Symlink attacks

- **Do not assume that symlinks are trustworthy:**
  - Example 1
    - Application A creates a file for writing in /tmp. It assumes that since the file name is unusual, or because it encodes A's name or pid, there is no need to check if the file is already present
    - Attacker creates a symlink with same name that points to an important file F. When root runs A, F will be overwritten.
  - Example 2
    - User A runs an application that creates a file in /tmp/x and then later updates it.
    - User B attacks this application by removing /tmp/x and then creating a symlink named /tmp/x that points to an important file F.
- **Hard links and file/directory renames can also be used to carry out some of these attacks, but they are difficult because there are more restrictions on them.**

Race conditions

- **Time-of-check-to-time-of-use (TOCTTOU) attacks**
  - Often arise when an application tries to protect itself against name-based attacks
- **Example**
  - A setuid application permits a non-root user to specify the name of an output file, say, for logging
  - It checks if the real user has permission to write this file, usually using the *access* system call
  - Attacker modifies the file between *access* and *open*
    - Checks OK, but the attack succeeds!

Race condition examples

- **access/open**
- **chmod/chown**
- **Directory renames**
  - Root invokes rm -r on /tmp/* to clean up /tmp
  - Attacker creates a directory /tmp/a and then another directory /tmp/a/b
  - rm may (1) cd into /tmp/a/b, remove all files in it, (2) cd into "..", (3) continue to remove files in /tmp/a, (4) cd ".." and (5) continue to remove files in /tmp
  - Attacker moves /tmp/a/b to /tmp between (1) and (3), causing files in / to be removed in step (5).

Succeeding in Races ...

- **It may seem that it would be hard for the attacker to succeed, but he can mount "algorithmic complexity attacks"**
  - Make a normally fast operation take very long
  - Example: Instead of creating a file `/tmp/a`, make it point to a symlink which in turn points to a symlink and so on. Access operation, which needs to resolve this sequence of symlinks will take very long. Can further slow it down by creating deep directory trees.
  - As a result, races can succeed with near 100% probability!

Common Software Vulnerabilities

- **CWE (Common Weakness Enumeration) is an excellent source on currently prevalent software vulnerabilities**
- **CWE Top-25 is a good point to start**
  - You are expected to be familiar with the vulnerabilities in this list – read the list and understand what each vulnerability means

Common Software Weaknesses

- **Input validation**
  - Injection vulnerabilities
    - Cross-site scripting, SQL/command injection, code/script injection, format-string, path-traversal, open redirect, ...
  - Buffer overflows
    - integer overflows, incorrect buffer size or bounds calculation
  - Many other application-specific effects of untrusted input
- **Failure to recognize or enforce trust boundaries**
  - Calling function that trust their inputs with untrusted data
  - Including code without understanding its dependencies
  - Relying on form data or cookies in a web application
- **Missing security operation**
  - Authentication: missing, weak, or using hard-coded credentials
  - Authorization: missing checks
    - Cross-site request forgery
  - Failure to encrypt, hash, use salt, …

Common Software Weaknesses

- **Use of weak security primitives**
  - Weak random numbers, encryption, hash algorithms, …
- **Information leakage**
  - Error messages that reveal too much information
    - Software version, source code fragments, database table names or errors, …
    - Timing channels
- **Execution with unnecessary privileges**
  - Executing code with admin privileges
  - Incorrect (or missing) permission settings
- **Error/exception-handling code**
  - Failure to check error codes, e.g., open, malloc, …
  - Failure to test error/exception-handling code
- **Race conditions**

CWE -1000: Research View of CWEs

- Top 25 is useful to understand current trends, but the descriptions can often be uninformative
- CWE-1000 organization has a much better structure and organization
- You don’t necessarily get a sense of completeness from these, but reading them will still significantly broaden your understanding of software vulnerabilities and more secure coding practices.

Secure Coding Practices

- **The goal of this course is to expose you to a range of vulnerabilities and exploits, so you can learn how to build secure systems and develop secure code**
- **But we don't necessarily provide a "cook book"**
  - The hope is that you will learn more from understanding the examples in depth than reading a long laundry list
- **Nevertheless, several good sources are available on the Internet that discuss secure coding practices**
  - CERT top 10 secure coding practices
  - CERT Secure coding standards for C, C++, and Java
  - OWASP Secure coding principles

Vulnerabilities Vs Malicious Code

- **These two pose very different threats**
  - With vulnerable code, you have a relatively weak adversary: one that is constrained to exploiting an existing vulnerability, but has no way of controlling it.
  - So, relatively weak defenses such as randomization can work.
  - With malicious code, you have a strong adversary
    - Can modify code to evade specific defenses
    - You cannot make assumptions such as the absence of intentionally introduced errors, obfuscation, etc.