OS Security Basics

CS642: Computer Security

Lecture 2

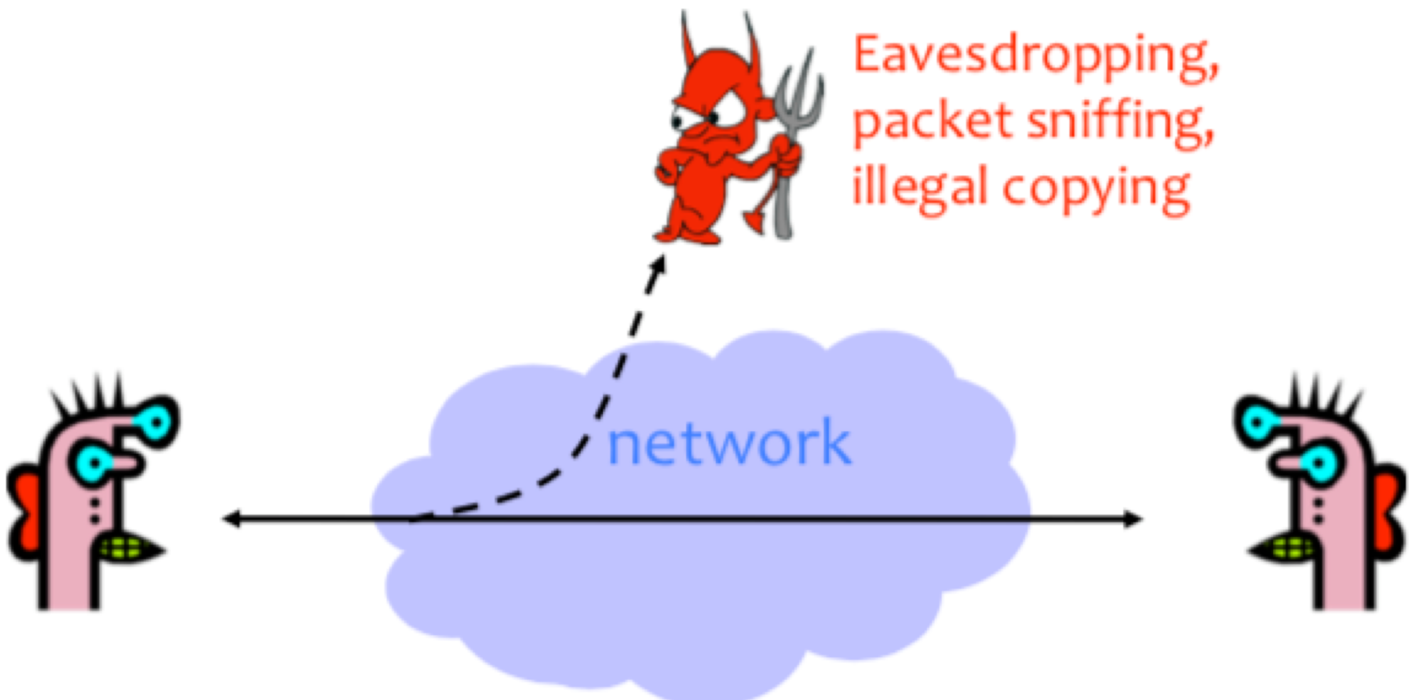
Operating System Security

Learning Goals

- Goals for OS security
- OS security mechanisms
- Password authentication
- Access control in Unix

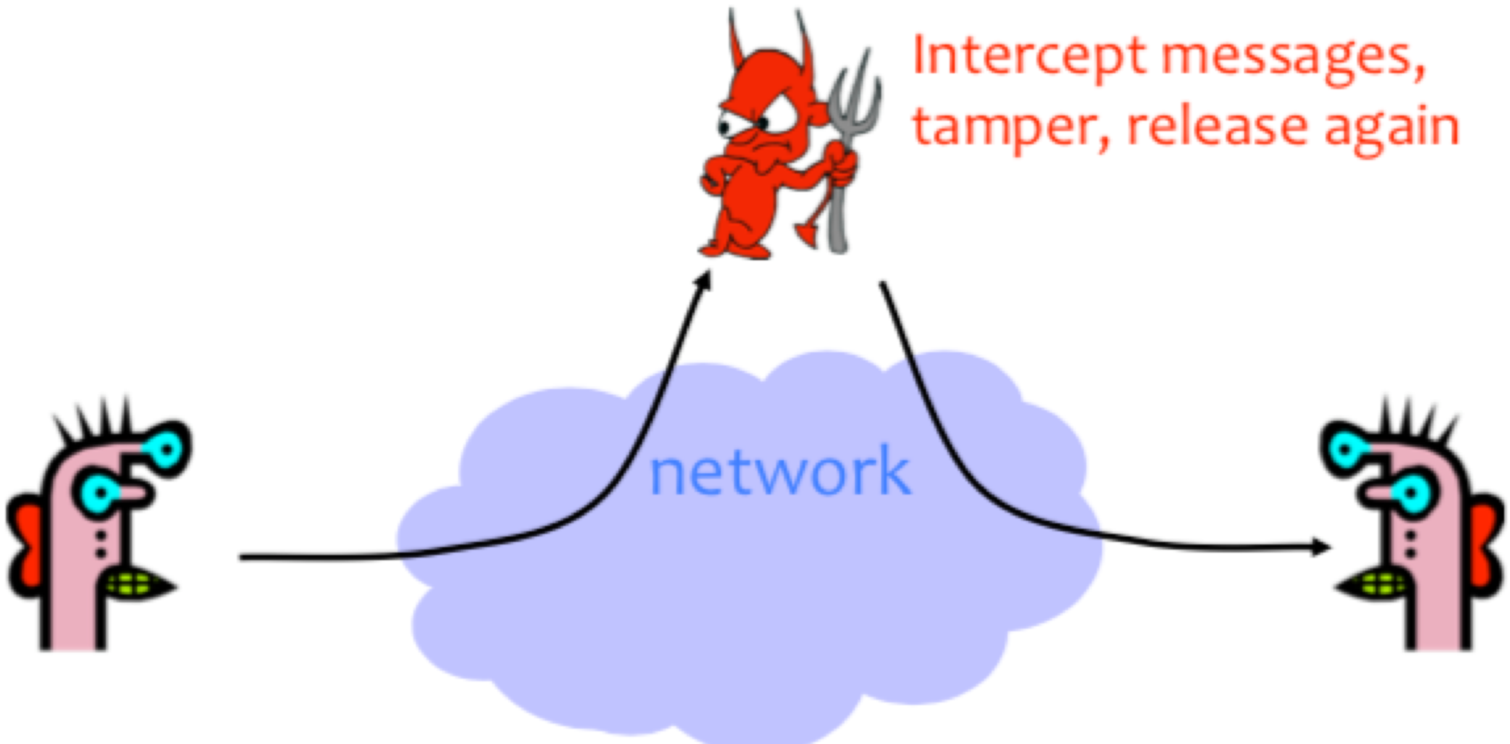
Confidentiality/privacy

- Confidentiality is concealment of information.



