OS Security

Thierry Sans

Security is a wide topic

✓ CSCD27 Computer and Network Security covers

- Cryptography
- Network security
- System security
- Web security

(recap) Protection

File systems implement a protection system

- Who can access a file
- How they can access it

➡ A protection system dictates whether a given action performed by a given subject on a given object should be allowed

- You can read and/or write your files, but others cannot
- You can read "`/etc/motd`", but you cannot write it

DAC vs MAC

DAC - Discretionary Access Control
Users define their own policy on their own data

MAC - Mandatory Access Control

The administrator defines a system level policy to control the propagation of data between users

➡ DAC and MAC are not exclusive and can be used together

Discretionary Access Control

Unix protection on files

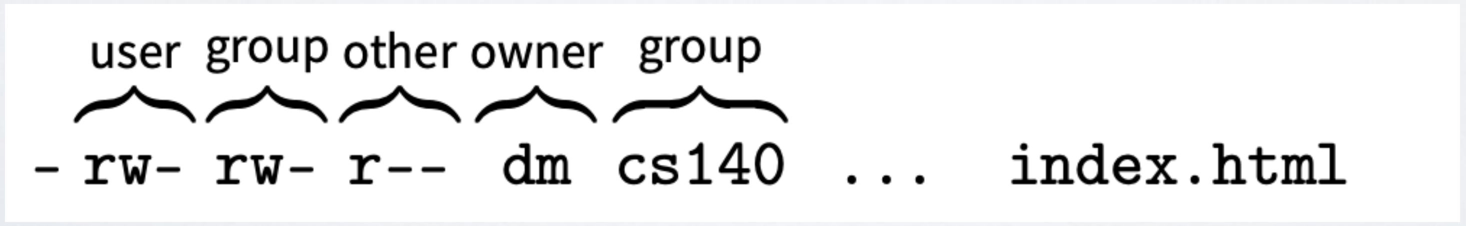
Each process has a User ID & one or more group IDs

System stores with each file

- User who owns the file and group file is in
- Permissions for user, any one in file group, and other

Shown by output of `"ls -l"` command

- Each group of three letters specifies a subset of **r**ead, **w**rite, and **e**xecute permissions
- User permissions apply to processes with same user ID
- Else, group permissions apply to processes in same group
- Else, other permissions apply



```
- rw- rw- r--  dm  cs140  ...   index.html
```

Unix protection on directories

Directories have permission bits, too

- Need write permission on a directory to create or delete a file
- Execute permission means ability to use pathnames in the directory, separate from read permission which allows listing

Special user root (UID 0) has all privileges

- e.g. read/write any file, change owners of files
- Required for administration (backup, creating new users, etc.)

For instance `drwxr-xr-x 56 root wheel 4096 Apr 4 10:08 /etc`

- Directory writable only by root, readable by everyone
- Means non-root users cannot directly delete files in `/etc`

Unix permissions on non-files

Many devices show up in file system
e.g. `/dev/tty1` permissions just like for files

Other access controls not represented in file system
e.g. must usually be root to do the following

- Bind any TCP or UDP port number less than 1024
- Change the current process's user or group ID
- Mount or unmount most file systems
- Create device nodes (such as `/dev/tty1`) in the file system
- Change the owner of a file
- Set the time-of-day clock; halt or reboot machine

Example - login run as root

Unix users typically stored in files in `/etc` files `passwd`, `group`, and often `shadow` or `master.passwd`

For each user, files contain

- Textual username (e.g., "`dm`", or "`root`")
- Numeric user ID, and group ID(s)
- One-way hash of user's password: {`salt`; `H(salt; passwd)`}
- Other information, such as user's full name, login shell, etc.

For instance `/usr/bin/login` runs as root

- Reads username & password from terminal
- Looks up username in `/etc/passwd`, etc.
- Computes `H(salt; typed password)` & checks that it matches
- If matches, sets group ID & user ID corresponding to username
- Execute user's shell with `execve` system call

Setuid

Some legitimate actions require more privileges than UID
e.g. how users change their passwords stored in root-owned `/etc/passwd` and `/etc/shadow` files?

