Laura Berrill, Techworld, 2004.10.11:

“Solaris security suffers image problem

“A highly critical security hole has been reported in the X Pixmap (libXpm) technology shipped with Solaris and JDS for Linux, which could allow someone to run code on your system if a modified X Pixmmap [sic] (.xpm) image is loaded. . . .

“Sun is still building a patch to the problem, first detected on Friday, and advice to users is to avoid loading X PixMap (.xmp) [sic] images from untrusted sources in the meantime.”

Many programs use libxpm;
Sun's JDS is just one of them.

This bug was published
on 2004.09.15 by Chris Evans:
`http://msgs.securepoint.com`
`/cgi-bin/get/bugtraq0409/119.html`

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

Assignment due 2004.10.18: 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.

The printing problem, recap

If printer and user's `pagesprinted` file
are writable to user
(i.e., owned by user, or permissions 622),
then the user has too much power.

If writable to network server
that trusts user to identify himself,
then the user has too much power.

The setuid-`lpr` solution:
printer and user's `pagesprinted` file
are writable only to `root`;
`lpr` program is setuid,
so it runs as `root`.

Setuid `lpr` can be secure,
but only if it's written
very, very, very carefully.

Local attacker has many ways
to control a setuid program:
fds, args, environ, cwd, tty,
rlimits, timers, signals, etc.
Even worse, this list varies
between Linux, BSD, Solaris, etc.

Writing a program that handles
all of these channels safely
is much more difficult than
writing a program that handles
a single input channel safely.

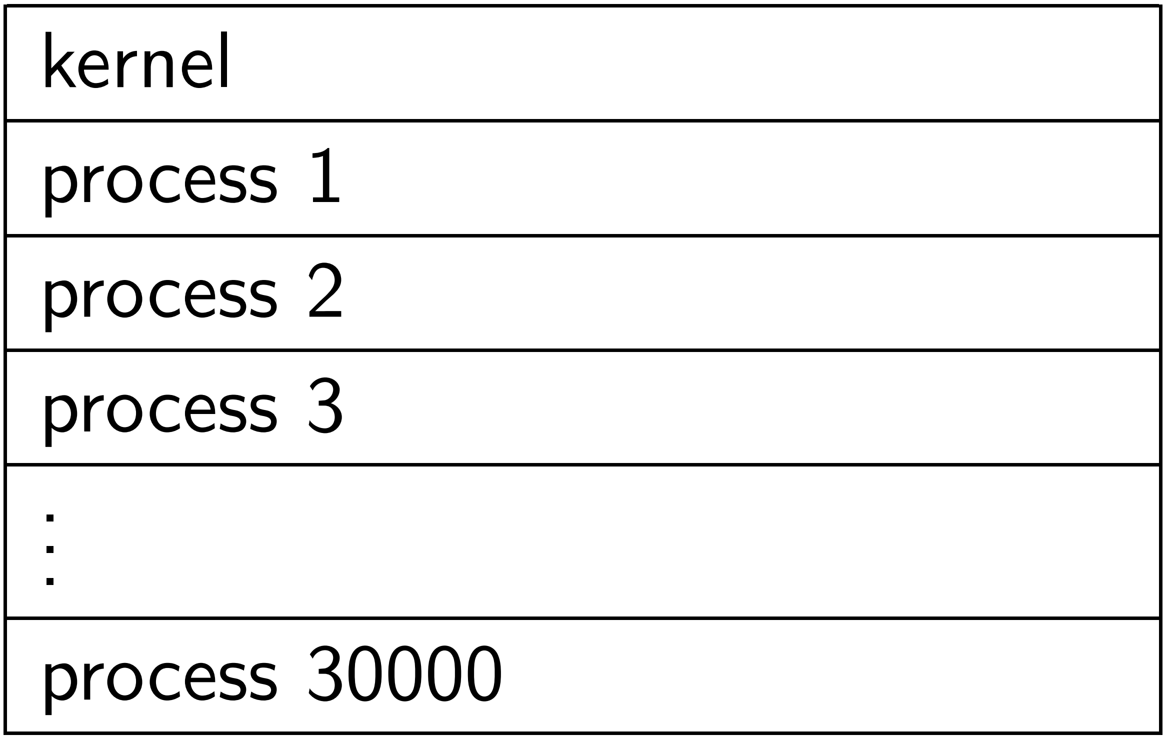
UNIX has many setuid programs
providing restricted access to
the password database, modems,
printers, mailboxes, terminals, etc.

Tiny bugs in these programs
have produced many security holes.
We'll see the details.

Could eliminate the setuid programs
using new `getpeereid` syscall
or using cryptographic tools,
but setuid is still widely used
and continues generating new holes.

What is a process?