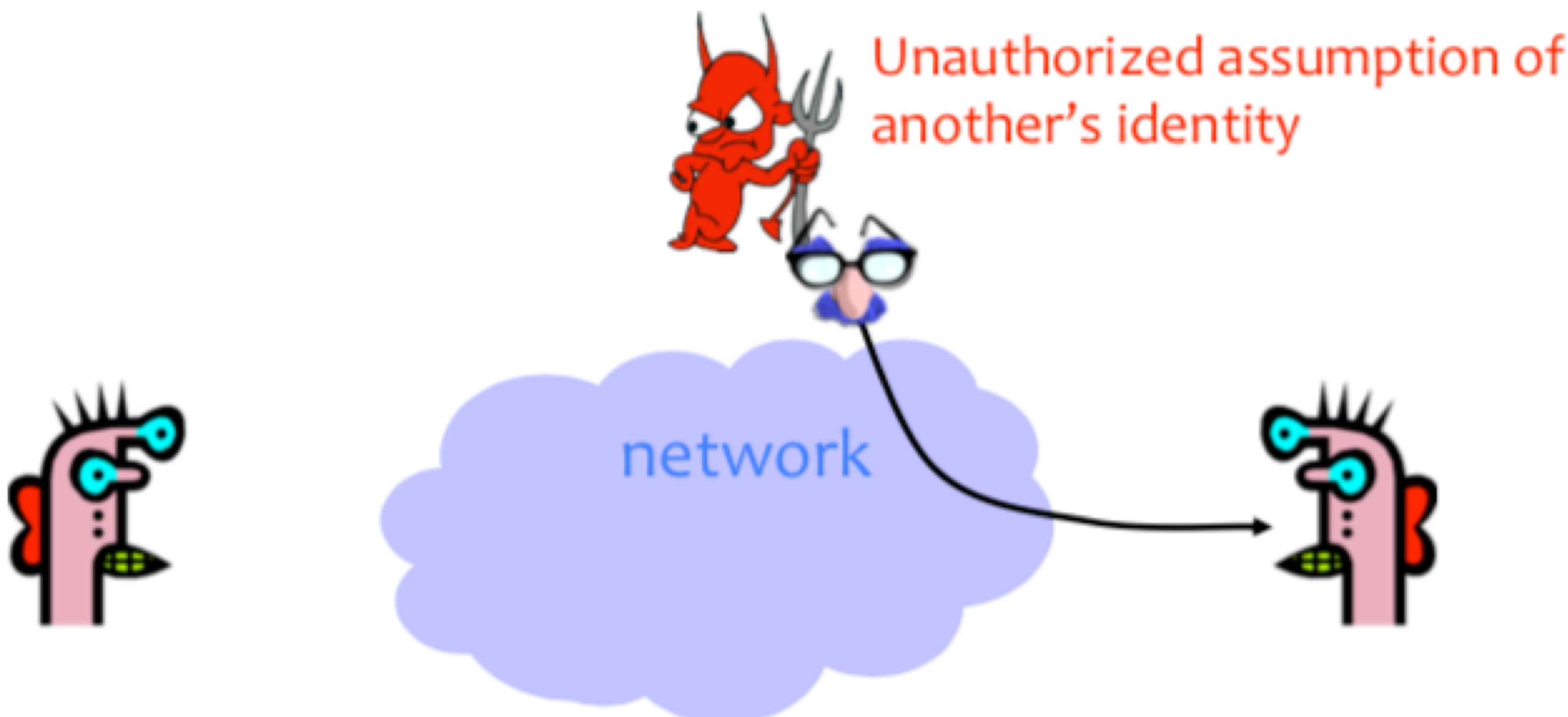
Integrity

- Integrity is prevention of unauthorized changes.