➡ Solution - `setuid` and `setgid` programs

- Run with privileges of file's owner or group
- Each process has real and effective UID/GID
- Real is user who launched `setuid` program
- Effective is owner/group of file, used in access checks

Shown as "s" in file listings

```
-rws--x--x 1 root root 52528 Oct 29 08:54 /bin/passwd
```

- Obviously need to own file to set the `setuid` bit
- Need to own file and be in group to set `setgid` bit

Setuid

Examples

- `passwd` – changes user's password
- `su` – acquire new user ID (given correct password)
- `sudo` – run one command as root
- `ping` (historically) – uses raw IP sockets to send/receive ICMP

Have to be very careful when writing `setuid` code

- Attackers can run `setuid` programs any time (no need to wait for root to run a vulnerable job)
- Attacker controls many aspects of program's environment

➡ You will write such attack in CSCD27

Linux capabilities

Linux subdivides root's privileges into 40 capabilities, e.g.

- `cap_net_admin` – configure network interfaces (IP address, etc.)
- `cap_net_raw` – use raw sockets (bypassing UDP/TCP)
- `cap_sys_boot` – reboot
- `cap_sys_time` – adjust system clock

For instance ping needs raw network access, not ability to delete all files

```
$ ls -al /usr/bin/ping
-rwxr-xr-x 1 root root 61168 Nov 15 23:57 /usr/bin/ping
$ getcap /usr/bin/ping
/usr/bin/ping = cap_net_raw+ep
```

See also: `getcap(8)`, `setcap(8)`, `capsh(1)`

Other permissions

When can process A send a signal to process B with kill?

- Allow if sender and receiver have same effective UID
- But need ability to kill processes you launch even if `setsuid`, so allow if real UIDs match, as well

Debugger system call `ptrace` - lets one process modify another's memory

- `setuid` gives a program more privilege than invoking user so do not let a process ptrace a more privileged process e.g. require sender to match real & effective UID of target
- Also disable/ignore `setuid` if ptraced target calls exec
- Exception - root can ptrace anyone

Unix security hole

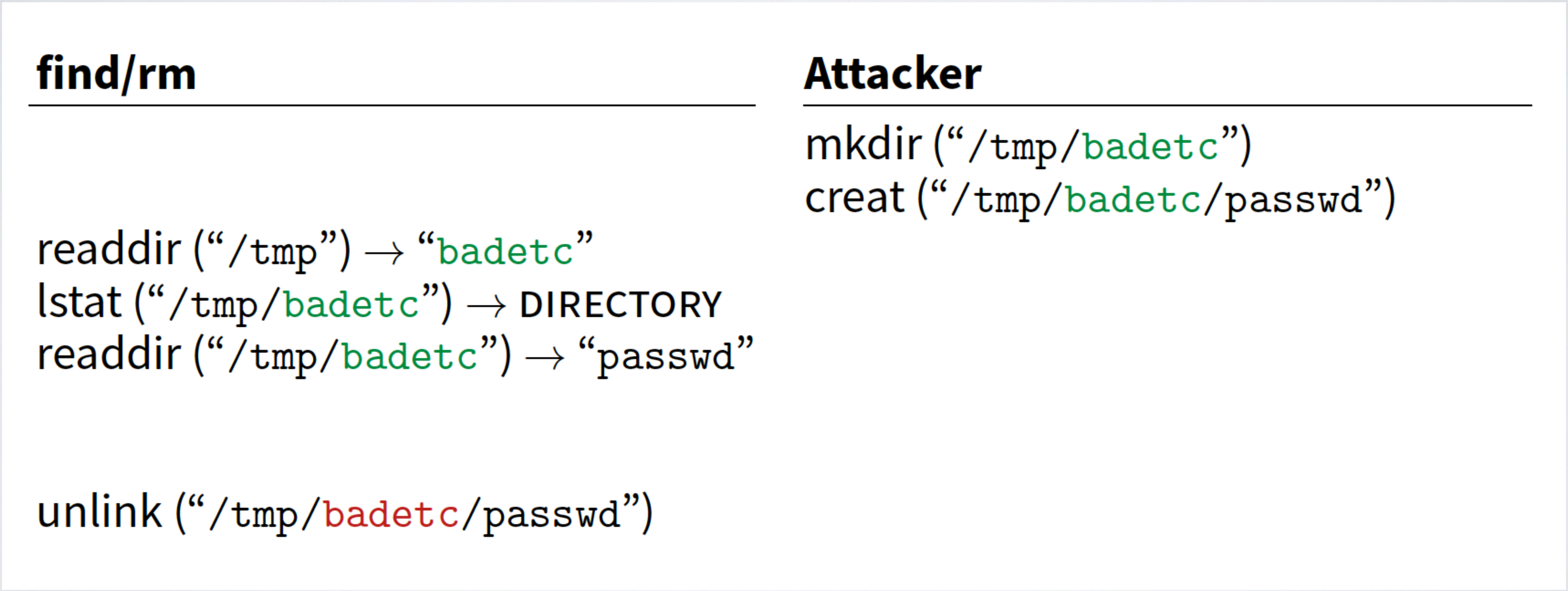
➡ Even without root (or setuid)
attackers can trick root owned processes into doing things

