Linux System Security

- The first step to sound system security is to deploy mechanisms for **controlling** and **authenticating** access to servers and services.

- Port security is also critical to network security. Become very familiar with all the ports and services on your system.

- Problems related to security and access can come from outside or inside. These can be malicious in intent, the result of "just looking around", or accidental.

- It is important to verify the cause and intent of the intrusion by careful examination of log files.

Pluggable Authentication Module (PAM)

- PAM is a suite of shared libraries that enable the local system administrator to choose how applications authenticate users.

- It is a generic authentication tool that enables implementation of specific rules without rewriting and recompiling individual services.

- Security conscious programs such as ***login***, ***su,*** or ***ftp*** are configured by PAM to perform a variety of authentication and security checks.

- PAM looks at its configuration file in ***/etc/pam.d*** to determine how to authenticate the user requesting a particular service.

- It may authenticate by checking ***/etc/passwd***, ***kerberos***, or **NIS**. PAM's rulesets for the particular service will determine whether or not the user is authenticated.

- The local configuration of those aspects of system security controlled by PAM is contained in one of two places: either the single system file, ***/etc/pam.conf***; or the ***/etc/pam.d/*** directory.

- The ***/etc/pam.d/*** directory contains one file for every PAM-aware service. Each file contains rules and the paths to the authentication modules that are used.

- Lines that start with the word **auth** define a path to the module to be used for authentication.

- Lines that contain the word **account** identify the modules that do all the necessary accounting, such as determining is he given password has expired.

- Lines that contain the word **password** specify a module that is used to handle tasks such as password verification and changing.

- Lines that contain the word **required** mean that the module's verification must pass.

- If a line contains the word **requisite** and that module fails verification, PAM will automatically deny authentication.

- The following is a sample PAM file, ***/etc/pam.d/login***

```
#%PAM-1.0
auth       required     /lib/security/pam_securetty.so
auth       required     /lib/security/pam_pwdb.so shadow nullok
auth       required     /lib/security/pam_nologin.so
account    required     /lib/security/pam_pwdb.so
password   required     /lib/security/pam_cracklib.so
password   required     /lib/security/pam_pwdb.so nullok use_authtok
md5 shadow
session    required     /lib/security/pam_pwdb.so
session    optional     /lib/security/pam_console.so
```

- The ***/lib/secur***ity directory contains all the security library files. Read the documentation for PAM in the ***/usr/doc*** directory.

- PAM uses ***/etc/securetty*** to authenticate user's **login**, **exec**, and **rlogin** requests. This file lists **ttys** on which root is allowed to log in. The default list is the system's virtual console **ttys.**

- Restricting root login to virtual consoles makes it very difficult for anyone to gain root privileges using a dial-in or network connection, since root must physically be at the terminal.

- This mechanism however does not prevent anyone from logging in as a user and then issuing a ***su - root*** command.

Network Information Service (NIS)

- NIS is a lightweight directory service that manages and distributes databases of users, passwords, groups, hosts and other data from a master server.

- NIS uses a client-server architecture to distribute information from the following files on the master server by default:

  /etc/passwd
  /etc/group
  /etc/hosts
  /etc/ethers
  /etc/networks
  /etc/netgroup
  /etc/protocols
  /etc/services

- NIS allows remote user authentication. An NIS master server keeps a copy of the password and group files, and optionally distributes them to secondary slave servers.

- When a user logs in on a network machine, the login request is authenticated remotely by the master server or one of its slaves.
- NIS requires that the RPC PortMapper ***/usr/sbin/portmap*** be running. RPC PortMapper servers convert RPC program numbering to TCP or UDP port numbers.

- The hosts line in ***/etc/nsswitch.conf*** file specifies the hostname lookup order. The default configuration of files nisplus nis dns indicates that the lookups will be performed in the following order:

    - ***/etc/hosts***
    - NIS+lookup
    - NIS lookup
    - DNS using ***/etc/resolv.conf*** and ***named***

- Check out the **/usr/doc/HOWTO/NIS-HOWTO** for more information.

- **NIS client Programs:**

    - ***ypbind***: finds and stores (in ***/var/yp/binding***) information about an NIS domain and server.
    - ***ypwhich***: returns the name of the NIS server used by the NIS client.
    - ***ypcat***: prints the values of all keys in the specified NIS database.
    - ***yppoll***: returns the version and master server of the specified NIS map.
    - ***ypmatch***: prints the values of keys from an NIS map.
    - ***yppasswd***: user command for changing an NIS password.