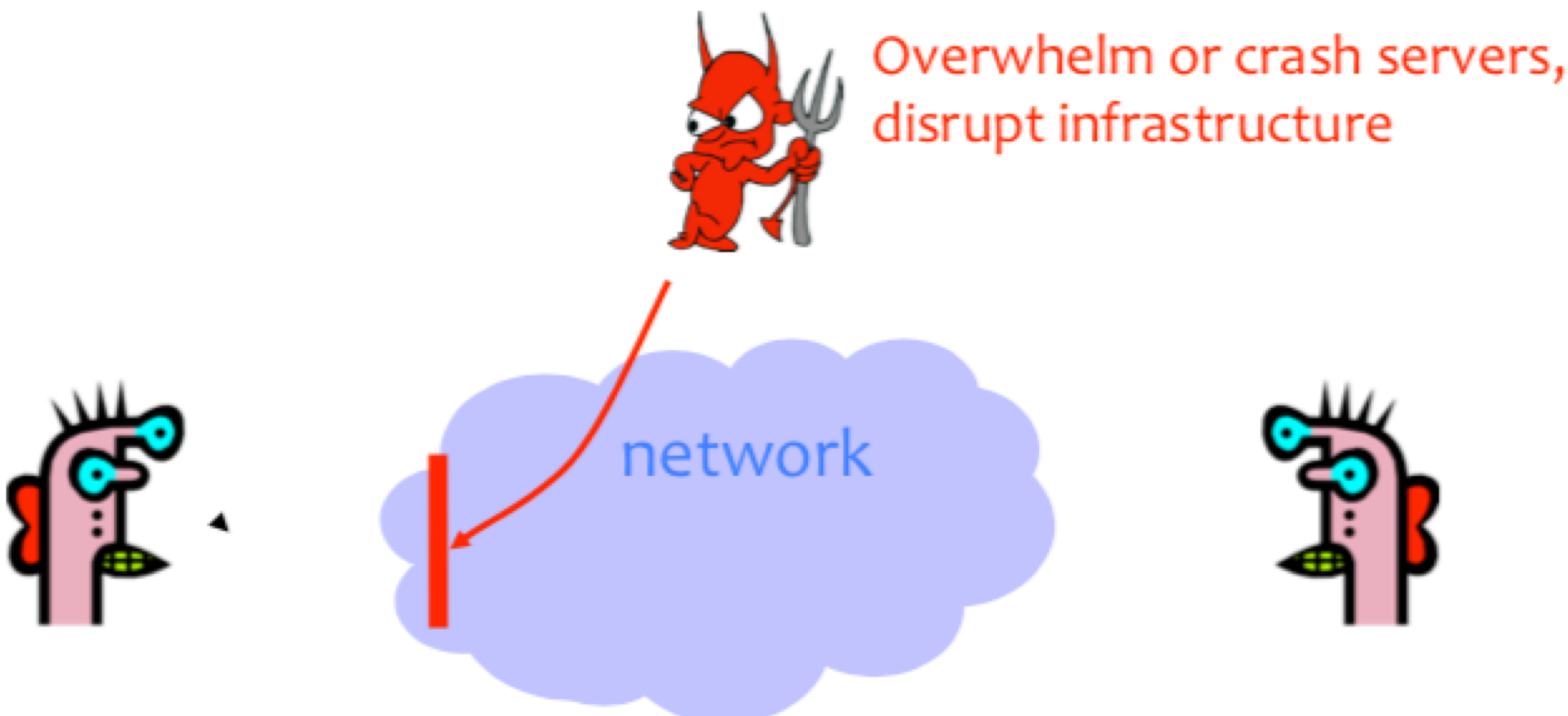
Authenticity

- Authenticity is knowing who you're talking to.



Availability

- Availability is ability to use information or resources.



Security Design Principles

- Saltzer &Schroeder, 1975, as part of Multics

1) Economy of mechanism
2) Fail-safe defaults
3) Complete mediation
4) Open design
5) Separation of privilege
6) Least privilege
7) Least common mechanism
8) Psychological acceptability

Economy of mechanism

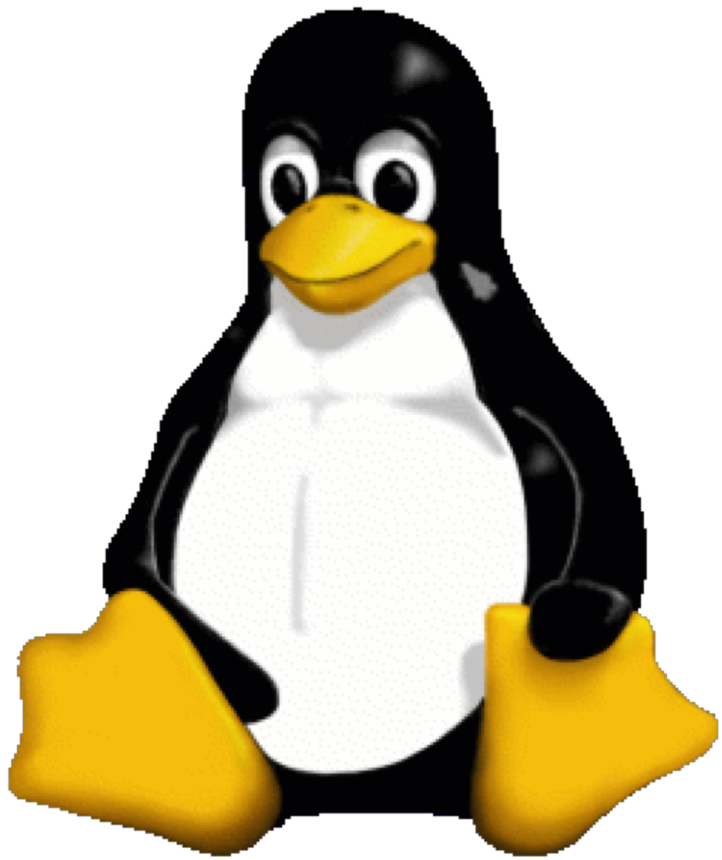
Fail-safe defaults

```
isAdmin = true;
try {
    codeWhichMayFail();
    isAdmin = isUserInRole( “Administrator” );
}
catch (Exception ex) {
    log.write( ex.toString() );
}
```

(Example from https://www.owasp.org/index.php/Secure_Coding_Principles)

