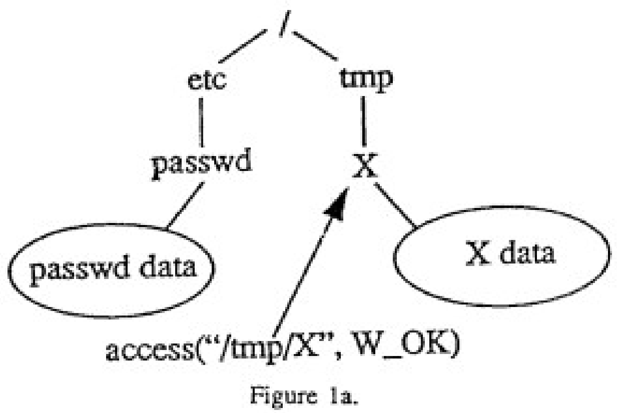
Solutions to TOCTTOU

CSE 536 Spring 2026

Filesystem TOCTTOU is a type of race condition

https://jedcrandall.github.io/courses/cse536spring2026/borisov.pdf https://www.usenix.org/legacy/publications/compsystems/1996/spr_bishop.pdf

Fundamental issue is that something changed between when you checked the file and when you used it.



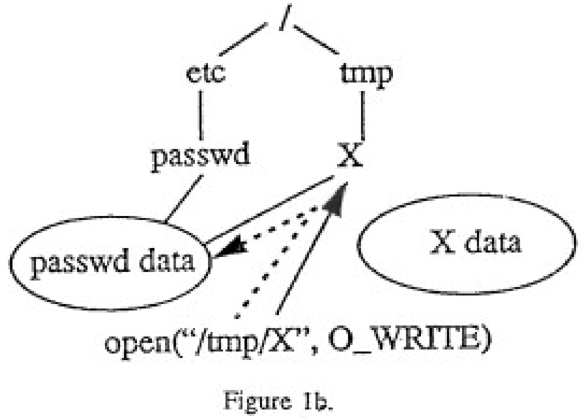
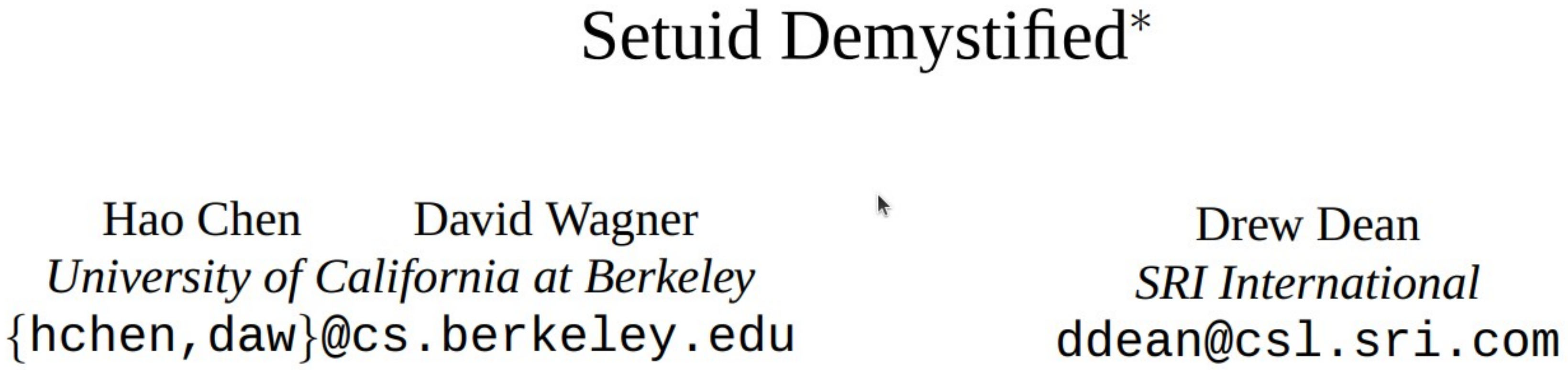


Figure 1. Example of the TOCTTOU binding flaw.