- **NIS server** uses all the NIS client programs plus:

    - ***makedbm***: translates ASCII databases, such as /***etc/passwd***, to NIS databases.
    - ***yppush***: notifies the slave servers of changes to the master databases.
    - ***ypserv***: the NIS server daemon.
    - ***yppasswdd***: the NIS daemon that handles NIS password changes.

TCP_wrappers

- TCP Wrappers is one of the most common methods of access control on a Unix system.

- A wrapper program 'wraps' around existing daemons and interfaces between clients and the server.

- Essentially this is a mechanism that provides a layer of security for daemons started from ***xinetd***.

- Any network services managed by **xinetd** (as well as any program with built-in support for libwrap) can use TCP wrappers to manage access.

- xinetd can use the **/etc/hosts.allow** and **/etc/hosts.deny** files to configure access to system services.

- As the names imply, hosts.allow contains a list of rules clients allowed to access the network services controlled by xinetd, and hosts.deny contains rules to deny access.

- The hosts.allow file takes precedence over the hosts.deny file. Permissions to grant or deny access can be based on individual IP address (or hostnames) or on a pattern of clients.

- The wrapper ***tcpd*** checks its configuration **/etc/hosts.allow** and **/etc/hosts.deny** to decide whether or not to allow the service to be provided.

- As a system administrator you can configure the rules in these files to allow or deny each **x*inetd***-managed service to specific hosts, domains, or subnets.

- The diagram show below illustrates the relationship between **services**, **tcp_wrappers** and **xinetd**.

- When a connection is initiated on one of the ports **x*inetd*** is bound to, **x*inetd*** looks up the service name by its port number in ***/etc/services***, then the configuration of that service in ***/etc/xinetd.conf.***

- If ***tcpd*** is listed as the server executable to run, its argument (the daemon's executable name) is looked up first in **/etc/hosts.allow**. If access is permitted by rules in that file, access to the service is allowed.

- Then, if access is denied by the rules in **/etc/hosts.deny,** the connection is dropped. If no rule matches in either file, access to the service is allowed.

- The parsing daemon stops at the first match and inspects the following files:

    1. hosts.allow
    2. hosts.deny
    3. default: Allow

- The default configuration of **/etc/hosts.allow** and **/etc/hosts.deny** is to allow all server requests. You must edit these files to restrict access.

- A typical and very simple configuration is to deny ALL services to ALL, but allow ALL services to hosts on your local subnet.

- **Permissive Strategy**

    - Leave **hosts.allow** empty the configure **hosts.deny**.

- **Paranoid Strategy**

    - In **hosts .deny** deny ALL, i.e., **ALL:ALL**
    - Then configure **hosts.allow** to permit access.

- The following is a sample of **/etc/hosts.deny** showing the syntax:

  ALL: bcit.ca
  in.fingerd: 142.232.66.1/255.255.255.192
  in.ftpd: ALL EXCEPT LOCAL

- The following are important points to consider when using TCP wrappers to protect network services:

  - Because access rules in hosts.allow are applied first, they take precedence over rules specified in hosts.deny.

  - Therefore, if access to a service is allowed in hosts.allow, a rule denying access to that same service in hosts.deny is ignored.

  - Since the rules in each file are read from the top down and the first matching rule for a given service is the only one applied, the order of the rules is extremely important.

  - If no rules for the service are found in either file, or if neither file exists, access to the service is granted.

  - TCP wrapped services do not cache the rules from the hosts access files, so any changes to hosts.allow or hosts.deny take effect immediately without restarting network services.

Formatting Access Rules

- The format for both **/etc/hosts.allow** and **/etc/hosts.deny** are identical. Any blank lines or lines that start with a hash mark (#) are ignored, and each rule must be on its own line.

- Each rule uses the following basic format to control access to network services:

```
<daemon list>: <client list> [: <option>: <option>: ...]
```

- *<daemon list>* — A comma separated list of process names (*not* service names) or the ALL wildcard.

- *<client list>* — A comma separated list of hostnames, host IP addresses, special *patterns* or special wildcards that identify the hosts effected by the rule.