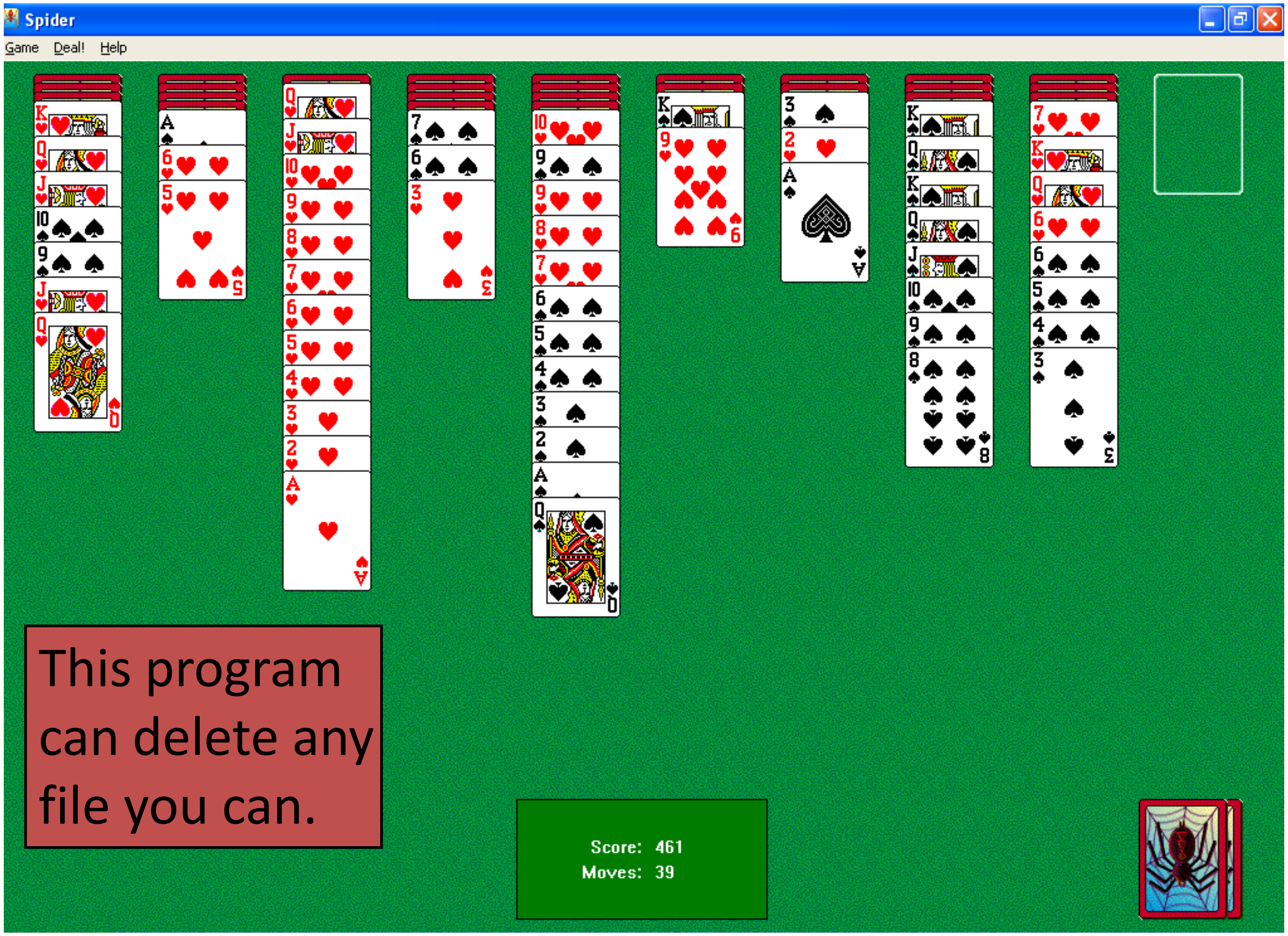
Complete mediation

Open design
(avoid “security by obscurity”)



Separation of privilege

Least privilege



(Courtesy of UCB CS161 slides)

Least common mechanism
(isolation)

Psychological acceptability
(consider human factors)

Principles from 1970's

- Do you think they are relevant today?
- A bit… abstract
- Recur over and over again

Limited Direct Execution

- Problem: how does the OS share the CPU between multiple processes, but remain in control
- Proposal: **privileged mode**– switching back and forth between user process and operating system
  - How maintain control, so a buggy/malicious process cannot take over?
- Solution: Limited direct execution
  - Let programs run directly on the CPU, but **not do everything**
  - To start a program, OS jumps to first instruction of program's main function (more or less)

Restricted Operations

- Problem: some operations shouldn't be available to programs
  - Writing data to a disk: how separate users from each other?
  - Decide which memory is accessible
- Solution: modes

Privileged instructions

- some instructions are restricted to the OS
  - known as protected or privileged instructions
- e.g., only the OS can:
  - directly access I/O devices (disks, network cards)
    - why?
  - manipulate memory state management
    - Which process can access which memory
  - halt instruction
    - why?

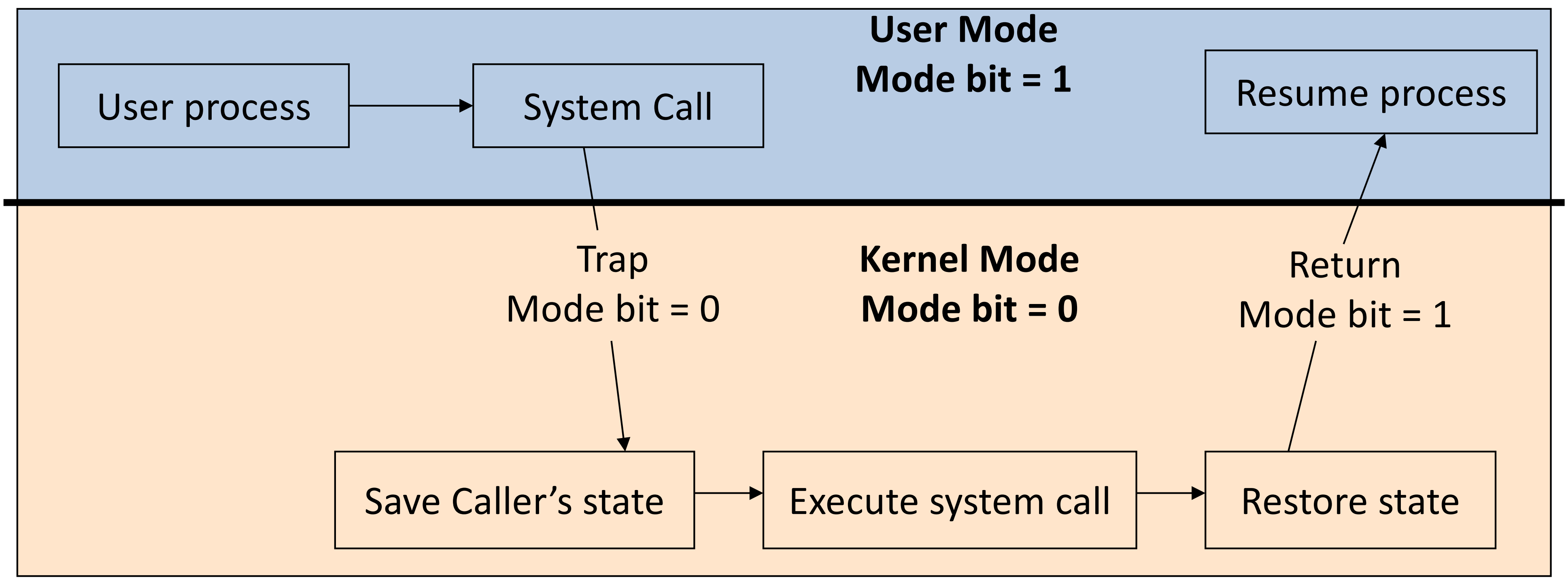
OS protection

- So how does the processor know if a protected instruction should be executed?
  - the architecture must support at least two modes of operation: kernel mode and user mode
  - mode is set by status bit in a protected processor register
    - user programs execute in user mode
    - OS executes in kernel mode (OS == kernel)
- Protected instructions can only be executed in the kernel mode
  - what happens if user mode executes a protected instruction?
- CPU enters protected mode automatically on interrupts/traps/exceptions
  - E.g. network packet arrives