Simply dropping privileges to open the file used to be problematic...

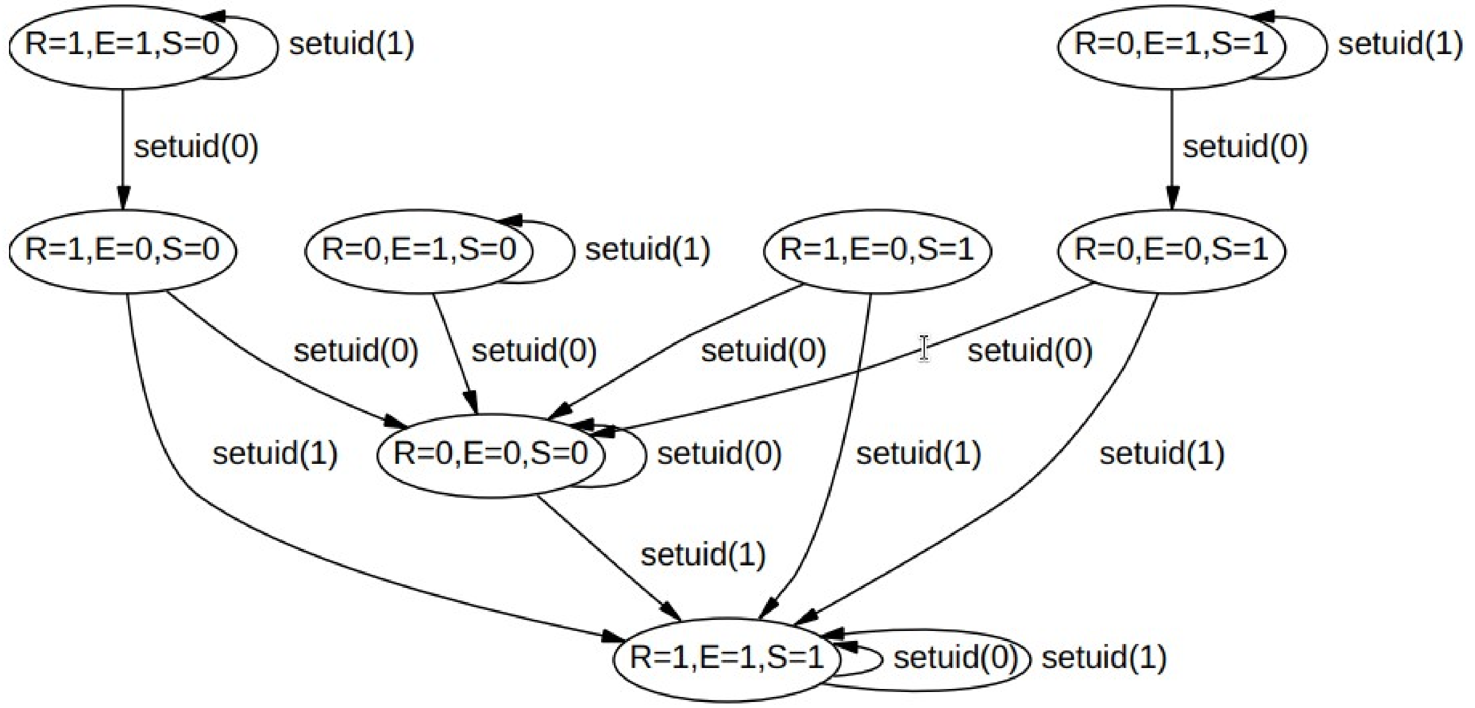


Setuid Demystified*

Hao Chen David Wagner
University of California at Berkeley
{hchen,daw}@cs.berkeley.edu

