Selinux; securing a legacy ftp server

by

Brian Tate

A paper submitted in partial fulfillment of the requirements for the GSEC v1.4c Option 1

January 13, 2005

Abstract

Selinux; securing a legacy ftp server

by Brian Tate

Some "secure" operating systems are proprietary in their application as a base for a single use host, for example the firewall offerings of the Cyberguard Corporation and Secure Computing Corporation. SELinux gives the information security community at large a new opportunity and powerful tool to build enterprise-class hosts with an operating system tuned to protect applications.

The SELinux protection mechanism can be applied to Apache, Oracle, or a myriad of other user-space applications. In this paper we will explore an application of SELinux security to a FTP host. We will begin with how SELinux's internals work to provide protection to the system. Then we will move on to a brief customization of the SELinux default policy, and we will culminate with a demonstration of SELinux averting some undesired behavior allowed by a FTP daemon.

Table of Contents

1. Chapter 1

1.1. Requirements for this FTP Server

This paper will look at the usage of SELinux (Security Enhanced Linux) to help a fake corporation, FakeCorp augment their system security. Specifically, we want a methodology resulting in a more secure bastion FTP server. FakeCorp uses their DMZ FTP server for placing files for client downloads. The files to be downloaded are not of a competition sensitive persuasion; so the FakeCorp team does not require encryption of the transaction. Access to the FTP server is controlled by a unique username, password and directory per client. Clients are only given the ability to download files from their own home directory and list the contents of their home directory, they cannot upload, rename, append or otherwise change files in their directory.

FakeCorp is concerned that an attacker could leverage a compromised FTP server to attack other, more valuable and critical systems. Access to these systems could then be leveraged to steal or deny service to important company data, network and computing resources. FakeCorp has already taken generally prudent steps by placing the FTP server in a DMZ zone outside of their core company network and following industry standard hardening and configuration guidelines for their servers.

The question becomes, can SELinux help improve the security of such a setup? More generally, and less specifically addressed in this paper, can SELinux be used for other types of single purpose servers, such as a database server or a web server? We will look at the role SELinux can play in stemming an example of a classic type of attack on FTP servers, namely the buffer overflow in its many incantations. The general idea of the FTP buffer overflow it to cause undesired (by the system owner) behavior to come out of legitimate commands perturbed. One example occurs when a FTP client trying a GET command with too long a file path specified (Niewiadomski, p.1).

1.2. SELinux; a Brief introduction

SELinux is originally an NSA project with research partners at various times including Secure Computing, Network Associates, MITRE and the University of Utah. The original research involved the Flask security architecture on the academic operating system known as Fluke, but recently spun off into a modified Flask architecture implemented on the Linux operating system (McCarty, p.16). SELinux is now an open source contribution to the Linux lineage and community contributions can be managed through http://sourceforge.net/projects/selinux/ and through the NSA SELinux site at http://www.nsa.gov/selinux/.

SELinux is an attempt to address the need for better security support in Linux through mandatory access control (MAC) security policies. The configuration of the security support structure is, if properly enabled, configurable by a system administrator, meaning there is latitude for site based rules at the kernel level of the operating system, providing another valuable layer of defense to critical systems in accordance with the principle of "Defense in Depth". We will use this provided latitude to make more secure a legacy FTP server functionality, protecting a FTP server better from the effects of possible buffer overflows in the FTP daemon.

First let us understand a little bit of how the SELinux mechanism works. The SELinux security model for Mandatory Access Control (MAC) is implemented in a fine-grained manner, i.e. each system resource has it's own security context; meaning that each file, directory, socket and so on has assigned to it a context, and that context is then interpreted by the underlying security server to create security identifiers (SIDs), which are used to enforce the policy.

During a system call, first the Linux Discretionary Access Control (DAC) mechanism is queried in the standard UNIX way, determining if the process originating the system call has appropriate membership as an owner, group or regular system user, if DAC is passed then the system passes the query to the MAC schema. This is one place where the implementation of the MAC scheme has changed in the newer Linux 2.6 kernel; in the 2.4 kernel; "when a security decision is required, the policy enforcement code passes a pair of SIDs (typically the SID of a subject and the SID of an object, but sometimes a pair of subject SIDs or a pair of object SIDs), and an object security calls to the security server" (Smalley, p.3).

