When designing systems and applications, it is important to keep security front-of-mind. Passwords should *never* be stored in plain text. Passwords should be stored in a one-way hash mechanism.

User input should be sanitized. This will help eliminate the possibility of an external attack that can take control of your system. Instead of blindly trusting your **SQL** or command queries, build them as stored procedures or functions which have well-controlled options.

You should learn and make use of the newer security frameworks. Many people, when they run into an application fighting against something like **SELinux**, opt to turn it off. A better choice would be to figure out what is causing the error and fix it. Besides **SELinux**, another security frameworks you can use is **AppArmor**.

Awareness is one of the best ways to combat insecurity. Security-related mailing lists like **CVE** (http://cve.mitre.org/) and the **CERT-Tech Alerts** (http://www.us-cert.gov/cas/techalerts/) help to inform system and network administrators of currently known vulnerabilities.

It also helps to think like a bad guy. You should have a non-production lab you attempt to break into, do security drills, and use resources like the ***2600 Magazine: The Hacker Quarterly*** to see how someone could infiltrate your network.

Distribution errata is another security concern.

You should never underestimate the power of human nature. Be aware of **phishing** scams or other types of social engineering attacks. You should continuously train your users on how to avoid falling prey to these attacks. You should always be aware of the **layer 8** (human) errors.

Network inspection includes packet sniffers, network scanners, and intrusion detection systems.

The tools to help secure your network include:

- **nmap**:
  A port scanner used for detecting open or listening network ports remotely.
- **tcpdump/Wireshark/pcap**:
  Tools which listen to traffic "on the wire" to help diagnose network issues.
- **SNORT**:
  A network-based intrusion detection system (**IDS**). **Snort** listens on the network for traffic signatures of known exploits.
- **Tripwire**:
  A host-based intrusion detection system. **Tripwire** keeps a database of your system state and alerts if this changes without your knowledge.

Network firewalls provide protection to your server from the outside world. They can be implemented as a service running on your server (**Netfilter**). Network firewalls can also be implemented as an appliance built with specialized hardware.

A good network design combines a front-end firewall (either a software or hardware appliance), as well as per-host firewalls. The important part of creating these layers is to keep them complex enough to protect without being burdensome to configure/change.

Tcpwrappers is an application-layer firewall which started as an **ACL** (**A**ccess **C**ontrol **L**ist) system for the **inet** daemon. Now, any tool which links against the **libwrap** library can have an application-layer firewall.

Netfilter, or **iptables**, is a software-based network firewall which exists in the **Linux** kernel.

Applications themselves also have security tools in the form of **ACL**s or configuration options.

You should secure your applications with daemon control over subsets of protocol and user/password control. Certain daemons can also make use of the knowledge of a given protocol to block only parts of it. For example, the **Apache** daemon can block per-method access to the web server (denying **POST**).

The **Linux** system uses a framework for authentication called **PAM** (**P**luggable **A**uthentication **M**odule).

Another security tool is a jail root (`chroot`), or restricted root. This tool limits the damage a compromised daemon can cause.

14.12. Chroot

The **`chroot()`** system call will change the apparent root directory of a process. The root user is the only user allowed to make the **`chroot()`** system call. This limits the damage that can be done by broken or malicious software. **Chroot Jails** (as the new root environments are called) can then be used for many purposes:

- Building software:
  Eliminates dependency poisoning, every build can have the exact needed dependencies.
- Securing network daemons:
  Suspect system daemons can be run in a *jail*, thus limiting the damage any security breach may cause. Repair systems can **`bootstrap`** and **`chroot`** into a broken system.
- Testing software:
  Installing and testing software in a *jail* will protect the parent system from the outcome of bugs.

14.2. Learning Objectives

By the end of this session, you should be able to:

- **Discuss concepts of network security.**
- **Examine tools and facilities to implement network security.**

Having multiple, diverse layers of security is a good idea. It will help protect your network or systems if there is a failure in one of the layers. The diversity of the layers is also important: you are only as strong "as your weakest link". If you have multiple layers of the same type of security, a single bug or breach may bypass all of them at once.

Strong logging and auditing will help to mitigate problems when they do occur. Make sure your auditing system has checks and balances to avoid manipulation by an unwanted party. Log to a remote server if possible. Always think like a bad guy.

A common issue facing network security is the failure to follow the "Principle of least privilege". Not everyone needs full super-user access. Limit access using tools such as **sudo** or **su**.

Be aware of the "confused deputy" problem, which is an example of a privileged escalation attack.

1. A user makes a request to a service on a network or host (the deputy), and specifies an output file they normally would not have access to.
2. The deputy, with its increased or different access control, modifies the file the user would not have been able to modify directly.
3. The user reaps the benefits of whatever the modification was.

Start your security mindset in a "mostly closed" paradigm. Restrict all access, only opening those things which need to be open. This will protect you if you inadvertently enable a service which should not be.

Token or key-based access is also a better bet than passwords. Passwords are easily guessed, or brute-force attacked. Cryptographically-strong keys are less likely to be compromised in that manner.

Given the choice between a protocol which is encrypted, and one that is not, you should choose the encrypted one.

It is also a good idea to log not only failures in authentication/authorization, but successes as well. This helps give you a proper audit trail if the "successful" login was an intrusion.

Good security practices include:

- Use encrypted protocols if possible. If this is not possible, use digest or other non-plain-text modes.
- Log both success **AND** failure.
- Restrict access to users that need it.

A **demilitarized zone (DMZ)** is a special-purpose network housing business critical servers which need access to a large untrusted network.

When setting up a **DMZ**, proper auditing is very important. Firewalls can be set between the **DMZ** and the external *untrusted* network, as well as between the **DMZ** and the internal *trusted* network. Be aware that hosts can also fall prey to the confused deputy problem.

With a **DMZ**, you must remember the following:

- It contains the least trusted services in a network.
- You should monitor and control inbound access.
- You should monitor and control outbound access.