NAME

hosts_access - format of host access control files

DESCRIPTION

This manual page describes a simple access control language that is based on client (host name/address, user name), and server (process name, host name/address) patterns. Examples are given at the end. The impatient reader is encouraged to skip to the EXAMPLES section for a quick introduction.

An extended version of the access control language is described in the *hosts_options*(5) document. The extensions are turned on at program build time by building with -DPROCESS_OPTIONS.

In the following text, *daemon* is the the process name of a network daemon process, and *client* is the name and/or address of a host requesting service. Network daemon process names are specified in the inetd configuration file.

ACCESS CONTROL FILES

The access control software consults two files. The search stops at the first match:

- Access will be granted when a (daemon,client) pair matches an entry in the */etc/hosts.allow* file.
- Otherwise, access will be denied when a (daemon,client) pair matches an entry in the */etc/hosts.deny* file.
- Otherwise, access will be granted.

A non-existing access control file is treated as if it were an empty file. Thus, access control can be turned off by providing no access control files.

ACCESS CONTROL RULES

Each access control file consists of zero or more lines of text. These lines are processed in order of appearance. The search terminates when a match is found.

- A newline character is ignored when it is preceded by a backslash character. This permits you to break up long lines so that they are easier to edit.
- Blank lines or lines that begin with a ‘#’ character are ignored. This permits you to insert comments and whitespace so that the tables are easier to read.
- All other lines should satisfy the following format, things between [] being optional:

daemon_list : client_list [: shell_command]

daemon_list is a list of one or more daemon process names (argv[0] values) or wildcards (see below).

client_list is a list of one or more host names, host addresses, patterns or wildcards (see below) that will be matched against the client host name or address.

The more complex forms *daemon@host* and *user@host* are explained in the sections on server endpoint patterns and on client username lookups, respectively.

List elements should be separated by blanks and/or commas.

With the exception of NIS (YP) netgroup lookups, all access control checks are case insensitive.

PATTERNS

The access control language implements the following patterns:

- A string that begins with a '.' character. A host name is matched if the last components of its name match the specified pattern. For example, the pattern '.tue.nl' matches the host name 'wzv.win.tue.nl'.

- A string that ends with a '.' character. A host address is matched if its first numeric fields match the given string. For example, the pattern '131.155.' matches the address of (almost) every host on the Eindhoven University network (131.155.x.x).

- A string that begins with an '@' character is treated as an NIS (formerly YP) netgroup name. A host name is matched if it is a host member of the specified netgroup. Netgroup matches are not supported for daemon process names or for client user names.

- An expression of the form 'n.n.n.n/m.m.m.m' is interpreted as a 'net/mask' pair. A host address is matched if 'net' is equal to the bitwise AND of the address and the 'mask'. For example, the net/mask pattern '131.155.72.0/255.255.254.0' matches every address in the range '131.155.72.0' through '131.155.73.255'.

- An expression of the form '[n:n:n:n:n:n:n:n]/m' is interpreted as a '[net]/prefixlen' pair. A IPv6 host address is matched if 'prefixlen' bits of 'net' is equal to the 'prefixlen' bits of the address. For example, the [net]/prefixlen pattern '[3ffe:505:2:1::]/64' matches every address in the range '3ffe:505:2:1::' through '3ffe:505:2:1:ffff:ffff:ffff:ffff'.

- A string that begins with a '/' character is treated as a file name. A host name or address is

matched if it matches any host name or address pattern listed in the named file. The file format is zero or more lines with zero or more host name or address patterns separated by whitespace. A file name pattern can be used anywhere a host name or address pattern can be used.

WILDCARDS

The access control language supports explicit wildcards:

ALL
: The universal wildcard, always matches.

LOCAL
: Matches any host whose name does not contain a dot character.

UNKNOWN
: Matches any user whose name is unknown, and matches any host whose name *or* address are unknown. This pattern should be used with care: host names may be unavailable due to temporary name server problems. A network address will be unavailable when the software cannot figure out what type of network it is talking to.

KNOWN
: Matches any user whose name is known, and matches any host whose name *and* address are known. This pattern should be used with care: host names may be unavailable due to temporary name server problems. A network address will be unavailable when the software cannot figure out what type of network it is talking to.

PARANOID
: Matches any host whose name does not match its address. When tcpd is built with -DPARANOID (default mode), it drops requests from such clients even before looking at the access control tables. Build without -DPARANOID when you want more control over such requests.