- *<option>* — An optional action or colon separated list of actions performed when the rule is triggered. Option fields support **expansions**), launch shell commands, allow or deny access, and alter logging behavior.

- The following is a basic sample hosts access rule:

```
vsftpd : .example.com
```

- This rule instructs TCP wrappers to watch for connections to the FTP daemon (vsftpd) from any host in the example.com domain.

- If this rule appears in hosts.allow, the connection will be accepted. If this rule appears in hosts.deny, the connection will be rejected.

- The next sample hosts access rule is more complex and uses two option fields:

```
sshd : .example.com  \
: spawn /bin/echo `/bin/date` access denied>>/var/log/sshd.log \
: deny
```

- The above rule states that if a connection to the SSH daemon (sshd) is attempted from a host in the **example.com** domain, execute the echo command (which will log the attempt to a special file), and deny the connection.

- Because the optional deny directive is used, this line will deny access even if it appears in the hosts.allow file.

Wildcards

- Wildcards allow TCP wrappers to more easily match groups of daemons or hosts. They are used most frequently in the client list field of access rules.

- The following wildcards may be used:

    - ALL — Matches everything. It can be used for both the daemon list and the client list.

    - LOCAL — Matches any host that does not contain a period (.), such as localhost.

    - KNOWN — Matches any host where the hostname and host address are known or where the user is known.

    - UNKNOWN — Matches any host where the hostname or host address are unknown or where the user is unknown.

    - PARANOID — Matches any host where the hostname does not match the host address.

Patterns

- Patterns can be used in the client list field of access rules to more precisely specify groups of client hosts.

- The following is a list of the most common accepted patterns for a client list entry:

    - **Hostname beginning with a period (.)** — Placing a period at the beginning of a hostname, matches all hosts sharing the listed components of the name.

    - The following example would apply to any host within the example.com domain:

    ```
    ALL : .example.com
    ```

    - IP address ending with a period (.) — Placing a period at the end of an IP address matches all hosts sharing the initial numeric groups of an IP address.

    - The following example would apply to any host within the 192.168.x.x network:

    ```
    ALL : 192.168.
    ```

    - IP address/netmask pair — Netmask expressions can also be used as a pattern to control access to a particular group of IP addresses.

    - The following example would apply to any host with an address of 192.168.0.0 through 192.168.1.255:

    ```
    ALL : 192.168.0.0/255.255.254.0
    ```

    - The asterisk (*) — Asterisks can be used to match entire groups of hostnames or IP addresses, as long as they are not mixed in a client list containing other types of patterns.

    - The following example would apply to any host within the `example.com` domain:

    ```
    ALL : *.example.com
    ```

    - The slash (/) — If a client list begins with a slash, it is treated as a file name.

    - This is useful if rules specifying large numbers of hosts are necessary. The following example refers TCP wrappers to the /etc/telnet.hosts file for all Telnet connections:

```
in.telnetd : /etc/telnet.hosts
```

Operators

- At present, access control rules accept one operator, **EXCEPT**. It can be used in both the daemon list and the client list of a rule.

- The **EXCEPT** operator allows specific exceptions to broader matches within the same rule.

- In the following example from a hosts.allow file, all example.com hosts are allowed to connect to all services except cracker.example.com:

```
ALL: .example.com EXCEPT cracker.example.com
```

- In the another example from a hosts.allow file, clients from the 192.168.0.*x* network can use all services except for FTP:

```
ALL EXCEPT vsftpd: 192.168.0.
```

Logging

- Option fields let administrators easily change the log facility and priority level for a rule by using the severity directive.

- In the following example, connections to the SSH daemon from any host in the example.com domain are logged to the default **authpriv** facility (because no facility value is specified) with a priority of **emerg**:

```
sshd : .example.com : severity emerg
```

- It is also possible to specify a facility using the **severity** option.

- The following example logs any SSH connection attempts by hosts from the example.com domain to the local0 facility with a priority of alert:

```
sshd : .example.com : severity local0.alert
```

- Note that in practice, this example will not work until the **syslog daemon** (***syslogd***) is configured to log to the local0 facility.

Access Control

- Option fields also allow administrators to explicitly allow or deny hosts in a single rule by adding the allow or deny directive as the final option.