Example - clear unused files in /tmp every night

```
$ find /tmp -atime +3 -exec rm -f -- {} \;
```

- `find` identifies files not accessed in 3 days
- rm -f -- path deletes file path

Let us look at the system calls



find/rm	Attacker
	mkdir (“/tmp/badetc”)
	creat (“/tmp/badetc/passwd”)
readdir (“/tmp”) $\rightarrow$ “badetc”	
lstat (“/tmp/badetc”) $\rightarrow$ DIRECTORY	
readdir (“/tmp/badetc”) $\rightarrow$ “passwd”	
unlink (“/tmp/badetc/passwd”)	

TOCTOU attack

find/rm	Attacker
	mkdir (“/tmp/badetc”)
	creat (“/tmp/badetc/passwd”)
readdir (“/tmp”) → “badetc”	
lstat (“/tmp/badetc”) → DIRECTORY	
readdir (“/tmp/badetc”) → “passwd”	
	rename (“/tmp/badetc” → “/tmp/x”)
	symlink (“/etc”, “/tmp/badetc”)
unlink (“/tmp/badetc/passwd”)	

Time-of-check-to-time-of-use (a.k.a TOCTOU) bug

- find checks that `/tmp/badetc` is not symlink
- but meaning of file name changes before it is used

Another example - xterm

`xterm` Provides a terminal window in X-windows - used to run with setuid root privileges

- Requires kernel pseudo-terminal (`pty`) device
- Required root privileges to change ownership of `pty` to user
- Also writes protected `utmp`/`wtmp` files to record users

Had feature to log terminal session to file

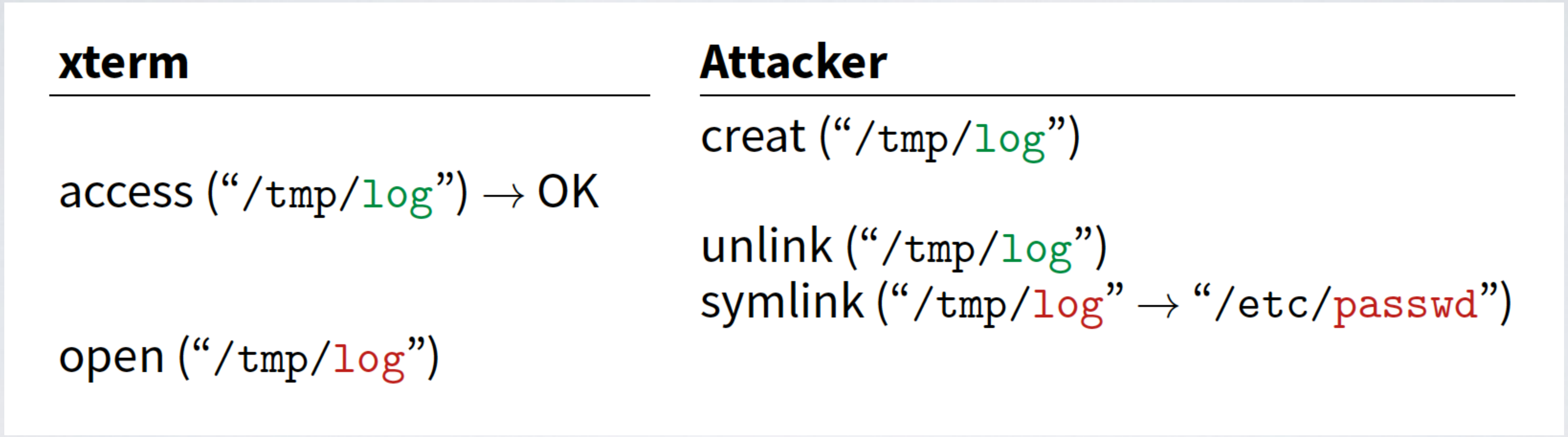
```
if (access (logfile, W_OK) < 0)
  return ERROR;

fd = open (logfile, O_CREAT|O_WRONLY|O_TRUNC, 0666);
/* ... */
```