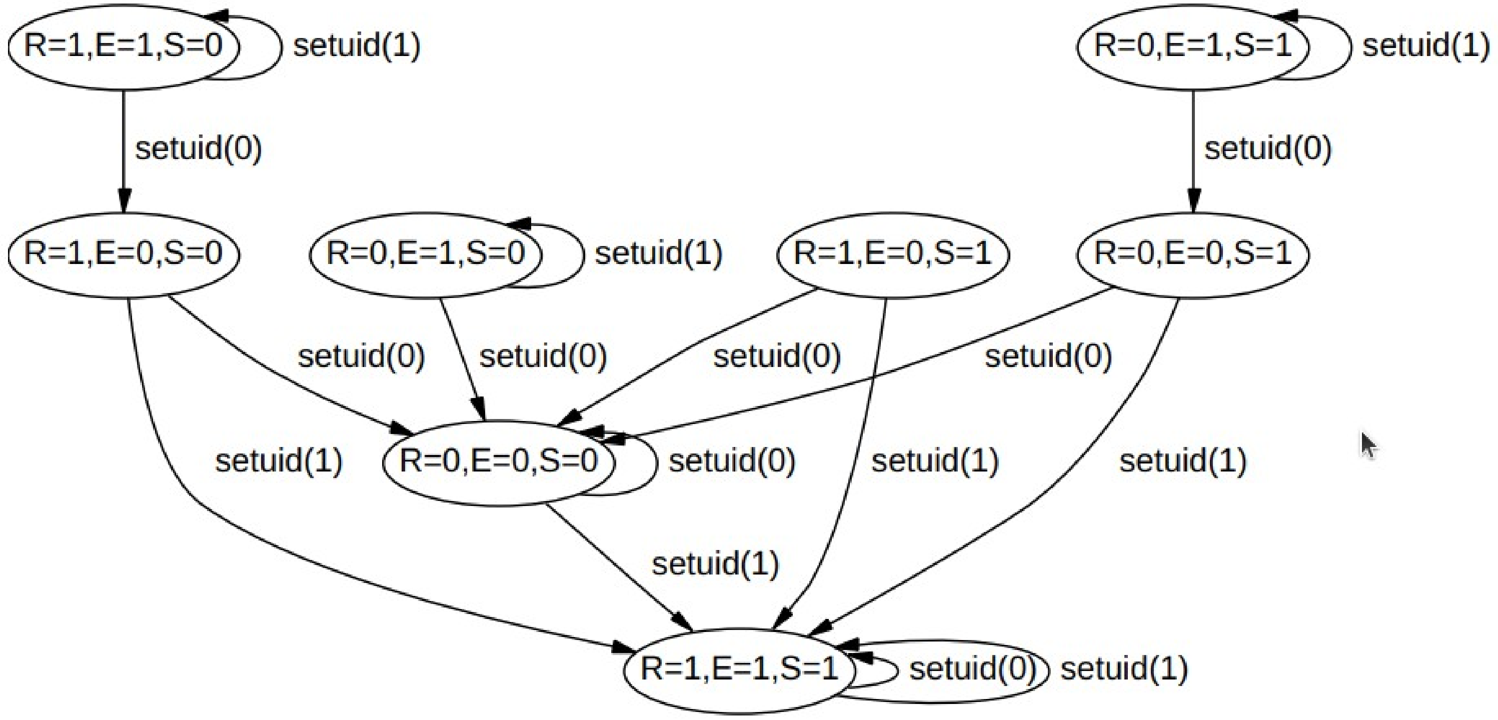
Drew Dean
SRI International
ddean@csl.sri.com