- For instance, the following two rules allow SSH connections from client-1.example.com, but deny connections from client-2.example.com:

```
sshd : client-1.example.com : allow
sshd : client-2.example.com : deny
```

- By allowing access control on a per-rule basis, the option field allows administrators to consolidate all access rules into a single file: either hosts.allow or hosts.deny.

Shell Commands

- Option fields allow access rules to launch shell commands through the following two directives:

  - **spawn — Launches a shell command as a child process**.

  - This option directive can perform tasks like using /usr/sbin/safe_finger to get more information about the requesting client or create special log files using the echo command.

  - In the following example, clients attempting to access Telnet services from the example.com domain are quietly logged to a special file:

```
in.telnetd : .example.com \
  : spawn /bin/echo `/bin/date` from %h>>/var/log/telnet.log \
  : allow
```

  - **twist — Replaces the requested service with the specified command**.

  - This directive is often used to set up traps for intruders (also called "honey pots").

  - It can also be used to send messages to connecting clients. The twist command must occur at the end of the rule line.

  - In the following example, clients attempting to access FTP services from the example.com domain are sent a message via the echo command:

```
vsftpd : .example.com \
: twist /bin/echo "Rabbit Hole Ahead, Turn Back!"
```

Expansions

- Expansions, when used in conjunction with the spawn and twist directives provide information about the client, server, and processes involved.

- The following is a list of supported expansions:

  %a — The client's IP address.
  %A — The server's IP address.
  %c — Supplies a variety of client information, such as the username and hostname, or the username and IP address.
  %d — The daemon process name.
  %h — The client's hostname (or IP address, if the hostname is unavailable).
  %H — The server's hostname (or IP address, if the hostname is unavailable).
  %n — The client's hostname. If unavailable, unknown is printed. If the client's hostname and host address do not match, paranoid is printed.
  %N — The server's hostname. If unavailable, unknown is printed. If the server's hostname and host address do not match, paranoid is printed.
  %p — The daemon process ID.
  %s — Various types of server information, such as the daemon process and the host or IP address of the server.
  %u — The client's username. If unavailable, unknown is printed.

- The following sample rule uses an expansion in conjunction with the spawn command to identify the client host in a customized log file.

- It instructs TCP wrappers that if a connection to the SSH daemon (sshd) is attempted from a host in the **example.com** domain, execute the echo command to log the attempt, including the client hostname (using the %h expansion), to a special file:

```
sshd : .example.com  \
: spawn /bin/echo `/bin/date` access denied to %h>>/var/log/sshd.log \
: deny
```

- Similarly, expansions can be used to personalize messages back to the client.

- In the following example, clients attempting to access FTP services from the `example.com` domain are informed that they have been banned from the server:

```
vsftpd : .example.com \
: twist /bin/echo "%h has been banned from this server!"
```

The *xinetd* Super Server

- The **extended Internet services daemon** - ***xinetd*** - provides many of the capabilities seen with **TCP_wrappers** and the **portmapper** and adds some capabilities not incorporated into those utilities.

Advantages Of *xinetd*

- ***xinetd*** provides access control in a way that is quite similar to TCP_wrappers or the portmapper.
- It provides access control for TCP, UDP, and RPC services.
- Implements access limitations based on time.
- Extensive logging capabilities for both successful and unsuccessful connections.
- Provides transparency to both the client host and the wrapped network service. Both the connecting client and the wrapped network service are unaware that TCP wrappers are in use.
- Legitimate users are logged and connected to the requested service while connections from banned clients fail.
- Centralized management of multiple protocols. — TCP wrappers operate separately from the network services they protect, allowing many server applications to share a common set of configuration files for simpler management.
- Provides for **hard reconfiguration** by killing services that are no longer allowed.
- Provides numerous mechanisms to prevent **DoS** attacks.
  - Limit on the number of daemons of a given type that can run concurrently.
  - An overall limit of processes forked by ***xinetd***.
  - Limits on log file sizes.
- Provides a compile-time option to include ***libwrap***, the TCP_wrappers library.
  - Causes ***/etc/hosts.allow*** and ***/etc/hosts.deny*** access control checks in addition to ***xinetd*** access control checks.
- Provides for the invocation of ***tcpd***.
  - All TCP-wrappers functionality is available.
- Services may be bound to specific interfaces.
- Services may be forwarded (**proxied**) to another system.