Crossing protection boundaries

- So how do user programs do something privileged?
  - e.g., how can you write to a disk if you can’t do I/O instructions?
- User programs must call an OS procedure
  - OS defines a sequence of system calls
  - how does the user-mode to kernel-mode transition happen?
  - Why limit set of functions that can be called?
- There must be a system call operation, which:
  - causes an exception (throws a software interrupt), which vectors to a kernel handler
  - passes a parameter indicating which system call to invoke
  - saves caller’s state (regs, mode bit) so they can be restored
  - OS must verify caller’s parameters (e.g., pointers)
  - must be a way to return to user mode once done

A kernel crossing illustrated



System call details

- How does the kernel know which system call?
  - In a register
- Where are the parameters?
  - in a register
  - on the stack
  - in a memory block
- Limit set of kernel functions
  - System call table

```
<open>:        push   %ebx
<open+1>:      mov    0x10(%esp),%edx
<open+5>:      mov    0xc(%esp),%ecx
<open+9>:      mov    0x8(%esp),%ebx
<open+13>:     mov    $0x5,%eax
<open+18>:     syscall
<open+20>:     pop    %ebx
<open+21>:     cmp    $0xfffff001,%eax
<open+26>:     jae    0x2a189d <open+29>
<open+28>:     ret
```

```
# system call handler stub
ENTRY(system_call)
    pushl %eax                  # save orig_eax
    SAVE_ALL
    GET_THREAD_INFO(%ebp)
    cmpl $(nr_syscalls), %eax
    jae syscall_badsys
    syscall_call:
    call *sys_call_table(,%eax,4)
    movl %eax,EAX(%esp)         # store the return value
```

Validating parameters

- Sample calls:
  - fd = open(filename, O_RDONLY)
  - Result = read(fd, buffer, nbytes)
- What if filename is not null terminated?
  - Kernel overwrites local buffer & corrupts
- What if buffer points to a kernel address?
  - Kernel overwrites kernel structure with file data

Linux Validation

- Sample calls:
  - fd = open("/tmp/my_data", O_RDONLY)

```
long do_sys_open(int dfd,
    const char __user *filename,
    int flags, umode_t mode)
{
  struct filename *tmp;
  if (fd)
    return fd;
  tmp = getname(filename);
  …
}
```

```
getname_flags(const char __user *filename,
    int flags, int *empty)
{
    struct filename *result;
    char *kname;
    int len;
    len = strncpy_from_user(kname,
            filename,
            EMBEDDED_NAME_MAX);
    if (unlikely(len < 0)) {
        return ERR_PTR(len);
    }
...
```

Safe string copy

```
long strncpy_from_user(char *dst, const char __user *src, long count)
{
	unsigned long max_addr, src_addr;

	if (unlikely(count <= 0))
		return 0;

	max_addr = user_addr_max();
	src_addr = (unsigned long)src;
	if (likely(src_addr < max_addr)) {
		unsigned long max = max_addr - src_addr;
		long retval;

		kasan_check_write(dst, count);
		check_object_size(dst, count, false);
		user_access_begin();
		retval = do_strncpy_from_user(dst, src, count, max);
		user_access_end();
		return retval;
	}
	return -EFAULT;
}
EXPORT_SYMBOL(strncpy_from_user);
```

Fast string copy

```
ENTRY(copy_user_enhanced_fast_string)
        ASM_STAC
        cmpl $64,%edx
        jb .L_copy_short_string /* less then 64 bytes, avoid the costly 'rep' */
        movl %edx,%ecx
1:      rep
        movsb
        xorl %eax,%eax
        ASM_CLAC
        ret

        .section .fixup,"ax"
12:     movl %ecx,%edx          /* ecx is zerorest also */
        jmp copy_user_handle_tail
        .previous

        _ASM_EXTABLE(1b,12b)
```

Safe copy to usermode

```
static int copyout(void __user *to, const void *from, size_t n)
{
        if (access_ok(VERIFY_WRITE, to, n)) {
                kasan_check_read(from, n);
                n = raw_copy_to_user(to, from, n);
        }
        return n;
}

static __always_inline __must_check unsigned long
raw_copy_to_user(void __user *dst, const void *src, unsigned long size
)
{
    int ret = 0;
    __uaccess_begin();
    __put_user_asm(*(u32 *)src, (u32 __user *)dst,
                   ret, "l", "k", "ir", 4);
    __uaccess_end();
    return ret;
}
```

Authentication

- Establish the identity of user/machine by
  - Something you know (password, secret)
  - Something you have (credit card, smart card)
  - Something you are (retinal scan, fingerprint)
- In the case of an OS this is done during login
  - OS wants to know who the user is
- Passwords: secret known only to the subject
  - Simplest OS implementation keeps (login, password) pair
  - Authenticates user on login by checking the password
  - Try to make this scheme as secure as possible!
    - Display the password when being typed? (Windows, UNIX)

Online passwords attacks

- Online attacks: system used to verify the guesses
  - How someone broke into LBL

```
LBL> telnet elxsi
ELXSI AT LBL
LOGIN: root
PASSWORD: root
INCORRECT PASSWORD, TRY AGAIN
LOGIN: guest
PASSWORD: guest
INCORRECT PASSWORD, TRY AGAIN
LOGIN: uucp
PASSWORD: uucp
WELCOME TO THE ELXSI COMPUTER AT LBL
```

  - Thwart these attacks:
    - limit the number of guesses
    - better passwords

Why encrypt passwords?

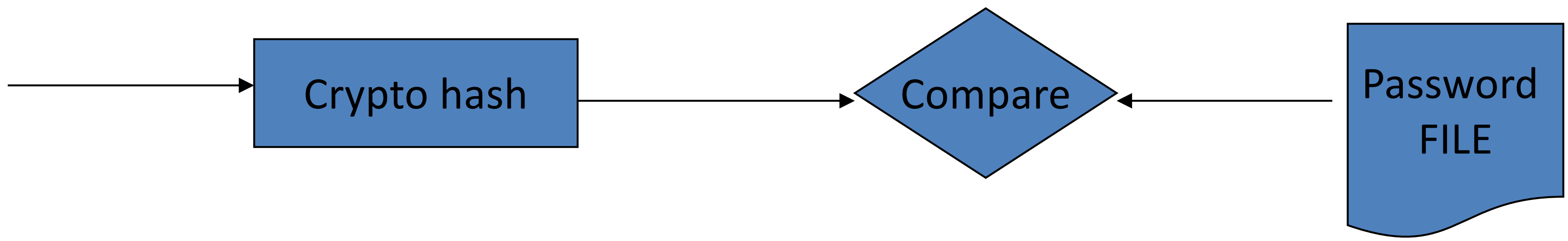
Example: TENEX page-fault caper

- TENEX, Secure system done at BBN in the ‘70s
- Tiger team dispatched to try to break a secure system.
- Passwords were 8 characters—machine too slow to do exhaustive search.
- So, align password on page boundary.
  - Time password check.
  - If process takes a page fault, you can tell how many of the characters were valid.
  - Turns 52^8 to a 52 * 8 problem.

f	aaaaaaa
fa	aaaaaa
fea	aaaaa

Better password storage

- store username/encrypted password in file
  - Properties of the one-way hash function *h*:
    - *h* is not invertible: *h(m)* easy to compute, $h^{-1}(m)$ difficult
    - It is hard to find *m* and m' s.t. *h(m) = h(m')*
  - Should use standard functions, such as SHA, MD5, etc.



