TOCTTOU (time of check to time of use) Vulnerability (Race Conditions)
There is a window of time between the operations where permission check is performed and the data is used after the permission check where an attacker can change object in between.

Example:

1. Temp file bug
   A compiler compiles a file t.c and creates a file */tmp/t.a*
   An attacker now gives commands:
   mv /tmp/t.a /tmp/t.b

   ln –s /etc/passwd /tmp/t.a
   Later, when the complier deletes file */tmp/t.a* in believing that it created it, */etc/passwd* will also be deleted. (Actually, this attack does not work since the remove command will typically remove the link and not the file. But you should get the general idea.)

2. setuid programs typically run with different real and efffective userids, e.g., with real *uid x* an effective *uid y.* It the person that executed the setuid program specifies the name of an output file to the setuid program, it needs to check if the real user has has the permission to write this file. It may use the access system call to do this:

3. ```
   if (access (f) == ok)    // uses real uid
         open file (f); // uses effective uid=0, so will always succeed
   ```
   The check function *access()* is done using the real *uid*, while the open file function uses the *euid*. An attacker can wait until the *access(f)* is done, and then does something similar to the previous attack, i.e. link the file *f* to some sensitive file *x* the attacker want to access, because open file *(f)* will be done using *euid*, the attacker may now be able open a file that he did not have access to.

4. cron
   A daemon called cron in UNIX could be used to periodically clear the directory */tmp/** Assume that it works as follows. It recursively descends in to a sub directory of /tmp. removes all files, moves to the next higher directory, as so on. Suppose that the attacker has created a subdirectory called a/b within /tmp. As some point, rm might do the following

   ```
   cd  /tmp/a/b
   rm all files in current directory
   cd ..
   rm all files in current directory (/tmp/a)
   cd ..
   rm all files in current directory (/tmp)
   ```

   an attacker will wait until "*cd /tmp/a/b*" is executed, and then he/she will use the command:
   mv /tmp/a/b /tmp/b

   Now, when *cd ..* is executed, the current directory is actually the /tmp, and when the next cd is done, it will be in the root directory, and may then delete all files in the root directory.

   Note: a directory (or a file) is an object in the file system; it is distinct from the name used to access the directory (or file). The above attacks were possible since the code implicitly assumed that the same name will reference the same object, but the attack violated this assumption.

   Most TOCTTOU attacks arise due to name-based operations --- two statements that assume that they are referencing the same object because they use the same name, but the attacker may be able to invalidate this assumption.

Summary
The term "softwre vulnerability" is used in the context of programs we trust, not malicious/untrusted programs. For such programs, we assume that they wont intentionally do any thing bad, but they may have vulnerabilities that may be exploited by an attacker to cause damage. So, we focussed on vulnerability mitigation techniques so far.

When dealing with untrusted software, it may be intentionally malicious. There is no need for an attacker writing a piece of software to rely on vulnerabilities to cause damage. They may directly code attacks into their software. For instance, if you download a freeware program that purports to compress
   ```

files, and you run it on your system, it can directly remove all the files on your system. It does not have to rely on buffer overflows or race conditions for this purpose.

In the next part of the course, we will discuss security techniques for untrusted software. The primary way to deal with untrusted software is to ensure that it observes a security policy specified by the person running that code.

Moreover, it is important to realize that untrusted code may not cooperate with you in policy enforcement --- instead, it may work against it, and try to actively subvert it. As an example, consider the stackguard policy which (attempts to) ensure that return address is not corrupted due to a buffer overflow. This is done by instrumenting the program so that the following check is performed before a return:

```
if (canary == global_random)
        return;
else abort();
```

If this instrumentation is added to malicious code, it can be easily subverted. For instance, untrusted code may change the value of global_random so that the above check will always succeed and the return allowed to continue. Alternatively, it may contain a direct jump to the return statement, thereby bypassing the check.

(Although in high level languages you cannot jump to any point inside a code, it is certainly possible in binary code, which is the form in which most untrusted code arrives.).