➡ `xterm` is root, but should not log to file user cannot write

✓ `access` call does permission check with real, not effective UID

◉ .... but another TOCTOU bug

Another TOCTOU attack



xterm	**Attacker**
	creat (“/tmp/log”)
access (“/tmp/log”) $\to$ OK	
	unlink (“/tmp/log”)
	symlink (“/tmp/log” $\to$ “/etc/passwd”)
open (“/tmp/log”)	

Attacker changes `/tmp/log` between check and use

➡ `xterm` unwittingly overwrites `/etc/passwd`

- OpenBSD man page - "CAVEATS : access() is a potential security hole and should never be used."

Preventing TOCTOU

➡ Use new APIs that are relative to an opened directory file descriptor

- `openat`, `renameat`, `unlinkat`, `symlinkat`, `faccessat`
- `fchown`, `fchownat`, `fchmod`, `fchmodat`, `fstat`, `fstatat`
- `O_NOFOLLOW` flag to open avoids symbolic links in last component

◉ But can still have TOCTOU problems with hardlinks

✓ Alternative solution - lock resources, though most systems only lock files (and locks are typically advisory)

✓ Alternative solution - wrap groups of operations in OS transactions
e.g. Microsoft supports for transactions on Windows Vista and newer
`CreateTransaction`, `CommitTransaction`, `RollbackTransaction`

Mandatory Access Control

Mandatory Access Control

➡ **Mandatory access control (MAC)** can restrict propagation
e.g. security administrator may allow you to read but not disclose file

FYI, do not confuse:

- MAC - Mandatory Access Control (in OS security)
- MAC address - Medium Access Control (in networks)
- MAC - Message Authentication Code (in cryptography)
- MacOS (from Apple)

MAC Motivation

Prevent users from disclosing sensitive information (whether accidentally or maliciously) e.g. classified information requires such protection

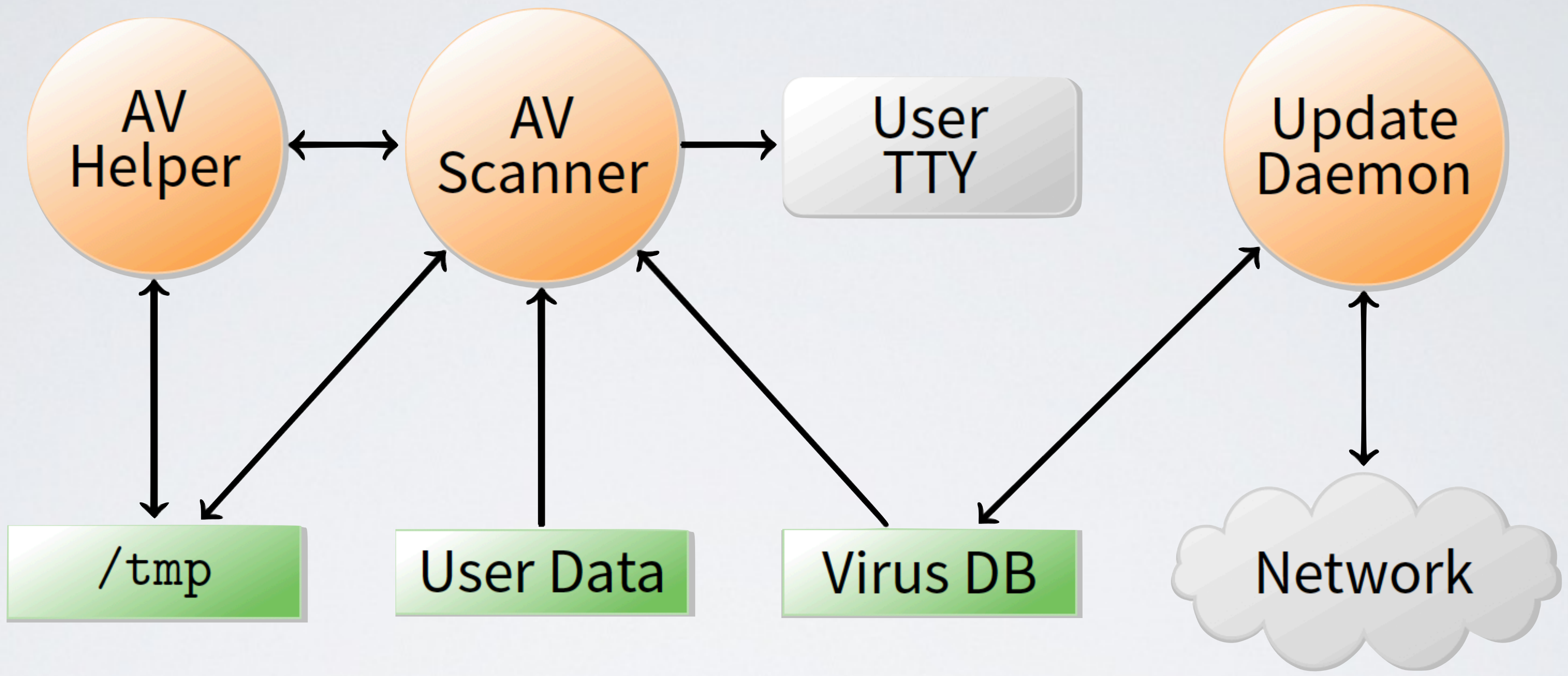
Prevent software from surreptitiously leaking data - seemingly innocuous software may steal secrets in the background (Trojan Horse)