Offline Attacks

- Previous scheme can be attacked: Dictionary Attack
  - Attacker builds dictionary of likely passwords offline
  - At leisure, builds hash of all the entries
  - Checks file to see if hash matches any entry in password file
  - There will be a match unless passwords are truly random
  - 20-30% of passwords in UNIX are variants of common words
    - Morris, Thompson 1979, Klein 1990, Kabay 1997
- Solutions:
  - Passwords should be made secure: increase complexity from 26^6 to 72^8
    - Length, case, digits, not from dictionary
  - Shadow files: move password file to /etc/shadow
    - This is accessible only to users with root permissions
  - Salt: store *(user name, salt, E(password+salt))*
    - Simple dictionary attack will not work. Search space is more.

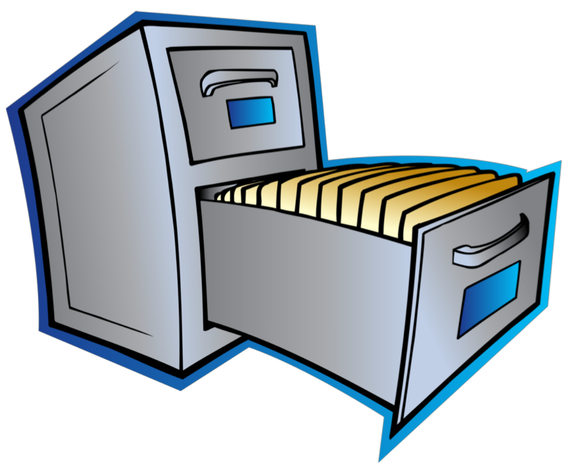
Salting Example

Bobbie, 4238, e(Dog4238)
Tony, 2918, e(6%%TaeFF2918)
Laura, 6902, e(Shakespeare6902)
Mark, 1694, e(XaB@Bwcz1694)
Deborah, 1092, e(LordByron,1092)

- If the hacker guesses Dog, he has to try Dog0001, …

Access controls

- Basic question: who gets access to what for what purpose?
  - Who = subjects. Users, programs
  - What = objects. Files, other programs, OS objects
  - Purpose = read, write, search, execute, change access



Access control matrix

Objects

Subjects

	file 1	**file 2**	**…**	**file n**
user 1	read, write	read, write, own		read
user 2				
…				
user m	append	read, execute		read,write, own

User i has permissions for file j as indicated in cell [i,j]

Due originally to Lampson in 1971

Access control in Real world

- Guard: something that checks for access
- Approach 1: Check a list of allowed people
- Approach 2: Check if person has a key or ticket

Access control implementation paradigms

	file 1	**file 2**	**…**	**file n**
user 1	read, write	read, write, own		read
user 2				
…				
user m	append	read, execute		read,write,own

(1) Access control lists

Column stored with file

File 1: user1: RW, user m:A

(2) Capabilities

Row stored for each user

User 1: file 1:RW, file 2:RWO, file n:R

How? Unforgeable **tickets** given to user

ACLs compared to Capabilities

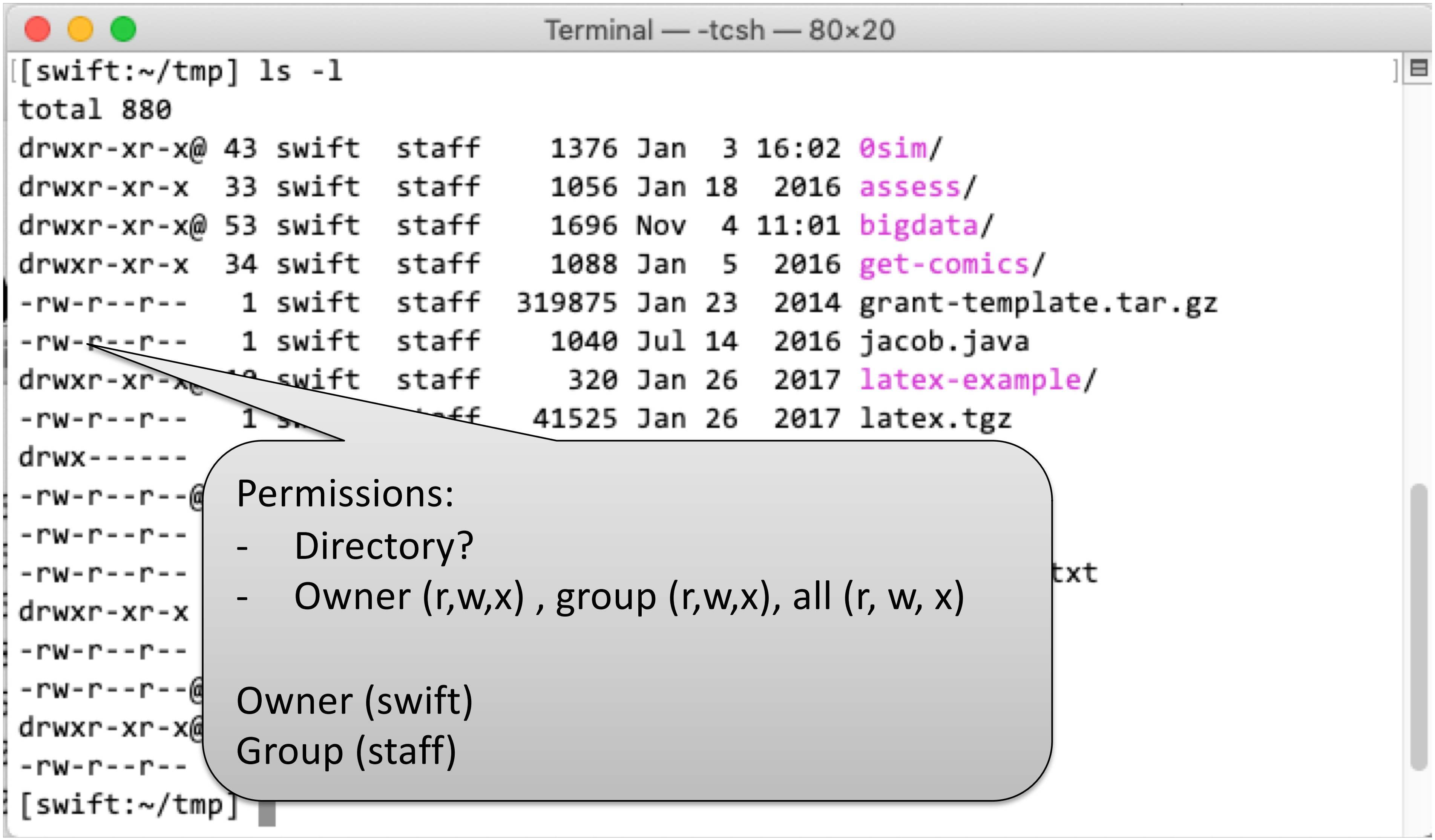
ACLs requires authenticating user	Token-based approach avoids need for auth
Processes must be given permissions	Tokens can be passed around
Reference monitor must protect permission setting	Reference monitor must manage tokens