https://www.usenix.org/legacy/events/sec02/full_papers/chen/chen.pdf



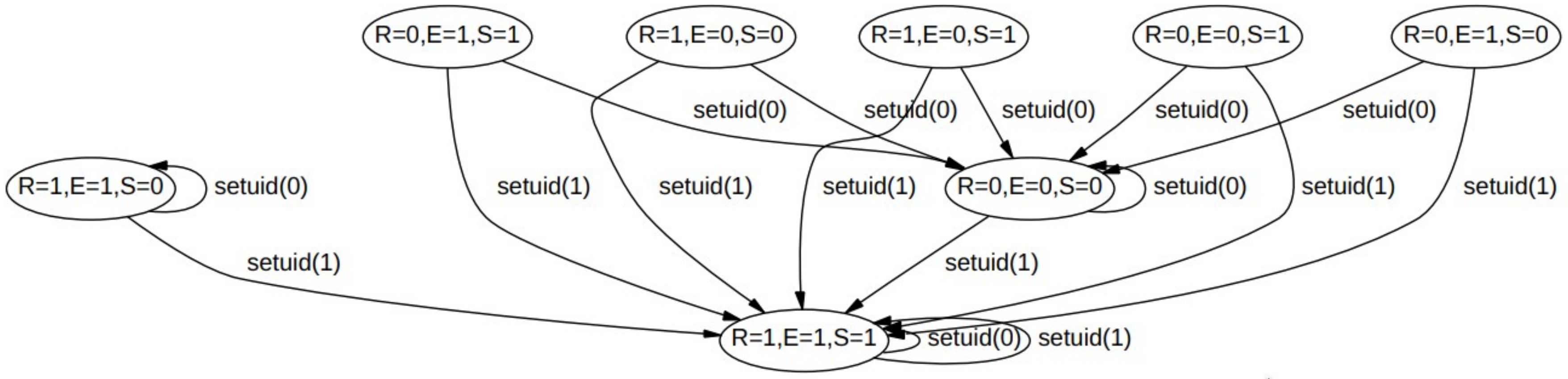
(a) An FSA describing *setuid* in Linux 2.4.18

https://www.usenix.org/legacy/events/sec02/full_papers/chen/chen.pdf



(b) An FSA describing *setuid* in Solaris 8

https://www.usenix.org/legacy/events/sec02/full_papers/chen/chen.pdf



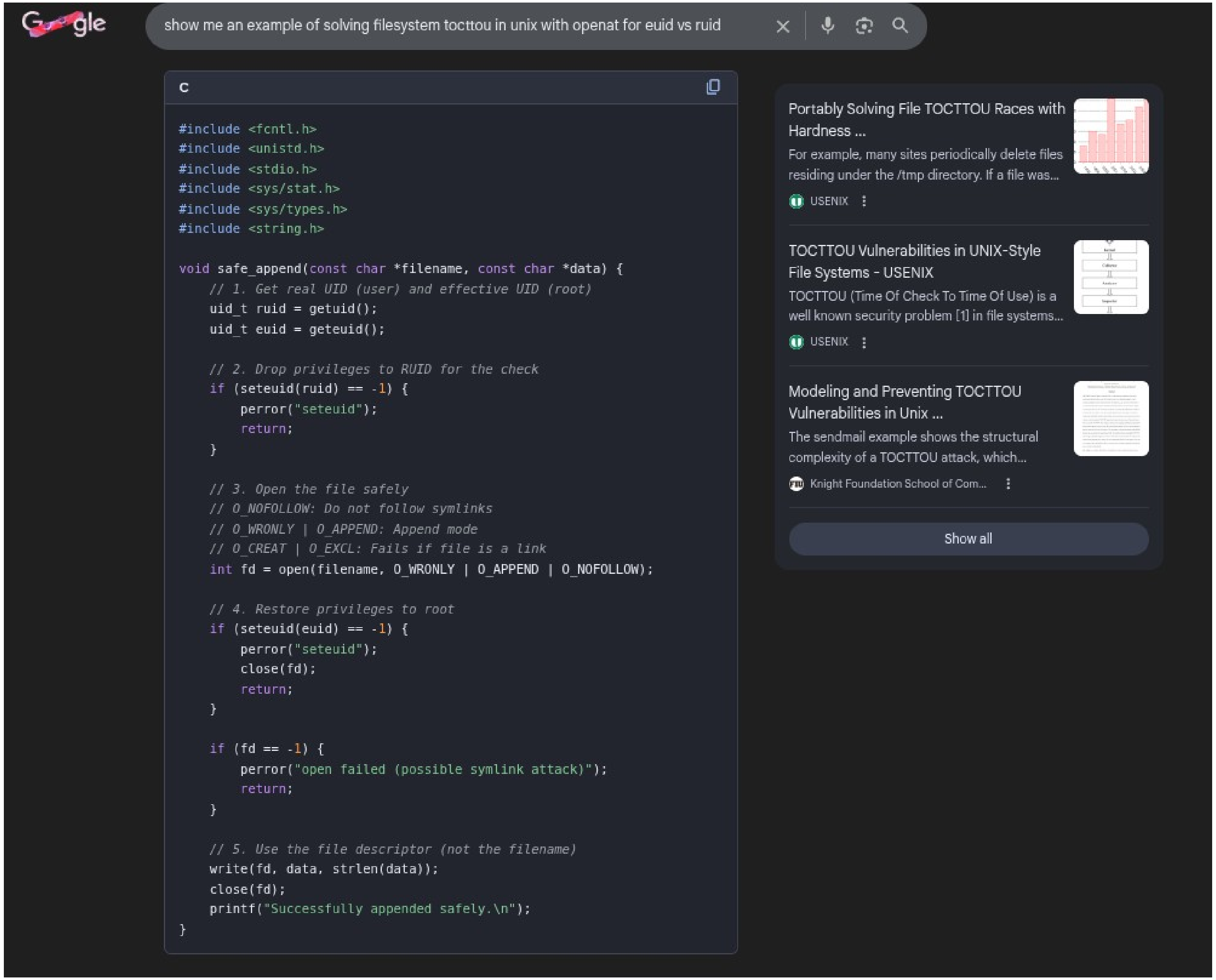
(c) An FSA describing *setuid* in FreeBSD 4.4