Case study - Symantec AntiVirus 10

- Inherently required access to all of a user's files to scan them
- Contained a remote exploit (attacker could run arbitrary code)

➡ Can an OS protect private file contents under such circumstances?

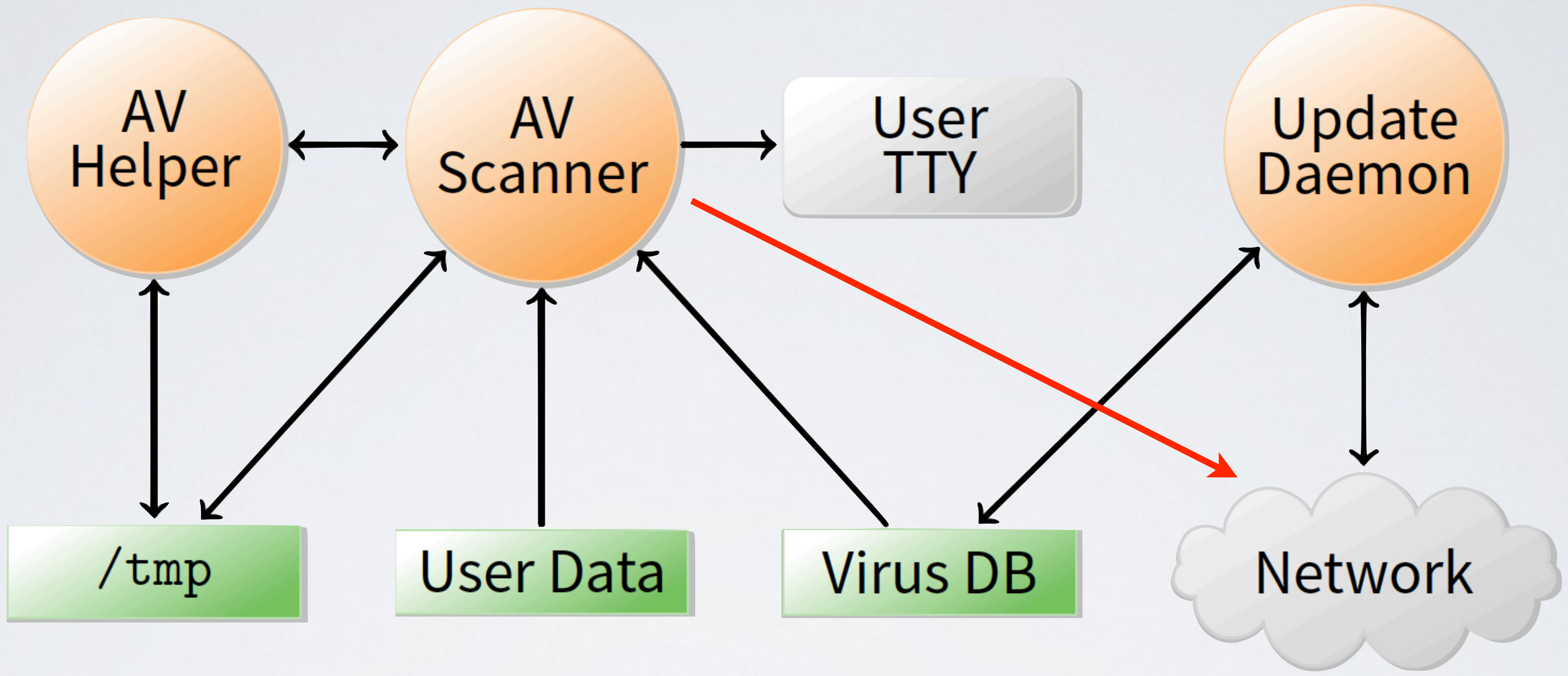
Example - anti-virus software