Disadvantages of *xinetd*

- The configuration file, ***/etc/xinetd.conf***, is incompatible with the older ***/etc/inetd.conf***.
    - You can however use a conversion utility, ***xtoa***, that is included with the distribution.
- Time-outs and other problems occur for RPC services, especially on busy systems.
    - However, ***xinetd*** and ***portmap*** may coexist, allowing RPC through the **portmapper**.

- The configuration file for ***xinetd*** is **/etc/xinetd.conf**, but the default file only contains a few defaults and an instruction to include the **/etc/xinetd.d** directory.
- To enable or disable a xinetd service, edit its configuration file in the /etc/xinetd.d directory.
- If the disable attribute is set to **yes**, the service is disabled. If the disable attribute is set to **no**, the service is enabled.
- If you edit any of the xinetd configuration files or change its enabled status using **Serviceconf**, **ntsysv**, or chkconfig, you must restart xinetd with the command service xinetd restart before the changes will take effect.

The *xinetd* Configuration File

- The ***xinetd*** configuration file is, by default, ***/etc/xinetd.conf***. Its syntax is quite different than, and incompatible with, ***/etc/inetd.conf***.
- Essentially, it combines the functionality of ***/etc/inetd.conf***, ***/etc/hosts.allow*** and ***/etc/hosts.deny*** into one file. Each entry in ***/etc/xinetd.conf*** is of the form:

```
service service-name
{
        attribute operator value value ...
        ….
}
```

- **service** is a required keyword and the braces must surround the list of attributes.
- The ***service*** name is arbitrary, but is normally chosen from the standard network services.
- Additional and completely nonstandard services may be added, as long as they are invoked through a network request, including network requests from the **localhost** itself.

- There are a number of **attributes** available, which are described in detail in the man pages. The **values** are the parameters set to the given attribute.

- We will also discuss some of the more common ones together with the allowed values next.

- The operator may be one of or =, +=, or -=. All attributes may use = which has the effect of assigning one or more values.

- Some attributes may use the forms, += and/or -= which have the effect of adding to an existing list of values or removing from an existing list of values, respectively.

<u>**Attributes and Values**</u>

- ***socket_type***

    - The type of TCP/IP socket used. Acceptable values are ***stream*** (TCP), ***dgram*** (UDP), ***raw***, and ***seqpacket*** (reliable, sequential datagrams).

- ***protocol***

- Specifies the protocol used by the service. Must be an entry in /etc/protocols. If not specified, the default protocol for the service is used.

- ***server***

    - Daemon to invoke. Must be absolutely qualified.

- ***server-args***

    - Specifies the flags to be passed to the daemon.

- ***port***

    - Port number associated with the service. If listed in /etc/services, it must match.

- ***wait***

    - There are two possible values for this attribute. If yes, then xinetd will start the requested daemon and cease to handle requests for this service until the daemon terminates. This is a single-threaded service.

    - If no, then xinetd will start a daemon for each request, regardless of the state of previously started daemons. This is a multithreaded service.

- ***user***
  - Sets the UID for the daemon. This attribute is ineffective if the effective UID of xinetd is not 0.
- ***nice***
  - Specifies the nice value for the daemon.
- ***id***
  - Used to uniquely identify a service when redundancy exists.
  - For example, echo provides both dgram and streams services.
  - Setting ***id=echo_dgram*** and ***id=echo_streams*** would uniquely identify the **dgram** and **streams** services, respectively.
  - If not specified, id assumes the value specified by the **service** keyword.
- ***access_times***
  - Sets the time intervals for when the service is available. Format is **hh:mmhh:mm**; for example, **08:00-18:00** means the service is available from 8 A.M. through 6 P.m.
- ***only_from***
  - Space-separated list of allowed clients. The syntax for clients is as follows:
  - ***hostname***
    - A resolvable hostname. All IP addresses associated with the hostname will be used.
  - ***IPaddress***
    - The standard IP address in dot decimal form.
  - ***net_name***
    - A network name from ***/etc/networks***.