UNIX-style file system

Terminal — -tcsh — 80×20

```
[swift:~/tmp] ls -l
total 880
drwxr-xr-x@ 43 swift  staff    1376 Jan  3 16:02 0sim/
drwxr-xr-x  33 swift  staff    1056 Jan 18  2016 assess/
drwxr-xr-x@ 53 swift  staff    1696 Nov  4 11:01 bigdata/
drwxr-xr-x  34 swift  staff    1088 Jan  5  2016 get-comics/
-rw-r--r--   1 swift  staff  319875 Jan 23  2014 grant-template.tar.gz
-rw-r--r--   1 swift  staff    1040 Jul 14  2016 jacob.java
drwxr-xr-x@ 10 swift  staff     320 Jan 26  2017 latex-example/
-rw-r--r--   1 swift  staff   41525 Jan 26  2017 latex.tgz
drwx------   3 swift  staff      96 Jun 28  2016 music/
-rw-r--r--@  1 swift  staff   51884 Jan 18  2017 nsys.tar.gz
-rw-r--r--   1 swift  staff    3973 Mar 13  2018 offices.txt
-rw-r--r--   1 swift  staff       0 Sep 20  2017 outputfile.txt
drwxr-xr-x  46 swift  staff    1472 Oct 15 11:19 p1/
-rw-r--r--   1 swift  staff    6404 Mar  3  2017 pg11.html
-rw-r--r--@  1 swift  staff    4906 Nov 20  2017 server.c
drwxr-xr-x@ 37 swift  staff    1184 Sep 25 16:57 ultron/
-rw-r--r--   1 swift  staff      27 Nov  6  2014 zits.pl
[swift:~/tmp]
```

UNIX-style file system ACLs



Permissions:
- Directory?
- Owner (r,w,x) , group (r,w,x), all (r, w, x)

Owner (swift)
Group (staff)

Unix File system capabilities

```
int fd = open("~/class/exam.txt");
read(fd,buffer,1024);
```

- File descriptors are capabilities
  - Allow access to file even if ACL changes
  - Can be passed to other processes/users

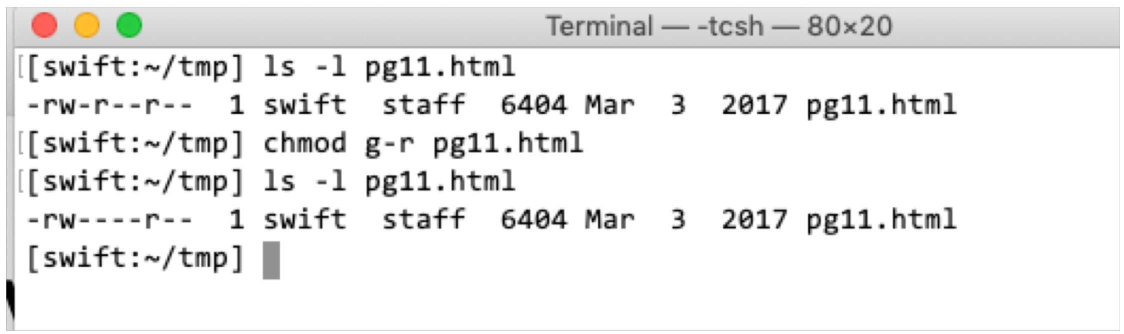
```
void send_fd(int socket, int fd) {
  struct   msghdr msg = {0};
  msg.msg_control = buf; msg.msg_controllen
       = sizeof buf;
  struct cmsghdr * cmsg =
     CMSG_FIRSTHDR(&msg);
  cmsg->cmsg_level = SOL_SOCKET;
  cmsg->cmsg_type = SCM_RIGHTS;
  cmsg->cmsg_len = CMSG_LEN(sizeof fd);
  *((int *) CMSG_DATA(cmsg)) = fd;
  msg.msg_controllen = cmsg->cmsg_len; //
  sendmsg(socket, &msg, 0);
}
```

Delegation

- Need to give a process, other user access
- With ACLs,
  - Option 1: run a new process, inherits user’s permissions
  - Option 2: change ACL to grant access to another user
    - What if it is not your file?
- With, pass around token

