Capsicum

What is the confused deputy problem?

- Existing compiler given new privilege to write /sysx/stat file
- Bad users can use this privilege to clobber /sysx/billing
  - fort -o /sysx/billing file.f
- Reference: http://www.cap-lore.com/CapTheory/ConfusedDeputy.html

What is the goal of this work?

- Provide a "realistic adoption path" for least privilege

What are DAC/MAC - and why do they not help with least privilege?

- DAC protects users from each other
  - Not helpful to protect a single application from itself
  - Leads to "ambient authority" - what's this?
    - E.g., Unix uid/gid like global variables that affect all calls
  - "All eggs are in one basket" (p. 15)
- MAC traditionally enforced on users by system administrator
  - Not useful/available as mechanism within an application
    - Exception: Decentralized Information Flow Control (DIFC)
      But that is more the realm of research OSes

What is privilege separation?

- Break an application into multiple protection domains (Usually processes)
- Example: protecting sshd server from remote exploits
  - Server requires privileges: port 22, private host key, assuming user id
  - Split into two processes:
    - Privileged parent process: UID root, listens on port 22, has private key
      - Spawns worker, does private key ops, approves user authentication
    - Worker process: UID nobody, chroot /var/empty, has single TCP connection
      - Does everything else (parsing protocol, etc.)
  - SSH goes through two phases: pre-authentication and post-authentication
    - Once authentication okay, worker should assume privileges of user
      - Needs to spawn shell, handle X11 forwarding, etc.
    - But how to endow worker with more privileges?
      - Can't: can transition UID from root, but not from nobody
      - Gnarly hack: Spawn new process as user, transfer over memory of worker

Why is it hard for libraries to spawn sandboxed helper processes?

- If child exits, parent may get SIGCHLD signal, could confuse parent
  - Means code won't be 100% compatible (p. 10), but not a huge deal, either
  - E.g., gconf and alsa/pulseaudio libraries can spawn processes on linux
- Bigger problem: Often creating the sandbox requires privileges
  - (e.g., have to be root to call chroot, or to setuid nobody)?

Are ordinary Unix file descriptors a form of capabilities?

- Like capabilities:
  - Simultaneously designate and confer access to a resource
  - Can pass them between processes to delegate access
    - Both through inheritance, and explicit passing over unix-domain sockets
- Unlike capabilities:
  - Can't attenuate a file descriptor (e.g., go from read/write -> read-only)
  - Read-only isn't really read-only
    - Change fd state (e.g., lseek, fcntl)
    - Worse: change file state (e.g., fchmod, futimes, flock)
  - Set of file descriptors doesn't say anything useful about process authority
    - Can create new fd's out of thin air (e.g., open, socket, etc.)

What is Capsicum?

- New file descriptor type: capability
  - Wraps normal file descriptor with bit map of 60 possible rights
  - Examples: CAP_READ, CAP_WRITE, CAP_SEEK
    - CAP_READ lets you read file with pread
    - Read (which modifies fd offset) requires both CAP_READ and CAP_SEEK

- New system call: cap_enter
  - Drops access to all global namespaces
    - No access to process IDs, sysctl, named shared memory segments
    - No access to root directory, current working directory
    - How do you open or manipulate files?
      - Relative to existing capabilities (e.g., openat, unlinkat, renameat)
      - But note relative component cannot contain ".."
        - ".." ban must apply to direct argument and to any expanded symlinks
- New system call: cap_new
  - Creates an attenuated (less privileged) version of existing capability
  - Can mask off some of the 60 different possible permissions

How does capsicum solve library helper process problem?
- New system calls: pdfork, pdgetpid, pdkill, pdwait4
- pdfork: creates new process associated with process file descriptor
  - If file descriptor is ever closed (e.g., parent dies), kills child
  - But not airtight--can create cycles or PD_DAEMON to avoid kill-on-close
- Other syscalls work on returned fd, so independent of kill/wait/SIGCHLD
  - Can use poll on fd to catch exit
  - Actually nicer than SIGCHLD which interacts trickily with select/poll

What is helper library libcapsicum (libcasper) and why?
- API to simplify creating and delegating rights to a sandbox
  - Creates unix-domain socket for communication with child
  - Cleans child of any extra capabilities before calling cap_enter
  - Executes child process with fdexec
- Actually executes special Capsicum-aware run-time linker--why?
  - Regular dynamic linking uses absolute pathnames (forbidden)
  - Also ELF embeds absolute pathname of interpreter in binaries
- Note lcs_get() tells you if capability mode is even enabled for current proc
  - And returns unix-domain socket for talking to parent
  - Can request additional capabilities from parent using fd passing

What's involved in retrofitting capsicum to an existing OS such as FreeBSD?
- Must block access to system calls which access global state
  - FreeBSD has ~500 system calls, have to think which are permissible
  - Could insert hundreds of "if (capability_mode ())" checks
    - ...or use subsetted system call table (already supported for emulation)
  - A few system calls (sysctl, shm_open) need more fine-grained checks
- Must always check access rights against operation when using capabilities
  - System already has helper function fget to get file struct from fd
  - Capsicum modifies fget to take an extra argument (required rights)
    - Easy to check fget always throws error if wrong writes requested
    - Hard to miss call to fget, as compilation will fail if extra arg missing
  - What if some obscure corner of kernel bypasses fget?
    - Will get wrapper capability instead of underlying file struct
    - Presumably legacy code won't know what to do with capability, so okay
- Must prevent access to prohibited parts of file system (e.g., "..")
  - All file system walking handled by a single function, namei
    - So changes can be implemented in a single place
    - Note expanded symlinks also go through namei, so no issue there

What is procstat?
- ps-like tool that allows enumeration of file descriptors
- Extend to show capabilities and rights
- The set of capabilities a process has should tell us what it can do
- Also useful for debugging

For each of: tcpdump, dhclient, gzip, chromium
- What is the security issue?
- Why does it need high privileges?
- What is the natural way to privilege separate the application?
- How did they do it in Capsicum?
  - For chromium, be sure to go over Figure 12

What is PLASH, mentioned on p. 10 (sec 4.3)?
  Most apps described start privileged, open some files, then call cap_enter
  What if shell (PLASH) opened files and passed them into program?
    Then program could start in capability mode, would trust it much less
  What would this look like? E.g., plash$ cat /etc/motd
    plash would open /etc/motd, then exec "cat" "/dev/fd/3"

What questions should be answered by evaluation section?
  How simple or error-prone is Capsicum compared to other sandbox approaches?
  How impermeable or porous is Capsicum's isolation compared to alternatives?
  Is performance an issue with Capsicum?

How hard do you think it is to program with capsicum?
  Debugging privilege-separated app is distributed debugging--a hard problem
  Cool trick (p. 7), construct app so you *can* run everything in one process
    Allows you to disable security and perform conventional debugging

Does Capsicum solve the confused deputy problem?
  Not really any notion of embedding capabilities for /sysx in a program
  So would need to:
      1. Make fort setgid
      2. Open "stat" before parsing command-line arguments
      3. Drop privileges (setgid (getgid ())
      4. Parse command-line and open requested files
      5. At that point cap_enter ()
    Even without Capsicum and 5, would still solve the problem