        - ***x.x.x.0, x.x.0.0, x.0.0.0, 0.0.0.0***
            - The 0 is treated as a wildcard. For example, an entry like 88.3.92.0 would match all addresses beginning with 88.3.92.0 through and including 88.3.92.255. The 0.0.0.0 entry matches all addresses.
        - ***x.x.x. {a, b, ….}, x.x.{a, b, …}, x. {a, b, …}***
            - Specifies lists of hosts. For example, 172.19.32.{1, 56, 59} means the list of IP addresses, 172.19.32.1, 172.19.32.56, and 172.19.32.59.
        - ***IPaddress/netmask***
            - Defines the network or subnet to match. For example, 172.19.16.0/20 matches all addresses in the range 172.19.16.0 through and including 172.19.31.255.
    - If this attribute is specified without a value, it acts to deny access to the service..

- ***no_access***
    - Space separated list of denied clients. The syntax for clients is given previously.

- ***redirect***
    - This attribute assumes the syntax, ***redirect= IPaddress port***.
    - It has the effect of redirecting a TCP service to another system. The server attribute is ignored if this attribute is used.

- ***bind***
    - Binds a service to a specific interface. Syntax is ***bind = IPaddress.*** This allows hosts with multiple interfaces (physical or logical), for example, to permit specific services (or ports) on one interface but not the other.

- ***log_on_success***
    - Specifies the information to be logged on success. Possible values are:

      PID: PID of the daemon. If a new daemon is not forked, PID is set to 0.

      HOST: Client host IP address.

      USERID: Captures UID of the client user through an RFC1413 call. Available only for multithreaded, streams services.

      EXIT: Logs daemon termination and status.

      DURATION: Logs duration of session. By default, nothing is logged. This attribute supports all operators.

- ***log_on_failure***

    - Specifies the information to be logged on failure. A message indicating the nature of the error is always logged. Possible values are:

        ATTEMPT:Records a failed attempt. All other values imply this one.

        HOST: Client host IP address.

        USERID: Captures UID of the client user through an RFC1413 call. Only available for multithreaded, streams services.

        RECORD: Records additional client information such as local user, remote user, and terminal type.

    - By default, nothing is logged. This attribute supports all operators.

- ***disabled***

    - It has the same effect as commenting out the service entry in the ***/etc/xinetd.conf*** file. Syntax is ***disabled = yes***. If left out, the service is enabled.

- Consider a simple example. The attributes (everything inside the braces and to the left of the = symbol) are very straightforward in their meaning as are the associated values (everything inside the braces and to the right of the = symbol):

```
service ftp
{
        socket_type  =   stream
        protocol     =   tcp
        wait         =   no
        user         =   root
        server       =   /usr/sbin/in.ftpd
        server-args  =   -l -a
}



service telnet
{
        socket_type =    stream
        protocol    =    tcp
        wait        =    no
        user        =    root
        server      =    /usr/sbin/in.telnetd
}
```

The *defaults* entry

- The purpose of the defaults entry in the ***/etc/xinetd.conf*** file is to specify default values for all services in the file.
- These default values may be overridden or modified by each individual service entry.
- The following is an example of defaults entry as it might appear in ***/etc/xinetd.conf***.

```
defaults
{
        log_type         = SYSLOG local4 info
        log_on_success = PID HOST EXIT DURATION
        log_on_failure   = HOST
        instances        = 8
        disabled         = in.tftdp in.rexecd
}
```

- The above defaults specify that for all services, log messages will be sent to the ***syslogd*** daemon via the local4. info selector.

- Successful connections to services will cause the PID, client IP address, termination status, and time of connection to be logged.
- Unsuccessful connection attempts will have the client IP address logged. The maximum number of instances for any one service is set to eight. Two services, ***in.tftpd*** and ***in.rexecd*** are disabled.

- The Red Hat distribution puts a link in the ***xinetd.conf*** file to a directory where all the individual service files are configured and kept:

```
#
# Simple configuration file for xinetd
#
# Some defaults, and include /etc/xinetd.d/

defaults
{
      instances            = 60
      log_type             = SYSLOG authpriv
      log_on_success       = HOST PID
      log_on_failure       = HOST RECORD
}
includedir /etc/xinetd.d
```

- The individual services are then configured using individual files in the **/etc/xinetd.d** directory.