Revocation

- Take away access from user or process
- In ACL,
  - remove user from list

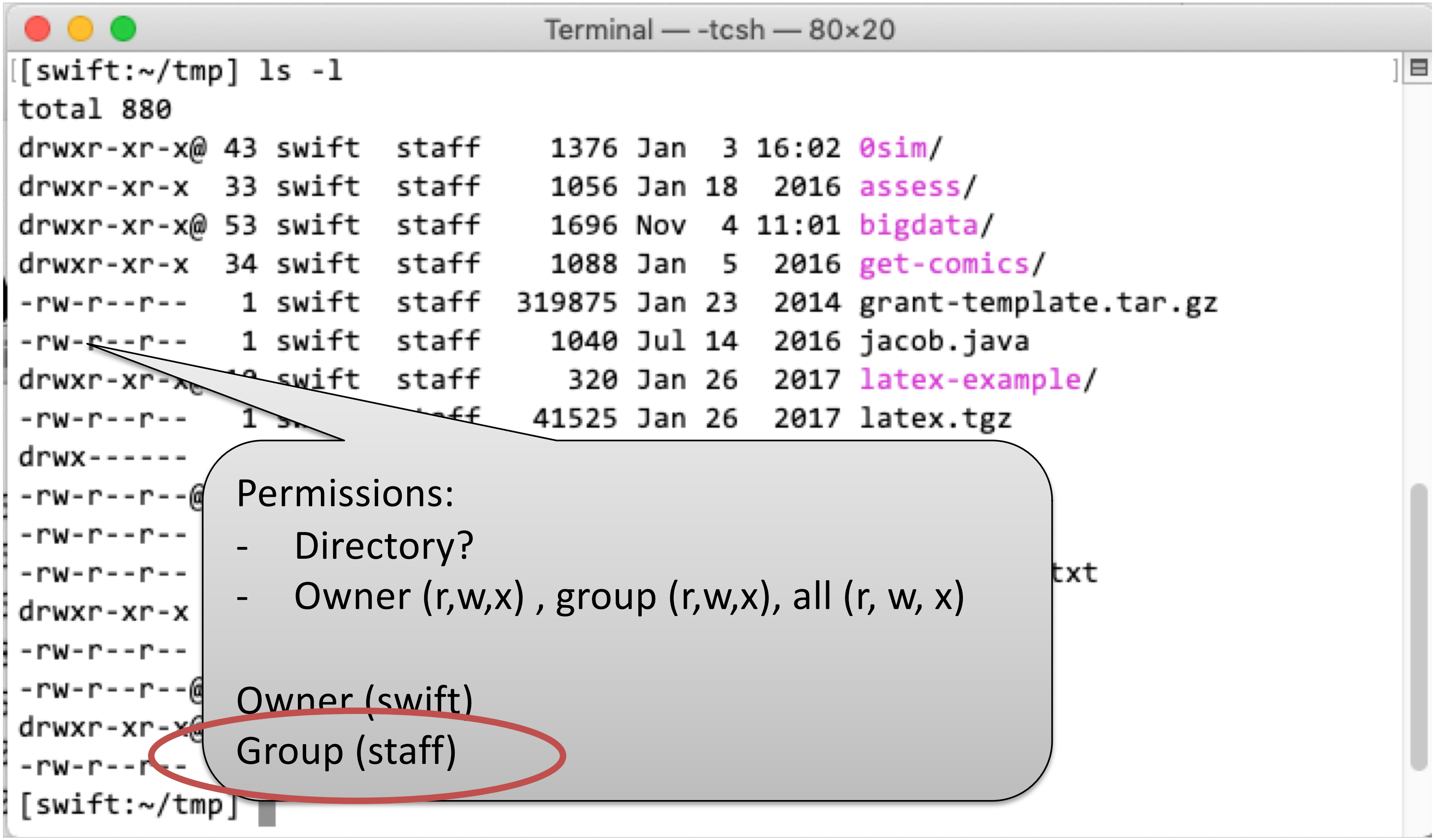


- In Capabilities
  - Option 1: track location of all capabilities, find and delete
  - Option 2: make capabilities indirect: point to an object that can be deleted, not to shared object directly

Who uses capabilities?

- Research operating systems:
  - Eros, ExoKernel, Barrelfish, Singularity
- Microkernels
  - Mach, Google Fuschia
- Mainstream OS (somewhere)
  - Use ACL when opening object, return a capability for faster access later
    - Linux/MacOS: file descriptors
    - Windows: handles

UNIX-style file system ACLs

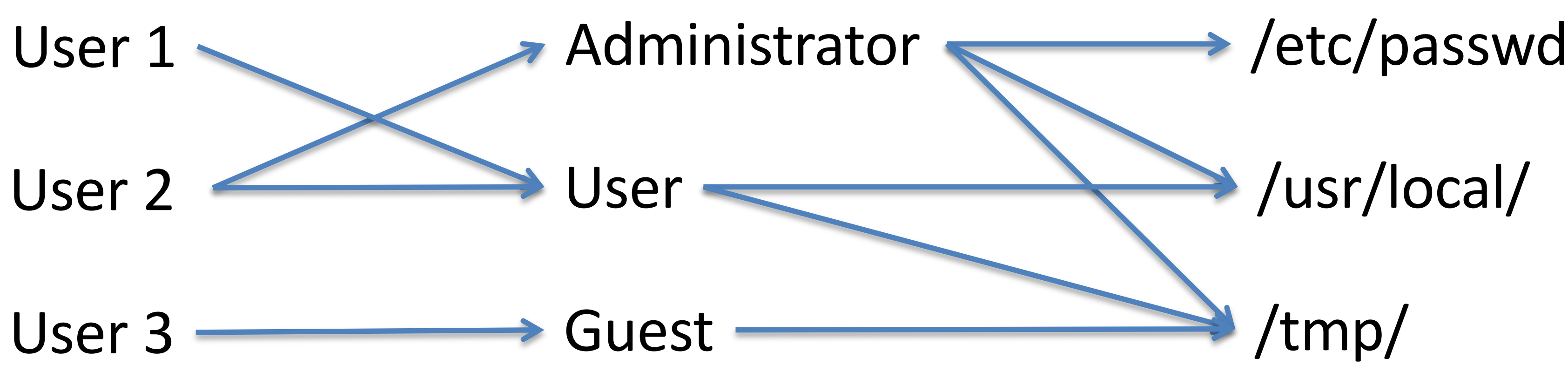


Roles (groups)

Group is a set of users

Administrator User Guest

Simplifies assignment of permissions at scale
Implemented as a UserID and multiple GroupIDs for each process



UNIX file permissions

- Owner, group
- Permissions set by owner / root
- Resolving permissions:
  - If user=owner, then owner privileges
    - Plus permissions to change ACL
  - If user in group, then group privileges
  - Otherwise, all privileges
- QUESTION: what do group members get if permission is –rw----rw?

Windows File permissions

- ACL = list of Access Control Entries (ACEs)
- Entries {allow,deny} permissions to a user or group
  - Multiple users or groups vs 1 for Linux
    - ACE1: UserA: alllow read
    - ACE2: UserB: allow/Write
  - Can explicitly deny access
    - ACE1: UserA: deny read
    - ACE2: Everyone: Read
  - Question: How implement
    Unix ACLs?