In the 2.6 kernel's implementation "decision requests are sent to the access vector cache (AVC), which passes requests through to the security server for interpretation" (Morris, p.1). Once the request gets to the security server the implementation works the same, it consults the security policy database and determines a result, then returns that result to the SELinux hook with either a pass or a no pass.

One question that might come to mind is where exactly are these security attributes of files stored? In a 2.6 kernel implementation they are stored in the Extended Attributes (EA), which limits presently the use of SELinux to "ext3, ext2, XFS and ReiserFS; the last users an external patch" (Morris, p.3). These attributes are passed along with files by non-SELinux aware portions of the kernel, meaning that labeling persists and is not removed by other parts of the kernel, libraries or executables.

The important attributes associated with processes are called domains, "a domain directly determines the access a process has (Coker Getting, p.4). A process's domain defines how it can relate to other system resources. A type is essentially the same thing (and some authors use the two interchangeably) but applied to Linux system objects like sockets and directories. A domain or type can be thought of as the perimeter of a sandbox containing the resources needed for functioning of processes inside that domain.

A role defines a set of actions and resources available to a user in that role; one of the common roles for domain-defined processes is system_r. Commonly seen roles include user_r, the general user, system_r, a user who can assume higher roles, and sysadm_r, or system administrator role, the role typically afforded the most permissions, and in Fedora Core 2 protected by a "su" mechanism in the same way that assuming root identity is in "regular" Linux (where by regular I simply mean non-SELinux).

The combination of types (or domains) and roles and the user creates the aforementioned context of a file or process, and forms the meat of the SID used by the 2.4 kernel. This context is compared to the policy for that type of user and the domain that user is running in to make decisions about system access.

Note that all of this depends upon the SELinux "hooks into a broad selection of user-level administrator tools as well as the Linux kernel" (Immunix, p.5). This is a point raised by some detractors of the SELinux implementation, but this low-level integration into the operating system does come with its advantages. Integration at the kernel level with a well defined interface mitigates the risk of attack against the SELinux structure as there are relatively fewer attack vectors available for use.

Note that SELinux only provides processes with the resources it has been allowed by policy to use; the default for an undeclared object is denial. This is fundamentally different than the traditional UNIX root paradigm, where root access equals ownership, as now even the sysadm_r role can be restricted, and users can be restricted from using the root password even if they knew it. This addresses some of the serious flaws in the traditional UNIX security model, for example the setuid root process. When a setuid root process executes it has all of the permissions of the system's root user, making it an attractive target for an attacker wanting root access, and thereby access to all the files and traffic associated with that server. "The setuid program, when accepting root access, and therefore full reign of the OS, is most likely accepting far more privileges than it actually needs" (Rajnic, p.2). Thus we restrict the access that the setuid process has by the domain that it resides in, limiting the process's reign to that of the root user inside the sandbox domain.

2. Chapter 2

2.1. SELinux in Red Hat Fedora Core 2

The implementation in each flavor of the Linux operating system varies mainly by the location of the source documents and the default policy provided by the SELinux developers. SELinux is currently available in the Red Hat Fedora Core 2 and Fedora Core 3, Gentoo Linux, Debian GNU/Linux, and SUSE Linux (McCarty, p.16). In Fedora Core 2 the source files are located under /etc/security/selinux/src/policy. This source directory will vary based on your chosen Linux distribution. With the installation of the development libraries and the policy source one can edit the source files, compile the additions into a policy and load that policy, all through the use of some simple "make" commands and a text editor.

The files included in the Appendix are the Fedora Core 2 default ftpd.fc and ftpd.te, the file context and type enforcement policy files for FTP applications and related files. The file ftpd.fc contains statements defining the types of files, and defaults for directories associated with the various defined FTP domains, ftpd_exec_t and xferlog_t for example. The file ftpd.te contains the type enforcement rules for the ftpd domain and contains the rules governing what this domain can do on the system. We will take a look at these default files for some explanation of the system actions they allow the FTP daemon.

