# DEMO suid program setgid directory

#### **Extended Attributes**

```
# lsattr /etc/passwd /etc/ssl
------e-- /etc/passwd
-----I--e-- /etc/ssl/certs
```

- Require support from file system
- Management via 1sattr, chattr
  - Undeletable (u)
  - Append only (a)
  - Immutability (i)
  - Secure deletion (s)
  - Compression (c)
  - Hashed trees indexing for directories (I)

# DEMO Extended Attributes

#### **POSIX ACLs**

Extend UNIX permission model to support finegrained access control

```
$ sudo setfacl -m u:pizzaman:r secret
$ getfacl secret
# file: secret
# owner: root
# group: root
user::rw-
user:pizzaman:r--
group::---
mask::r--
other::---
```

# DEMO POSIX ACLs

### Why use SUID at all?

- ping sends ICMP packets
  - Only privileged programs can access "raw" sockets
  - SUID root solves that problem
- Web server binds to port 80/443
  - Privileged ports (<1024) only root can bind to</li>
  - SUID root would make the entire web-server run as root (what's the problem?)

### **Linux Capabilities**

- Linux has a fine-grained notion of privilege called *capabilities* 
  - Not to be confused with actual capabilities
- Partition root privilege into smaller units
  - CAP NET ADMIN
  - CAP NET BIND SERVICE
  - CAP NET RAW
  - CAP\_KILL
  - CAP\_SYS\_MODULE

### **Linux Capabilities**

#### **Shells**

```
# echo $SHELL
/bin/sh
```

- Shells: the classic interface to UNIX systems
  - Interactive REPL environment
  - Also, a convenient programming language
- Program execution, pipelining
  - Fine-grained control of subprocess environment
  - Redirections & pipelining (<, |, and >)
- Many different flavors
  - Bourne shell (sh), Bourne again shell (bash), C shell (csh), Korn shell (ksh)

# The Unix Philosophy

Doug McIlroy (1978)

- (i) Make each program do one thing well. ...
- (ii) Expect the *output* of every program to become the *input* to another, as yet unknown, program. ...

```
(iii) ...
```

(iv) ...

### **Process System Calls**

- **fork** (duplicate current process, create a new process)
- **exec** (replace currently running process with executable)
- **exit** (end process)
- wait (wait for a child process)
- **getpid** (get process PID)
- **getpgrp** (get process GID)

#### **Executing Programs**

```
int execve(
   const char *path,
   char *const argv[],
   char *const envp[]);
```

- Executing a new program: Invoke the exec() syscalls
  - exec\*() replaces the current program with the program specified as path
  - exec\*() does not return
  - Initializes a new virtual address space
  - Invokes ld-linux.so.2, loads shared libs performs runtime linking (ELF, dynamically linked binaries)
  - Invokes interpreter specified in form of #! /path/to/interpreter

# fork()

Syntax: pid = fork();

#### Get almost identical copy (child) of the original (parent)

- File descriptors, arguments, memory, stack ... all copied
- Even current program counter
- But not completely identical why?

#### Return value from fork call is different:

- 0 in child
- PID > 0 of the child when returning in parent

# fork() cont.

```
pid t child = fork();
switch (child) {
  case -1:
    //something went wrong ...
    exit(1);
  case 0:
    //I'm the child
    break;
  default:
    //I'm the parent and the child's pid is child
    break;
```

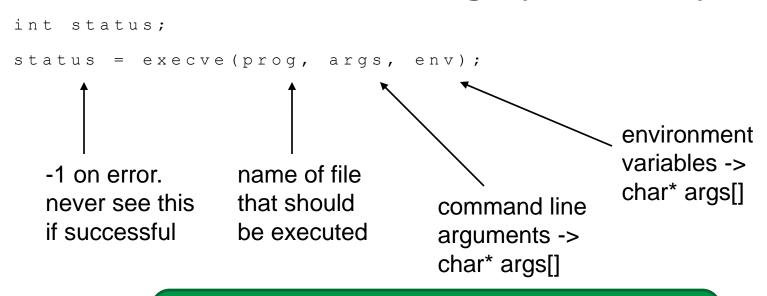
- system() wrapper around fork() then exec\*()
  - Implemented in libc.so <u>not</u> a system call

# exec()

#### Change program in process

i.e., launch a new program that replaces the current one

#### Several different forms with slightly different syntax



What does execve return?

# **Monitoring Programs**

```
pid_t waitpid(pid_t pid, int* status, int options);
```

- wait\*() family allows parent to check status of children
  - WIFEXITED, WEXITSTATUS
  - WIFSIGNALED, WTERMSIG
  - WIFSTOPPED, WSTOPSIG
- Performing wait\*() is required to clean up zombie processes
  - Otherwise, terminated programs remain in Z state