Old portable solution

- Parent had to fork a child
  - Child drops privileges and opens the file
    - Sends file descriptor back to parent through a UNIX domain socket

Now you can just use `seteuid()`, but you have to be careful of other concerns…

AI to the rescue! (?)



show me an example of solving filesystem tocttou in unix with openat for euid vs ruid

C

```c
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <string.h>

void safe_append(const char *filename, const char *data) {
    // 1. Get real UID (user) and effective UID (root)
    uid_t ruid = getuid();
    uid_t euid = geteuid();

    // 2. Drop privileges to RUID for the check
    if (seteuid(ruid) == -1) {
        perror("seteuid");
        return;
    }

    // 3. Open the file safely
    // O_NOFOLLOW: Do not follow symlinks
    // O_WRONLY | O_APPEND: Append mode
    // O_CREAT | O_EXCL: Fails if file is a link
    int fd = open(filename, O_WRONLY | O_APPEND | O_NOFOLLOW);

    // 4. Restore privileges to root
    if (seteuid(euid) == -1) {
        perror("seteuid");
        close(fd);
        return;
    }

    if (fd == -1) {
        perror("open failed (possible symlink attack)");
        return;
    }

    // 5. Use the file descriptor (not the filename)
    write(fd, data, strlen(data));
    close(fd);
    printf("Successfully appended safely.\n");
}
```