- For example, the **Telnet** service can be configured using a file called ***telnet***, the contents of which will be as follows (note that is turned on by default):

```
# default: on
# description: The telnet server serves telnet sessions; it uses \
#  unencrypted username/password pairs for authentication.
service telnet
{
   flags       = REUSE
   socket_type = stream
   wait        = no
   user        = root
   server      = /usr/sbin/in.telnetd
   log_on_failure    += USERID
}
```

- The ***tftp*** service file might contain (note that the service is turned off):

```
# default: off
# description: The tftp server serves files using the trivial file
# transfer
#  protocol.  The tftp protocol is often used to boot diskless
#  workstations, download configuration files to network-aware
#  printers,
#  and to start the installation process for some operating systems.
service tftp
{
   socket_type       = dgram
   wait              = yes
   user              = nobody
   log_on_success    += USERID
   log_on_failure    += USERID
   server            = /usr/sbin/in.tftpd
   server_args       = /tftpboot
   disable           = yes
}
```

- Here is another example where the ***imap*** service is off by default but it is explicitly turned on in the service file:

```
# default: off
# description: The IMAP service allows remote users to access their
mail using \
#              an IMAP client such as Mutt, Pine, fetchmail, or
Netscape \
#              Communicator.
service imap
{
   socket_type       = stream
   wait              = no
   user              = root
   server                  = /usr/sbin/imapd
   log_on_success          += DURATION USERID
   log_on_failure          += USERID
   disable
```

Access Control

- Consider a simple access control example below.

```
defaults
{
     log_type           =  SYSLOG local4 info
     log_on_success  =  PID HOST EXIT DURATION
     log_on_failure   =  HOST
     instances          =  8
}
service login
{
s          Socket_type        =  stream
     protocol            =  tcp
     wait                   =  no
     user                   =  root
     flags                  =  REUSE
     only_from         =  142.0.0.0
     no_access         =  142.100.0.0
     log_on success  += USERID
     log_on failure    += USERID
     server               =  /usr/sbin/in.ftpd
     server_args       = -l -a
}
```

- The configuration specifies a login service entry for a server (**milliways** for example) that allows access from any system whose IP address begins with **142** except for those whose address begins with **142.100**.

Using the ***bind*** Attribute

- The ***bind*** attribute allows for associating a particular service with a specific interface's IP address.

- Suppose we have an internal ftp server that supplies read-only resources for company employees via anonymous ftp.

- This ftp server has two interfaces, one that attaches to the corporate environment and the other that connects to a private internal network that is generally accessible only to employees who work in that particular group.

- We can implement the desired functionality using the following entries in */etc/xinetd.conf*:

```
defaults
{
      log_type            =  SYSLOG local4 info
      log_on_success      =  PID HOST EXIT DURATION
      log_on_failure      =  HOST
      instances           =  8
}

service ftp
{
      id                   =  ftp
      socket_type         =  stream
      protocol            =  tcp
      wait                =  no
      user                =  root
      only_from           =  172.17.0.0 172.19.0.0/20
      bind                =  172.17.1.1       # widget
      log_on_success      += USERID
      log_on_failure      += USERID
      server              =  /usr/sbin/in.ftpd
      server_args         =  -l -a
}

service ftp
{
      id                   =  ftp_chroot
      socket_type         =  stream
      protocol            =  tcp
      wait                =  no
      user                =  root
      bind                =  24.170.1.218  # widget server
      log_on_success      += USERID
      log_on_failure      += USERID
      access_times        =   8:30-1:30 13:00-18:00
      log_on_success      += USERID
      log_on_failure      += USERID
      server              =  /usr/sbin/anon/in.aftpd
}
```

- Each of the two ftp service entries has a unique id attribute.

- There is no limit to the number of services with the same name as long as each has a unique identifier.

- In this case, we set the id attribute to **ftp** for the internal ftp server and **ftp-chroot** for the external anonymous server.

- Note that in the latter case the daemon invoked is ***/usr/sbin/anon/in.aftpd*** (this is the effective equivalent to twist for TCP-wrappers), which is different than the former service.

- The use of ***bind*** in each case will allow packets destined only to that interface to invoke the indicated daemon.

- Thus, we can reach the anonymous server by executing ***ftp widget-srvr***. Since there are no access controls for that service, everyone has access.

- However, access is granted only between 8:30 A.M. through 11:30 A.M. and I P.M. through 6 P.m. due to the ***access_times*** attribute.

- On the other hand, executing ***ftp widget*** will be successful only if the first two octets of the client IP address begin with 172.17 or the request comes from an address in the range 172.19.0.0 through 172.19.15.255 because of the ***only_from*** attribute.