How can OS enforce security without trusting AV software ?

- Must not leak contents of your files to network
- Must not tamper with contents of your files

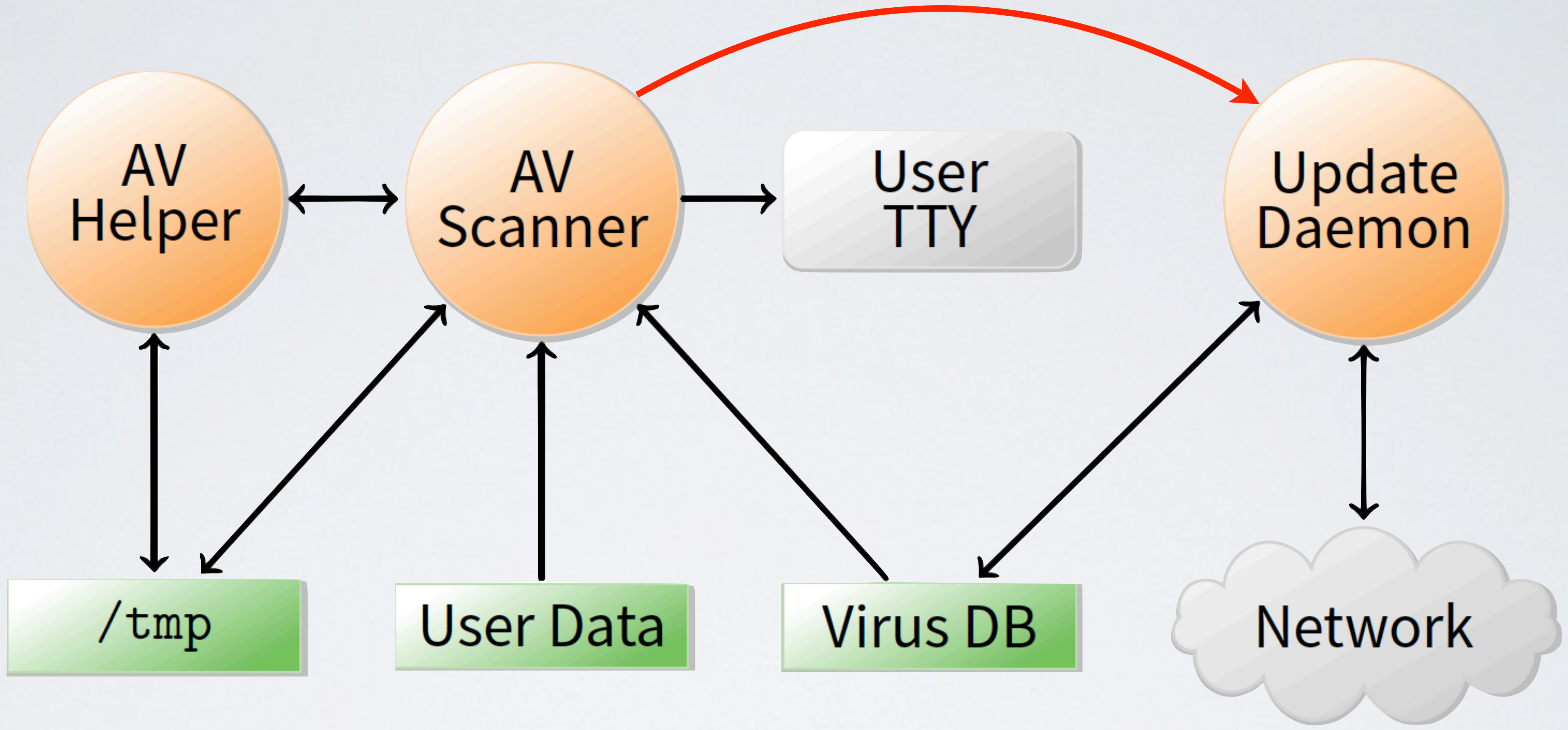
Example - anti-virus software



- Scanner can write your private data to network

➡ Prevent scanner from invoking any system call that might send a network messages?