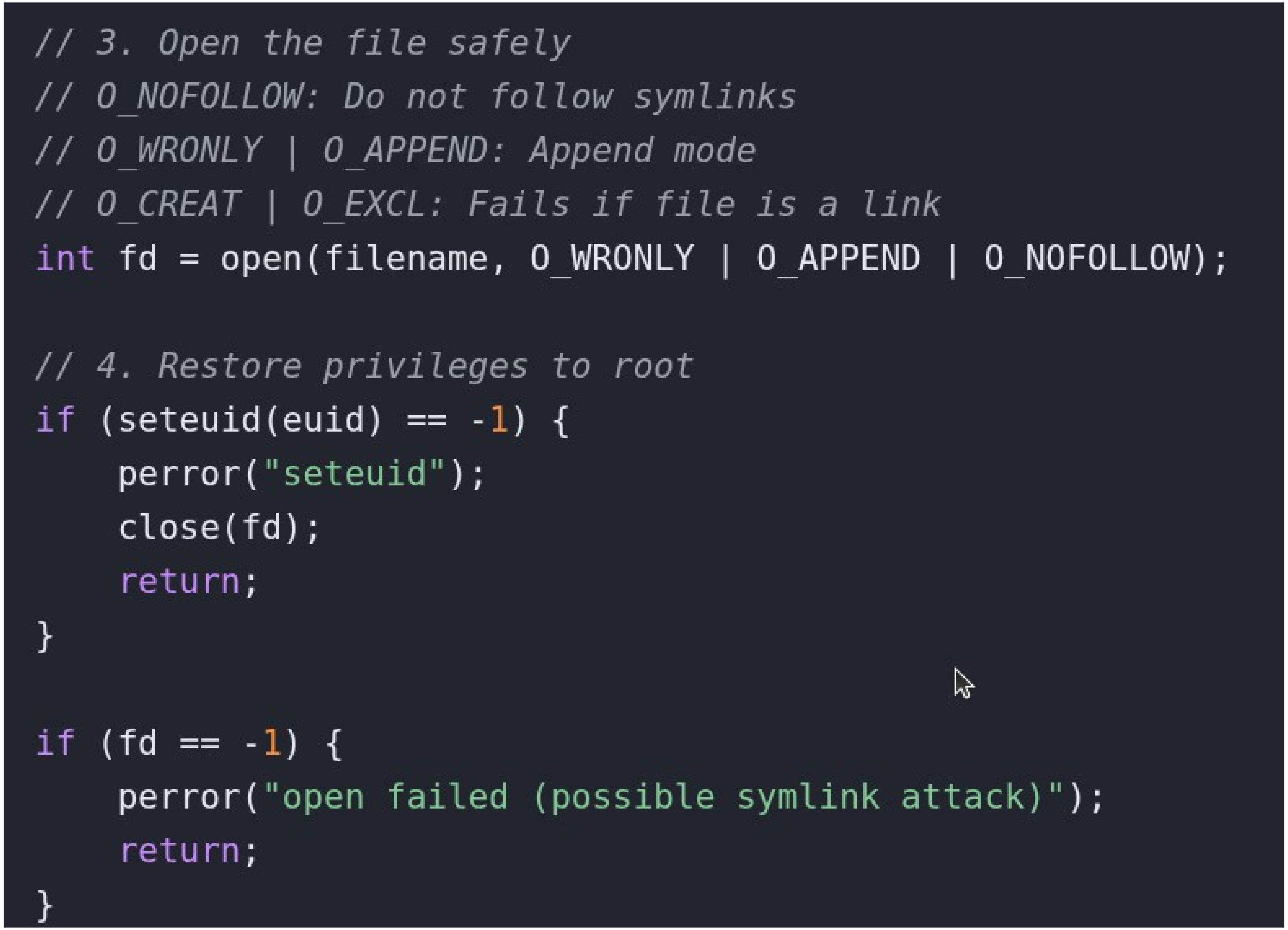
Portably Solving File TOCTTOU Races with Hardness ...
For example, many sites periodically delete files residing under the /tmp directory. If a file was...
USENIX

TOCTTOU Vulnerabilities in UNIX-Style File Systems - USENIX
TOCTTOU (Time Of Check To Time Of Use) is a well known security problem [1] in file systems...
USENIX

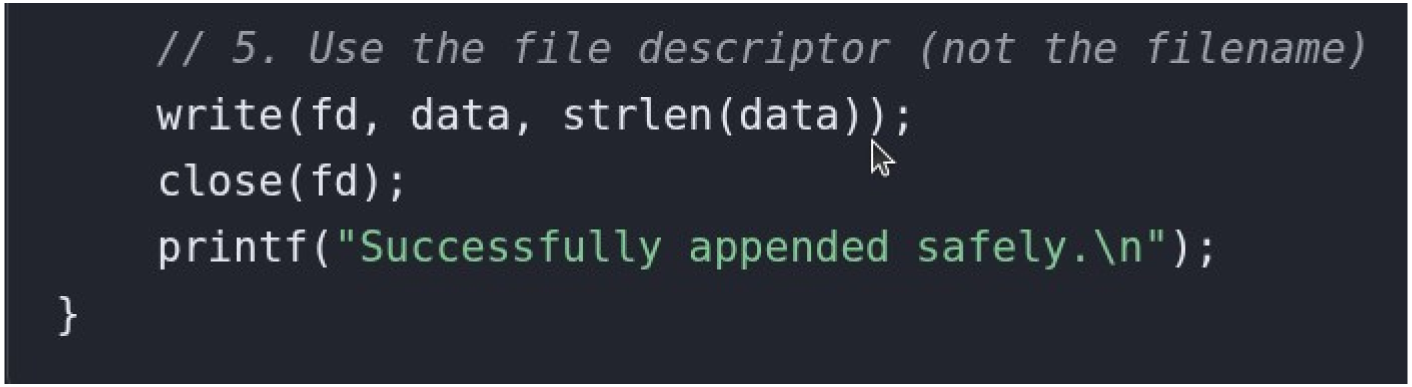
Modeling and Preventing TOCTTOU Vulnerabilities in Unix ...
The sendmail example shows the structural complexity of a TOCTTOU attack, which...
Knight Foundation School of Com...

