1 Overview

In the last lecture we introduced the goals of Computer Security, the concept of a trusted computing base and the idea of a threat model.
This lecture introduces a few important principles of computer security.

2 Important Principles of Computer Security

Computer Security is as much art as it is science, as there is no formal method to guarantee the security of a system. Therefore, the best we can do is rely upon a set of best practices when designing and implementing secure systems. There is no exact recipe for success.

2.1 Open Design

The design and implementation details of a system should be published.

2.1.1 Security through Obscurity

A system should still be secure even if an adversary has intimate knowledge of the inner workings of the system. The security of a system should not rely on the ignorance of an attacker.

Example: Consider a system that relies on security through obscurity. When the details of the system are made known, an entirely new system with an entirely new design is required to replace the obsolete, compromised system. Creating this new system will take a substantial amount of time and effort. Contrast this with a system whose security does not depend on the secrecy of its design and implementation. A disclosure of any of the details of that system, such as passwords or private keys, can be countered by changing those details, which is a trivial undertaking.

2.1.2 Review

There are additional benefits to publishing the details of the system, beyond the benefits of designing a system to withstand public scrutiny. By opening a system to review, weaknesses in that system can be discovered and fixed. With enough eyeballs, all bugs will be found, according to Linus Torvalds.

2.2 Least Privilege

A system should give a subject, such as a process, only the minimum privileges necessary to perform its task.
Example: Consider a file server running on a Unix system. All that server requires is access to parts of the file system that you wish to share with others. Providing that file server process with root privileges would violate the principle of Least Privilege, as the file server does not require many of the privileges offered by the root user, such as the ability to mount devices, the power to shutdown the system, and the ability to manage other processes and user accounts.

2.3 Complete Mediation

Complete mediation requires that all accesses from subjects to objects be checked to ensure that the subject has the authority to access the object.

Example: The kernel ensures that all accesses to files are made by users with the appropriate privileges, even when symbolic links are used. See the addendum for an additional, related example.

2.4 Fail-Safe Defaults

The default behavior of a secure system should be to deny access, not to grant it.

Example: Consider an authentication server. If that server goes down, all users should be denied access by default, not granted access. It is safer to deny access to users who should be allowed than it is to allow access to those who should not. In addition, it is significantly more likely that legitimate users denied access will report this problem, allowing the problem to be identified and fixed faster.

Example: Consider a web server, which maintains a blacklist of files that cannot be accessed. There are many ways to specify a path to the same file. For example, there are infinitely many ways to specify all files in a user’s home directory. A few are listed.

1. /home/hchen/*
2. ~hchen/*
3. /home/.hchen/*
4. /home/./hchen/*
5. /home/././hchen/*

This is the canonicalization problem, where multiple representations refer to the same file. A better approach is to maintain a whitelist of files that can be accessed, instead of a blacklist. By default, an attempt to access a file not on the whitelist will be denied.

3 Addendum: Race Conditions

Example: Consider a file server which accepts requests from a user to open a file, checks that the user has permissions to open that file, and finally opens the file if the user has sufficient authority. A portion of the code for that server is reproduced below.
st = stat(filename);
if (st.uid == uid && ...) {
    open(filename, ...);
}

The filename given to the file server may refer to a symbolic link. If this is the case, the file server will check the permissions of the destination file, not the permissions of the link itself. This introduces a race condition. If the symbolic link originally points to a file the user has permission to access, it is possible for that user to gain access to other files by changing the destination of the symbolic link after the stat call but before the open command.

Bugs such as the one in the above code are also known as time of check to time of use bugs, or TOCTTOU. The success rate of the attacker can be reduced exponentially by calling stat multiple times and comparing the values of each call to ensure that the file has not been modified. A better solution is to use a subprocess with the privileges of the user, and then call open in that subprocess. This delegates the permission checking to the kernel, completely avoiding the race condition.