OPERATORS

EXCEPT
: Intended use is of the form: ‘list_1 EXCEPT list_2’; this construct matches anything that matches *list_1* unless it matches *list_2*. The EXCEPT operator can be used in daemon_lists and in client_lists. The EXCEPT operator can be nested: if the control language would permit the use of parentheses, ‘a EXCEPT b EXCEPT c’ would parse as ‘(a EXCEPT (b EXCEPT c))’.

SHELL COMMANDS

If the first-matched access control rule contains a shell command, that command is subjected to %<letter> substitutions (see next section). The result is executed by a */bin/sh* child process with

standard input, output and error connected to */dev/null*. Specify an ‘&’ at the end of the command if you do not want to wait until it has completed.

Shell commands should not rely on the PATH setting of the inetd. Instead, they should use absolute path names, or they should begin with an explicit PATH=whatever statement.

The *hosts_options*(5) document describes an alternative language that uses the shell command field in a different and incompatible way.

% EXPANSIONS

The following expansions are available within shell commands:

%a (%A)
The client (server) host address.

%c Client information: user@host, user@address, a host name, or just an address, depending on how much information is available.

%d The daemon process name (argv[0] value).

%h (%H)
The client (server) host name or address, if the host name is unavailable.

%n (%N)
The client (server) host name (or "unknown" or "paranoid").

%p The daemon process id.

%s Server information: daemon@host, daemon@address, or just a daemon name, depending on how much information is available.

%u The client user name (or "unknown").

%%
Expands to a single ‘%’ character.

Characters in % expansions that may confuse the shell are replaced by underscores.

SERVER ENDPOINT PATTERNS

In order to distinguish clients by the network address that they connect to, use patterns of the form:

process_name@host_pattern : client_list ...

Patterns like these can be used when the machine has different internet addresses with different internet hostnames. Service providers can use this facility to offer FTP, GOPHER or WWW archives with internet names that may even belong to different organizations. See also the ‘twist’ option in the hosts_options(5) document. Some systems (Solaris, FreeBSD) can have more than one internet address on one physical interface; with other systems you may have to resort to SLIP or PPP pseudo interfaces that live in a dedicated network address space.

The host_pattern obeys the same syntax rules as host names and addresses in client_list context. Usually, server endpoint information is available only with connection-oriented services.

CLIENT USERNAME LOOKUP

When the client host supports the RFC 931 protocol or one of its descendants (TAP, IDENT, RFC 1413) the wrapper programs can retrieve additional information about the owner of a connection. Client username information, when available, is logged together with the client host name, and can be used to match patterns like:

daemon_list : ... user_pattern@host_pattern ...

The daemon wrappers can be configured at compile time to perform rule-driven username lookups (default) or to always interrogate the client host. In the case of rule-driven username lookups, the above rule would cause username lookup only when both the *daemon_list* and the *host_pattern* match.

A user pattern has the same syntax as a daemon process pattern, so the same wildcards apply (netgroup membership is not supported). One should not get carried away with username lookups, though.

- The client username information cannot be trusted when it is needed most, i.e. when the client system has been compromised. In general, ALL and (UN)KNOWN are the only user name patterns that make sense.

- Username lookups are possible only with TCP-based services, and only when the client host runs a suitable daemon; in all other cases the result is "unknown".

- A well-known UNIX kernel bug may cause loss of service when username lookups are blocked by a firewall. The wrapper README document describes a procedure to find out if your kernel has this bug.

- Username lookups may cause noticeable delays for non-UNIX users. The default timeout for username lookups is 10 seconds: too short to cope with slow networks, but long enough to irritate

PC users.

Selective username lookups can alleviate the last problem. For example, a rule like:

daemon_list : @pcnetgroup ALL@ALL

would match members of the pc netgroup without doing username lookups, but would perform username lookups with all other systems.

DETECTING ADDRESS SPOOFING ATTACKS

A flaw in the sequence number generator of many TCP/IP implementations allows intruders to easily impersonate trusted hosts and to break in via, for example, the remote shell service. The IDENT (RFC931 etc.) service can be used to detect such and other host address spoofing attacks.

Before accepting a client request, the wrappers can use the IDENT service to find out that the client did not send the request at all. When the client host provides IDENT service, a negative IDENT lookup result (the client matches ‘UNKNOWN@host’) is strong evidence of a host spoofing attack.

