started by an earlier program (process), and will typically run at the privilege level of that process. (On Unix systems, the *child* process inherits the UID of the *parent* process that created it.) The privileges of a typical user will allow access to all files that this user has access to, while **root privileges** will allow access to all user-space files on the machine.

If malware gets started "by" (on behalf of, or by exploiting) code that was running in the kernel itself, then the malware will run with kernel privileges (including access to kernel memory).

So there is a hierarchy of privileges, from lower to higher (malware perfers higher):
1. ordinary user (access to user space virtual memory of that user, and that user's files)
2. root user (this is still in user space, but allows access to all user space files)
3. kernel (access to both user space memory and kernel memory)

By exploting software vulnerabilities (beyond scope of COMP2109), an attacker can often move from lower to higher privileges (i.e., *escalate* their privileges).

Some ways for an attacker to get malware running on your machine, or gain control of it:

1) Get your credentials and log in as you, to your machine (or your bank account).

2) Trick you into installing malware yourself, perhaps using some form of **social engineering**.

   This may involve a **Trojan horse** program, i.e., having hidden or unexpected functionality. Another type of malware here is called a computer **virus** (these make use of regular software features, in perhaps unexpected ways). These programs will inherit the user's privileges.

3) Exploit a software vulnerability (flaw) to get malware installed/running. A common vector here is a computer **worm** that exploits a flaw in a network service program (**daemon**). Another is a **drive-by download** (silent malware that is installed upon visiting a shady or compromised web site).