Show all

```c
#include <unistd.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <string.h>

void safe_append(const char *filename, const char *data) {
    // 1. Get real UID (user) and effective UID (root)
    uid_t ruid = getuid();
    uid_t euid = geteuid();

    // 2. Drop privileges to RUID for the check
    if (seteuid(ruid) == -1) {
        perror("seteuid");
        return;
    }
```



```c
// 3. Open the file safely
// O_NOFOLLOW: Do not follow symlinks
// O_WRONLY | O_APPEND: Append mode
// O_CREAT | O_EXCL: Fails if file is a link
int fd = open(filename, O_WRONLY | O_APPEND | O_NOFOLLOW);

// 4. Restore privileges to root
if (seteuid(euid) == -1) {
    perror("seteuid");
    close(fd);
    return;
}

if (fd == -1) {
    perror("open failed (possible symlink attack)");
    return;
}
```

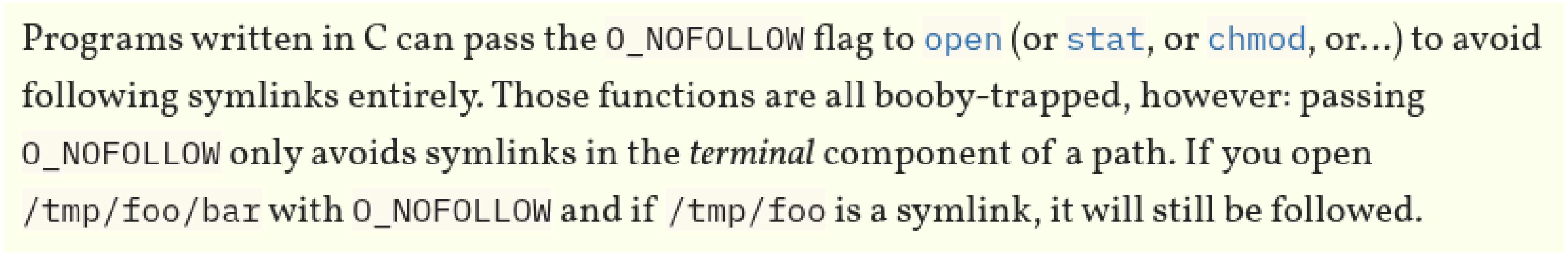


```c
    // 5. Use the file descriptor (not the filename)
    write(fd, data, strlen(data));
    close(fd);
    printf("Successfully appended safely.\n");
}
```

Caveats...

- `O_NOFOLLOW` doesn’t prevent symlinks in the path, and doesn’t apply to hard links
  - `filename` should not be used with root
  - Creating files and directories is a lot trickier than what AI came up with above
  - Doing things recursively requires `openat()` and working with file descriptors, starting at root (/)

https://michael.orlitzky.com/articles/posix_hardlink_heartache.xhtml



Programs written in C can pass the `O_NOFOLLOW` flag to `open` (or `stat`, or `chmod`, or…) to avoid following symlinks entirely. Those functions are all booby-trapped, however: passing `O_NOFOLLOW` only avoids symlinks in the *terminal* component of a path. If you open `/tmp/foo/bar` with `O_NOFOLLOW` and if `/tmp/foo` is a symlink, it will still be followed.