A positive IDENT lookup result (the client matches ‘KNOWN@host’) is less trustworthy. It is possible for an intruder to spoof both the client connection and the IDENT lookup, although doing so is much harder than spoofing just a client connection. It may also be that the client’s IDENT server is lying.

Note: IDENT lookups don’t work with UDP services.

EXAMPLES

The language is flexible enough that different types of access control policy can be expressed with a minimum of fuss. Although the language uses two access control tables, the most common policies can be implemented with one of the tables being trivial or even empty.

When reading the examples below it is important to realize that the allow table is scanned before the deny table, that the search terminates when a match is found, and that access is granted when no match is found at all.

The examples use host and domain names. They can be improved by including address and/or network/netmask information, to reduce the impact of temporary name server lookup failures.

MOSTLY CLOSED

In this case, access is denied by default. Only explicitly authorized hosts are permitted access.

The default policy (no access) is implemented with a trivial deny file:

/etc/hosts.deny:
ALL: ALL

This denies all service to all hosts, unless they are permitted access by entries in the allow file.

The explicitly authorized hosts are listed in the allow file. For example:

/etc/hosts.allow:
ALL: LOCAL @some_netgroup
ALL: .foobar.edu EXCEPT terminalserver.foobar.edu

The first rule permits access from hosts in the local domain (no '.' in the host name) and from members of the *some_netgroup* netgroup. The second rule permits access from all hosts in the *foobar.edu* domain (notice the leading dot), with the exception of *terminalserver.foobar.edu*.

MOSTLY OPEN

Here, access is granted by default; only explicitly specified hosts are refused service.

The default policy (access granted) makes the allow file redundant so that it can be omitted. The explicitly non-authorized hosts are listed in the deny file. For example:

/etc/hosts.deny:
ALL: some.host.name, .some.domain
ALL EXCEPT in.fingerd: other.host.name, .other.domain

The first rule denies some hosts and domains all services; the second rule still permits finger requests from other hosts and domains.

BOOBY TRAPS

The next example permits tftp requests from hosts in the local domain (notice the leading dot). Requests from any other hosts are denied. Instead of the requested file, a finger probe is sent to the offending host. The result is mailed to the superuser.

/etc/hosts.allow:
in.tftpd: LOCAL, .my.domain

/etc/hosts.deny:
in.tftpd: ALL: (/some/where/safe_finger -l @%h | \
/usr/ucb/mail -s %d-%h root) &

The safe_finger command is intended for use in back-fingering and should be installed in a suitable place. It limits possible damage from data sent by the remote finger server. It gives better protection than the standard finger command.

The expansion of the %h (client host) and %d (service name) sequences is described in the section on shell commands.

Warning: do not booby-trap your finger daemon, unless you are prepared for infinite finger loops.

On network firewall systems this trick can be carried even further. The typical network firewall only provides a limited set of services to the outer world. All other services can be "bugged" just like the above tftp example. The result is an excellent early-warning system.

DIAGNOSTICS

An error is reported when a syntax error is found in a host access control rule; when the length of an access control rule exceeds the capacity of an internal buffer; when an access control rule is not terminated by a newline character; when the result of %<letter> expansion would overflow an internal buffer; when a system call fails that shouldn't. All problems are reported via the syslog daemon.

IMPLEMENTATION NOTES

Some operating systems are distributed with TCP Wrappers as part of the base system. It is common for such systems to build wrapping functionality into networking utilities. Notably, some systems offer an *inetd*(8) which does not require the use of the *tcpd*(8). Check your system's documentation for details.

FILES

/etc/hosts.allow, (daemon,client) pairs that are granted access.
/etc/hosts.deny, (daemon,client) pairs that are denied access.

SEE ALSO

tcpd(8) tcp/ip daemon wrapper program.
tcpdchk(8), tcpdmatch(8), test programs.

BUGS

If a name server lookup times out, the host name will not be available to the access control software, even though the host is registered.

Domain name server lookups are case insensitive; NIS (formerly YP) netgroup lookups are case sensitive.

AUTHOR

Wietse Venema (wietse@wzv.win.tue.nl)
Department of Mathematics and Computing Science
Eindhoven University of Technology
Den Dolech 2, P.O. Box 513,
5600 MB Eindhoven, The Netherlands