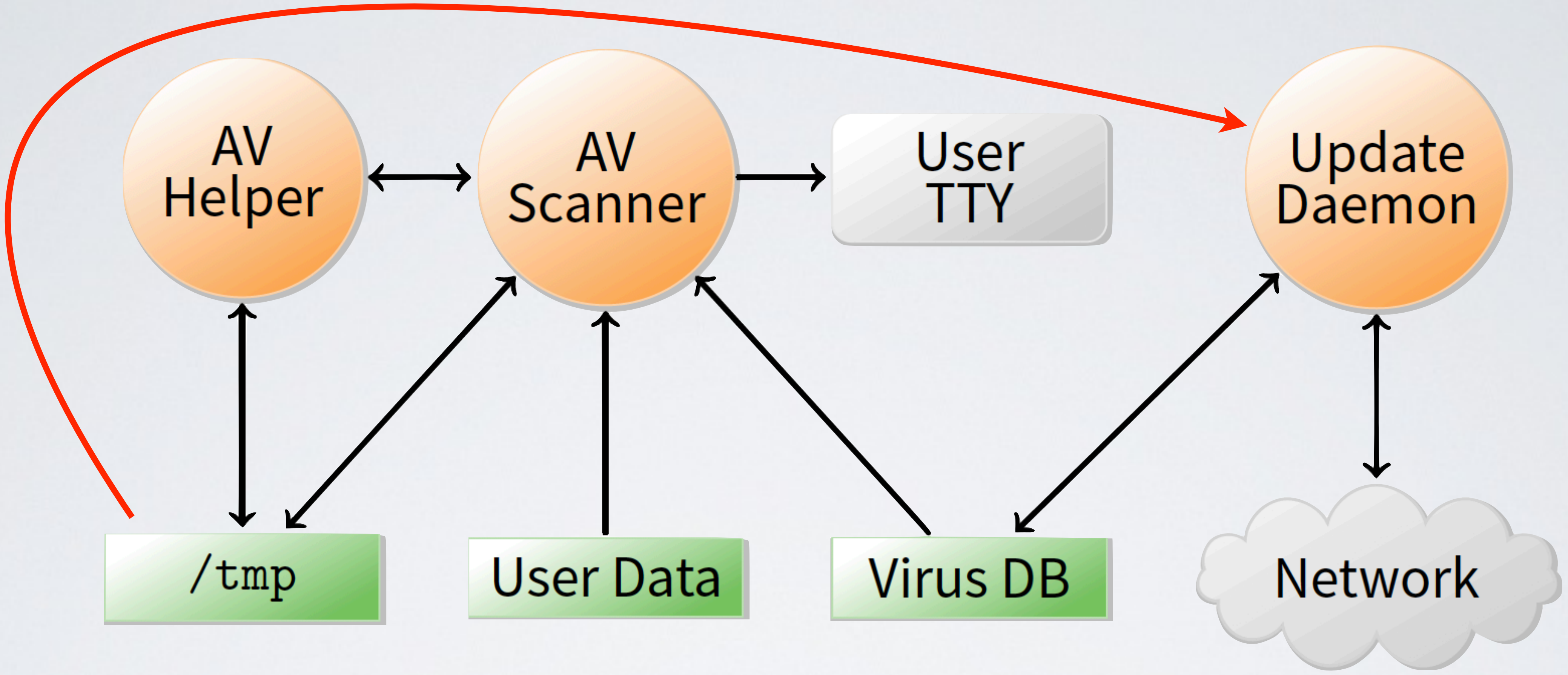
Example - anti-virus software



- Scanner can send private data to update daemon and update daemon sends data over network

➡ Block IPC & shared memory system calls in scanner?

