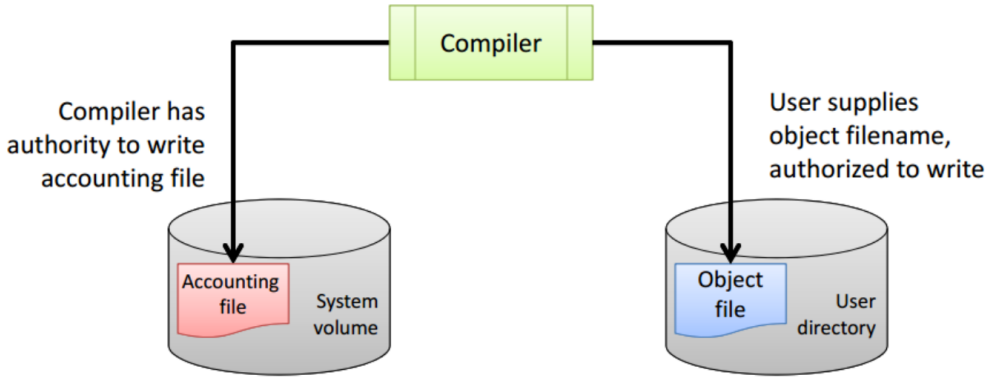
The Confused Deputy

Walking the garden path 1

- The compiler
  - Reads user source files
  - Writes object files and optional debug output files
  - Updates a system billing file, recording CPU consumed
- The OS uses traditional access control lists (ACLs) on named files
- Billing file
  - Owned by system
  - Writable by compiler
  - Not writable by regular users
- The compiler is trusted to update billing file correctly

Walking the garden path 2



- Compiler writes debugging info to fixed
- New feature: Let the user choose the de
- Attack: DEBUG=BILLING
  - Compiler overwrites billing file with debug information
  - Access check – can compiler write billing file – succeeds
- Compiler has two sources of authority
  - From user – can write user's directory
  - From administrator – can write billing file
  - Too much **ambient authority** – the compiler became confused!

Capabilities to the rescue

- A file descriptor
  - Is a handle to a specific open file or directory
  - Encodes what you can do (read, write, both)
  - Is unforgeable in the sense that:
    - Processes can’t just guess an fd and get a new object
    - Only get fds by inheritance (fork/exec) or by explicit passing (e.g., via Unix domain sockets).
- Launcher process opens the debug file
  - Launcher has limited task and limited ambient authority
  - Launcher passes debug fd to compiler
    - As a known fd number (3, 4, etc.)
    - Across a Unix domain socket (passing fds with SCM_RIGHTS)
    - Via a well-defined convention (“write debug output on fd 3”)

Confusion clears

- Caller/admin controls its authority
  - Admin determines what capabilities are passed to the compiler
- Separate, non-multiplexed roles
  - No ambient authority (e.g., from setuid program)
  - Roles express access decisions via capabilities
- Least privilege by construction
  - Privilege determined by capabilities granted