Using the *redirect* Attribute

- The ***redirect*** attribute provides a method for proxying a service through a server.

- In other words, the user may telnet to a particular server running ***xinetd***, and that server would open another connection to a different system.

- This can implemented this with the following entries in an ***/etc/xinetd.conf*** file:

```
service telnet
{
        socket_type      =  stream
        wait             =  no
        flags            = REUSE
        user             =  root
        bind             = 172.17.33.111
        log_on_success   =  PID HOST EXIT DURATION USERID
        log_on_failure   =  RECORD HOST
}

service telnet
{
        socket_type      =  stream
        wait             =  no
        flags            =  REUSE
        user             =  root
        bind             =  201.171.99.99
        redirect         =  172.17.1.1 23
        log_on_success   =  PID HOST EXIT DURATION USERID
        log_on_failure   =  RECORD HOST
}
```

- The ***REUSE*** attribute is essential since the server will be terminating and restarting continuously.

- The ***bind*** attribute has the following effect. If the command ***telnet 172.17.33.111*** is used, the connection will be made to the server itself.

- If, on the other hand, the command ***telnet 201.171.99.99*** is used, then the connection will be forwarded to **172.17.1.1** on port **23** (the telnet port).

Incorporating TCP_wrappers

- Including **TCP_wrappers** functionality in ***/etc/xinetd.conf*** is very simple.

- Wherever ***/usr/sbin/tcpd*** is set as the value for the **attribute server**, that service will be wrapped.

- The following examples illustrate this:

```
defaults
{
        log_type            =  SYSLOG local4 info
        log_on_success      =  PID HOST EXIT DURATION
        log_on_failure      =  HOST
        instances           =  8
}

service ftp
{
        id                  =  ftp_chroot
        socket_type         =  stream
        protocol            =  tcp
        wait                =  no
        user                =  root
        only_from           =  172.17.0.0
        log_on_success      += USERID
        log_on_failure      += USERID
        access_times        =  8:30 - 6:30
        server              =  /usr/sbin/tcpd
        server_args         =  /usr/sbin/in.ftpd -l -a
}
```

```
service telnet
{
      socket_type       =  stream
      wait              =  no
      flags             =  NAMEINARGS REUSE
      user              =  root
      bind              = 172.17.33.111
      server            =  /usr/sbin/tcpd
      server_args       =  /usr/sbin/in.telnetd
      log_on_success    =  PID HOST EXIT DURATION USERID
      log_on_failure    =  RECORD HOST
}
```

- The xinetd hosts access control differs from the method used by TCP wrappers. While TCP wrappers places all of the access configuration within two files, /etc/hosts.allow and /etc/hosts.deny, each service's file in /etc/xinetd.d can contain its own access control rules.

- The following hosts access options are supported by xinetd:

    - **only_from** — Allows only the specified hosts to use the service.

    - **no_access** — Blocks listed hosts from using the service.

    - **access_times** — Specifies the time range when a particular service may be used. The time range must be stated in 24-hour format notation, HH:MM-HH:MM.

- The only_from and no_access options can use a list of IP addresses or host names, or can specify an entire network.

- Like TCP wrappers, combining xinetd access control with the enhanced logging configuration can enhance security by blocking requests from banned hosts while verbosely record each connection attempt.

- For example, the following /etc/xinetd.d/telnet file can be used to block Telnet access from a particular network group and restrict the overall time range that even allowed users can log in:

```
service telnet
{
        disable         = no
        flags           = REUSE
        socket_type     = stream
        wait            = no
        user            = root
        server          = /usr/sbin/in.telnetd
        log_on_failure  += USERID
        no_access       = 10.0.1.0/24
        log_on_success  += PID HOST EXIT
        access_times    = 09:45-16:15
}
```

- In the above example, when a client system from the 10.0.1.0/24 network, such as 10.0.1.2, tries access the Telnet service, it will receive the following message:

```
Connection closed by foreign host.
```

- In addition, their login attempt is logged in /var/log/secure as follows:

```
May 15 17:38:49 boo xinetd[16252]: START: telnet pid=16256
from=10.0.1.2
May 15 17:38:49 boo xinetd[16256]: FAIL: telnet address from=10.0.1.2
May 15 17:38:49 boo xinetd[16252]: EXIT: telnet status=0 pid=16256
```