KCRA TV, 2004.11.04:

“Students accused of hacking computer, changing grades

“Three high school students in Elk Grove have broken into a campus computer and changed grades, according to investigators. . . .

“A teacher noticed that she thought that a grade had been changed,’ said Sheldon High School principal Paula Duncan.

“The school district launched an immediate investigation, calling in the Sacramento Sheriff’s Department’s High-Tech Crimes Task Force. . . .
“The students were allegedly able to gain access to the information by using spyware software during school hours in the school’s library. Computer experts said they have not seen any evidence that personal information was stolen.

“The students have been removed from the school and face potential expulsion. The school district has hired security experts from Microsoft to make sure the computers will be secure from any future hackings.”
The file-rewriting problem

Contents of /etc/passwd:

root:*:0:0:Root:/root:/bin/csh

djb:*:1001:1001:
D. J. Bernstein,410 SEO,
312-413-9322:/home/djb:
/bin/csh

joe:*:1002:1002:Joseph Evil,
,312-867-5309:/home/joe:
/bin/bash

etc. One line per account.

Joe is allowed to change some of the information on the joe line.

There are setuid programs designed to do this: chfn, chsh, etc.
Suppose djb changes shell from /bin/csh to /bin/tcsh. How does chsh handle /etc/passwd?

open("/etc/passwd",O_RDONLY);
read(...); read(...); read(...);
read(...); etc.; close().

open("/etc/passwd",
     O_WRONLY|O_TRUNC);
write(...); write(...); write(...);
write(...); etc.; close().
0_TRUNC truncates /etc/passwd: it’s now 0 bytes long. The first write() puts (e.g.) 512 bytes onto the end of /etc/passwd. The second write() puts 512 more bytes onto the end of /etc/passwd. Eventually /etc/passwd is complete.

What if another process reads /etc/passwd before it’s complete?

e.g. Immediately after the 0_TRUNC, before the first write, login program reads /etc/passwd looking for a user. User isn’t there!
Typical fix: Lock /etc/passwd.

Recall flock syscall:
wait until any previous programs that used flock have closed this file.

chsh locks /etc/passwd before reading it,
and leaves the reading descriptor open while writing.
login locks /etc/passwd before reading it.

So login waits for chsh to finish.

Security problem: What if Joe can stop chsh from completing the file?
Signals

Normal control flow in a process can be interrupted by a signal.

Sometimes a signal terminates the process. (May “dump core,” i.e., save the process RAM to a disk file.)

Sometimes a signal pauses the process.

Sometimes a signal makes the process call a function specified by the program.

Sometimes a signal is ignored.
Signals generated by bugs

When process tries to access a weird memory location, it receives a SEGV signal. ("Segmentation violation.") Normal effect: terminate process.

When process divides by 0, it receives an FPE signal. ("Floating-point exception"); but floating-point division by 0 doesn’t trigger the signal!) Normal effect: terminate process.

And more. To avoid these signals, don’t access weird memory locations, don’t divide by 0, etc.
Signals generated by \texttt{kill()}

Syscall \texttt{kill(382,15)} tries to send signal 15 (\texttt{TERM}) to process 382.
Normal effect: terminate process.

Command: \texttt{kill -15 382} or \texttt{kill -TERM 382} or \texttt{kill 382}.

Does kernel allow Joe to kill process 382?
Yes if process 382 has uid Joe or real uid Joe.

So, if Joe runs setuid program \texttt{chsh}, Joe can kill \texttt{chsh} at any moment.

Fix: \texttt{chsh} sets its real uid to 0.
Signals generated by the tty

When Joe types Control-C, “foreground” processes on that tty receive an INT signal. (“Interrupt.”) Normal effect: terminate process.

More signals like this: HUP, TSTP, etc.

What are the “foreground” processes? Complicated combination of system data: process tty, process sid, process pgrp, etc. Normally chsh is in foreground.

Fix: chsh “dissociates” from Joe’s tty.
Signals generated by timers

Syscall `alarm(10)` tells kernel to send `ALRM` signal to this process in 10 seconds.
Normal effect: terminate process.

More signals like this: `VTALRM`, `PROF`.
`execve` doesn’t clear alarms.
`/home/joe/evil` calls `alarm(10)` and then `execve("/usr/bin/chsh",...,` timing the alarm to interrupt the rewrite of `/etc/passwd`.

Fix: `chsh` turns off alarms.
Signals generated by fds

When a process writes to a closed network connection, closed pipe, etc., it receives a PIPE signal.
Normal effect: terminate process.

Fix: chsh tells kernel to ignore PIPE for this process.

Actually, can ignore all signals except STOP and KILL.
Don’t need to turn off alarms or dissociate from tty.
But do need to set real uid, so Joe can’t send KILL.
Resource limits

Each process has several resource limits in system data.

Complete list of resource limits depends on the system. See /usr/include/sys/resource.h.

Some important limits:
limit on CPU time;
limit on memory allocation;
limit on number of fds;
limit on number of bytes in a file being written.
Process can reduce its own resource limits.

execve preserves resource limits.

/home/joe/evil sets CPU-time limit for itself, then runs chsh, choosing the limit to kill chsh immediately after O_TRUNC.

Or sets number-of-bytes limit.

Or sets some system-specific limit that interferes with writing the file.

Fix: chsh can check the limits, if it knows the complete list.