Computer's memory is divided into **processes** and the **kernel**:



kernel

process 1

process 2

process 3

⋮

process 30000

Each process contains

system data, **registers**, and **RAM**:

kernel		
process 1 data	regs	RAM
process 2 data	regs	RAM
process 3 data	regs	RAM
⋮		
process 30000 data	regs	RAM

Picture is not to scale.

RAM has several big “segments”:

text (running program),

data (global variables),

stack (local variables),

heap (allocated variables), etc.

System data for a process
(often called `struct proc`):
user identifier, more identifiers,
process group, process session,
open file information,
signal actions, etc.
Details: `/usr/include/sys/proc.h`.

Process cannot read or write this data
except through syscalls.

Process cannot read or write
another process's data/regs/RAM
except through syscalls.

(CPU enforces these restrictions.
Syscalls are defined by kernel.)

Say user Joe has identifier 1257.

Joe logs in and runs a program,
i.e., creates a process
containing that program in its RAM.
Process user identifier is 1257.

Process tries opening `/dev/ulpt0`
for writing with `open` syscall.

The `open` syscall checks
whether this access is allowed.

Rules, a bit simplified: if `/dev/ulpt0`'s
owner matches process's user identifier,
or if process's user identifier is 0 (`root`),
or if file permissions are 622,
writing is allowed. Otherwise not.

Assume that `/dev/ulpt0`
has permissions 600
and owner different from 1257.
Then the `open` fails.

However, suppose the program has
owner 0 and permissions 4755 (setuid).
Then the process user identifier
is 0 instead of 1257,
so the `open` succeeds.

Similarly: If `root` logs in
and runs a non-setuid program,
the process user identifier is 0,
so the `open` succeeds.

A simple setuid security hole

Recall that `lpr` needs to handle `/etc/lpd/joe/pagesprinted`.

Where does it find username `joe`?

Here's an easy way: `getenv("USER")`.

Whoops, that's a security hole!
Joe can charge his printing to Bill.

SunOS 4.1.3 `chsh` command had the same security hole until 1997.
(Caught by Trevor Linton.)

What does Joe do? He runs

```
    env USER=bill lpr
```

so that `getenv("USER")` returns `"bill"` inside `lpr`.

What `env` does:

```
execve("/usr/bin/lpr"
    ,{"lpr",0}
    ,{"PATH=...","USER=bill",0})
```

The strings `PATH=...` and `USER=...` are **environment variables**. They're controlled by Joe.

`getenv` ends up reading `USER=bill` and trusting it.

Let's watch this attack in detail.

Process is owned by Joe:
i.e., user identifier 1257.

Process runs

```
execve("/usr/bin/lpr"
    ,{"lpr",0}
    ,{"PATH=...","USER=bill",0}).
```

What does the `execve` syscall do?

1. It copies the `lpr` code and data from the file `/usr/bin/lpr` into RAM.

2. Because `/usr/bin/lpr` is setuid 0, `execve` sets the process uid to 0.

3. It clears the rest of memory, except that it pushes `{"lpr",0}` and `{"PATH=...","USER=bill",0}` onto the stack.

4. It creates a variable `environ` pointing to `{"PATH=...",...}`.

5. It pushes that pointer, a pointer to `{"lpr",0}`, and `1` onto the stack.

6. It jumps to the start of `lpr`'s `main`.

Later `getenv` uses `environ` to find `"USER=bill"`.

What should `lpr` do instead?

Process has another uid in system data:
the “real uid.”
For setuid programs,
the uid changes; the real uid doesn’t.
In this case,
the uid is 0; the real uid is 1257.

`getuid` is a syscall
that returns the real uid.

`lpr` should call `getuid`,
handle `/etc/lpd/1257/pagesprinted`.

`getenv("USER")` can’t be trusted
in setuid programs, but `getuid()` can.

Another example

Sendmail bug fixed 1996.10.17:

```
h = res_search(host,...);
```

Why is this a bug?

Sendmail is a setuid program.

It accepts mail from local users
into an outgoing “mail queue”:

```
bill% sendmail -t
To: eric@cs
Here’s the secret number
you wanted: 867-5309.
```

Sendmail might deliver the mail now,
but not if this computer is busy.
(Attacker can make the computer busy.)

Sendmail also allows local users to “run the queue,” i.e., try delivering all mail now:

```
    joe% sendmail -q
```

Because Sendmail is setuid, it can read and write the queue.

Sendmail tries to deliver Bill’s message to `eric@cs`. It uses `res_search`, a BIND library function that sees

```
    search uic.edu
```

in `/etc/resolv.conf`, converts `cs` into `cs.uic.edu`, and looks up address of `cs.uic.edu`.

Okay; why is this a bug?