Time-of-check to time-of-use (TOCTOU) race conditions with file system API

Tuomo Turunen
2022-03-01

Introduction

- Definitions of TOCTOU
  - Wikipedia: Time-of-check to time-of-use

    “In software development, time-of-check to time-of-use (TOCTOU, TOCTTOU or TOC/TOU) is a class of software bugs caused by a race condition involving the checking of the state of a part of a system (such as a security credential) and the use of the results of that check.”
  - CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition

    “The software checks the state of a resource before using that resource, but the resource's state can change between the check and the use in a way that invalidates the results of the check. This can cause the software to perform invalid actions when the resource is in an unexpected state.

    This weakness can be security-relevant when an attacker can influence the state of the resource between check and use. This can happen with shared resources such as files, memory, or even variables in multithreaded programs.”
- This slide set concentrates on TOCTOU race conditions with **file system API**

What does it mean in practice?

- The most common TOCTOU race condition bug is:
  - First check if a file exists – time-of-check (TOC)
  - Then, based on the result of the check, do something with the file – time-of-use (TOU)
- Why is this a bug?
  - Another thread, another process or another user may remove or create the file between TOC and TOU – hence it is a race condition
- Are these serious bugs? Do we need to fix them?
  - Yes, race conditions are serious bugs
  - File system TOCTOU race conditions are potential security holes, as they can be exploited by an attacker with direct or indirect access to the file system
- See the next slides for concrete examples

Case 1: first check if a file exists, then use the file (buggy code)

```
/* BAD EXAMPLE! */
int remove_file_if_exists(const char *pathname)
{
    if (access(pathname, F_OK) == -1)
        /* File does not exist, do nothing */
        return 0;
    /* TOCTOU BUG! */
    if (unlink(pathname) == -1)
    {
        perror("unlink");
        return -1;
    }
    return 0;
}
```

Case 1: first check if a file exists, then use the file (fixed code)

```c
int remove_file_if_exists(const char *pathname)
{
    if (unlink(pathname) == -1)
    {
        if (errno == ENOENT)
            /* File does not exist, do nothing */
            return 0;
        else
        {
            perror("unlink");
            return -1;
        }
    }
    return 0;
}
```

Case 2: do multiple operations to the same file (buggy code)

```c
/* BAD EXAMPLE! */
int change_owner_and_mode(const char *pathname, uid_t uid, mode_t mode)
{
    if (chown(pathname, uid, -1) == -1)
    {
        perror("chown");
        return -1;
    }
    /* TOCTOU BUG! */
    if (chmod(pathname, mode) == -1)
    {
        perror("chmod");
        return -1;
    }
    return 0;
}
```

Case 2: do multiple operations to the same file (fixed code)

```c
int change_owner_and_mode_if_exists(const char *pathname, uid_t uid, mode_t mode)
{
    int fd = open(pathname, O_RDONLY | O_CLOEXEC);
    if (fd == -1)
    {
        perror("open");
        return -1;
    }
    if (fchown(fd, uid, -1) == -1)
    {
        perror("fchown");
        close(fd);
        return -1;
    }
    if (fchmod(fd, mode) == -1)
    {
        perror("fchmod");
        close(fd);
        return -1;
    }
    close(fd);
    return 0;
}
```

Case 3: create a file if it does not exist

```c
int create_file_if_not_exist(const char *pathname, const void *data, size_t size)
{
    int fd = open(pathname, O_WRONLY | O_CLOEXEC | O_CREAT | O_EXCL, 0644);
    if (fd == -1)
    {
        if (errno == EEXIST)
            /* File already exists, do not override */
            return 0;
        else
        {
            perror("open");
            return -1;
        }
    }
    write(fd, data, size);  /* Ignore write-errors for simplicity */
    close(fd);
    return 0;
}
```

What about directories?

- Opening files in directories other than the current working directory has similar race conditions
  - For example create two files:
    - `dir1/dir2/file1`
    - `dir1/dir2/file2`
  - Another thread, another process or another user can remove, rename or exchange `dir1` or `dir2` between creating `file1` and `file2` causing `file2` creation to fail or `file2` to be created into different directory
- These race conditions can be avoided by using `*at` variants of the file system API functions (`openat(2)`, `execveat(2)`, `faccessat(2)`, `fanotify_mark(2)`, `fchmodat(2)`, `fchownat(2)`, `fspick(2)`, `fstatat(2)`, `futimesat(2)`, `linkat(2)`, `mkdirat(2)`, `move_mount(2)`, `mknodat(2)`, `name_to_handle_at(2)`, `open_tree(2)`, `openat2(2)`, `readlinkat(2)`, `renameat(2)`, `statx(2)`, `symlinkat(2)`, `unlinkat(2)`, `utimensat(2)`, `mkfifoat(3)` and `scandirat(3)`)
- See the next slides for concrete example

Case 4: directory level race condition (buggy code)

```c
/* BAD EXAMPLE! */
int write_pub_priv_keys(const char *dir, const char *pub, const char *priv)
{
    char pub_filepath[PATH_MAX];
    snprintf(pub_filepath, sizeof(pub_filepath), "%s/%s", dir, "public.key");
    char priv_filepath[PATH_MAX];
    snprintf(priv_filepath, sizeof(priv_filepath), "%s/%s", dir, "private.key");

    int fd;
    fd = open(pub_filepath, O_WRONLY | O_CLOEXEC | O_CREAT | O_TRUNC, 0644);
    if (fd == -1)
    {
        perror("open");
        return -1;
    }
    write(fd, pub, strlen(pub)); /* Ignore write-errors for simplicity */
    close(fd);
    /* TOCTOU BUG! */
    fd = open(priv_filepath, O_WRONLY | O_CLOEXEC | O_CREAT | O_TRUNC, 0600);
    if (fd == -1)
    {
        perror("open");
        return -1;
    }
    write(fd, priv, strlen(priv)); /* Ignore write-errors for simplicity */
    close(fd);
    return 0;
}
```

Case 4: directory level race condition (fixed code)

```
int write_pub_priv_keys(int dir_fd, const char *pub, const char *priv)
{
    int fd;
    fd = openat(dir_fd, "public.key", O_WRONLY | O_CLOEXEC | O_CREAT | O_TRUNC, 0644);
    if (fd == -1)
    {
        perror("open");
        return -1;
    }
    write(fd, pub, strlen(pub)); /* Ignore write-errors for simplicity */
    close(fd);
    fd = openat(dir_fd, "private.key", O_WRONLY | O_CLOEXEC | O_CREAT | O_TRUNC, 0600);
    if (fd == -1)
    {
        perror("open");
        return -1;
    }
    write(fd, priv, strlen(priv)); /* Ignore write-errors for simplicity */
    close(fd);
    return 0;
}
```

Case 4 cont: opening a directory

```c
/* Using normal open() */
int fd = open("/path/to/dir", O_RDONLY | O_CLOEXEC | O_DIRECTORY);

/* Using openat() */
int fd = openat(dir_fd, "dir", O_RDONLY | O_CLOEXEC | O_DIRECTORY);

/* Getting fd from DIR* */
DIR *dirp = opendir("/path/to/dir");
int fd = dirfd(dirp);
```

Thanks!