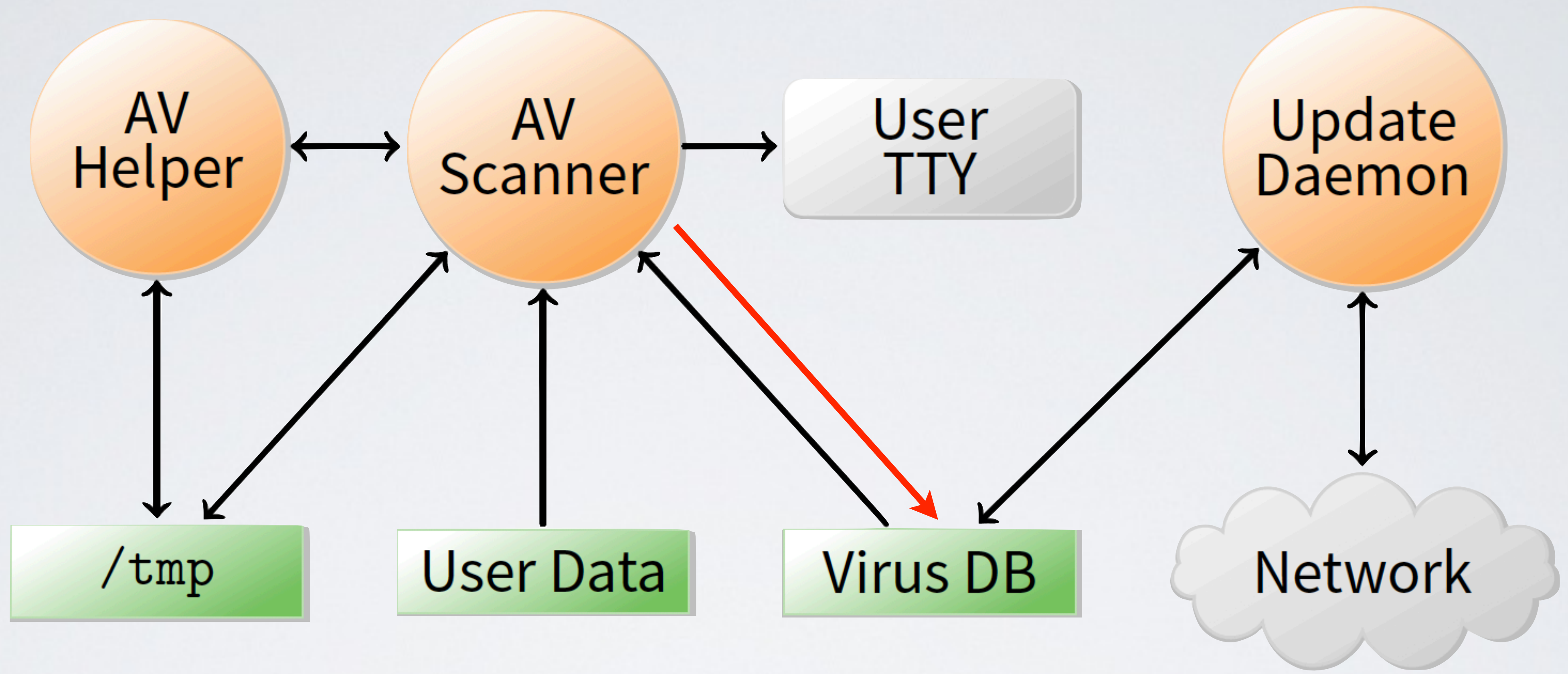
Example - anti-virus software



- Scanner can write data to world-readable file in `/tmp` and update daemon later reads and discloses file

➡ Prevent update daemon from using `/tmp`

Example - anti-virus software



- Scanner can acquire read locks on virus database to encode secret user data by locking various ranges of file and update daemon decodes data by detecting locks to discloses private data over the network

➡ Have trusted software copy virus DB for scanner

The list goes on

- Scanner can call setproctitle with user data
  Update daemon extracts data by running ps
- Scanner can bind particular TCP or UDP port numbers
  Sends no network traffic, but detectable by update daemon
- Scanner can relay data through another process (e.g ptrace) and exfiltrate data through yet another process (sendmail, httpd, portmap)
- Disclose data by modulating free disk space

Can we ever convince ourselves we have covered all possible communication channels (a.k.a covert channels)?

➡ Not without a more systematic approach to the problem

Mandatory Access Control Implementations

SELinux (Security Enhanced Linux)
enabled in major Linux distributions and Android

AppArmor
available in major Linux distributions

Smack (Simplified Mandatory Access Control Kernel)

Tomoyo Linux

Acknowledgments

Some of the course materials and projects are from

- David Mazière - teaching CS 140 at *Stanford*