A quick look in our default ftpd.fc for our vsftpd finds that it will have a context of system_u:object_r:ftpd_exec_t, meaning the generic system_u user, its role is the generic object_r and its type (or domain if you prefer) is ftpd_exec_t, simply named as it is an ftpd executable. Other items of interest in this file include the /var/log/xferlog.* of context system_u:object_r:xferlog_t, meaning it is of type xferlog_t.

The type enforcement declarations of ftpd.te require a much more in depth analysis, but with the help of the exhaustive tables in the back of Bill McCarty's SELinux book we shall embark down that path. We will not however finish the track laid out as a complete explanation of the type enforcement for the FTP daemon is beyond the scope of the paper.

Let's begin with the declaration "type ftp_port_t, port_type". This associates the ftp_port_t with being a port_type, as does the next line for the ftp_data_port_t. This lets us know that we expect two different ports to be used, and of different type, which for FTP is known are the control and data ports. The next statement in the ftpd.te file is "daemon_domain(ftpd,`,auth_chkpwd')", this is a m4 macro provided by Fedora Core 2 developers. The macro sets up the commonly needed use of process id (PID) files. The auth_chkpwd attribute allows this daemon to check passwords, i.e. it allows the domain the ability to authenticate users with the unix system apparatus.

The next statement we shall evaluate is the “can_network(ftpd_t)”; this is a Fedora Core 2 m4 macro that loads into the policy the ability for ftpd_t to access network resources. Related to those network resources are the two statements “allow ftpd_t ftp_port_t:tcp_socket name_bind” and “can_tcp_connect(userdomain, ftpd_t)”. These two statements allow the ftpd domain to bind to a tcp socket and then connect the tcp socket from the ftpd domain to the user domain, which contains user_t. This is the permission that allows a tcp connection to be directed to a user. We have not however yet defined rules that let the ftpd domain access our user’s home directories; this will be added by us in a later section.

The line “allow ftpd_t home_root_t:dir {getattr search}” enables the ftpd domain to search through directories with type of home_root_dir, which in most cases is simply /home, this is included to allow the daemon process to traverse up to /home from the root directory “/”. In the next section we shall add statements to the ftpd.te that will allow the daemon to proceed down from “/home” into a user’s home directory and read files from there.

Another interesting segment for UNIX aficionados out there is the line “allow ftpd_t self:capability {chown fowner fsetid setgid setuid net_bind_service sys_chroot sys_nice sys_resource}” because of the breadth of that statement we will take it one small piece at a time. First in general it allows the ftpd type capabilities to do things to itself, so here SELinux is protecting the FTP daemon from itself as well as other applications from the FTP daemon. The chown permission allows the ftpd type to change ownership of files and directories that it can access. Related to ownership as well are the fowner and fsetid capabilities, or the capability to override file ownership permissions and the capability to override effective userid checks. These checks are routinely overwritten in standard UNIX by a user running as root, so it is not surprising to see them here separately protected.

We also allow a setuid call, again fairly standard for an FTP daemon. Then we see the important sys_chroot, or allowing a system call to the chroot facility, used by FTP daemons to lock users into a “chroot jail”, wherein they are in their home directory and cannot traverse out of it. This chroot call is important to the security of an FTP server, so its inclusion here is welcome. The remaining two capabilities set the process priority (sys_nice) and usage of system resources as defined in /usr/include/linux/capabilities.h (McCarty, p.207).

FTP daemons often write transfer data out to the /var directory under the name xferlog. In our ftpd.te the line “type xferlog_t, file_type,sysadm_file,logfile” associates the xferlog type as a file owned by a sysadm and used as a logfile. The next line “file_type_auto_trans(ftpd_t, var_log_t, xferlog_t, file) is another m4 macro allowing the ftpd domain to automatically transition a file of type var_log_t to a file of type xferlog_t. This sounds strange, but the default type of a file under /var is var_log_t, so when the daemon creates a new file in /var it needs to be transitioned to the xferlog_t type.

2.2. Securing a FTP server

Now that we have examined the MAC scheme in a bit more depth, we will begin with the work of better securing a FTP server with the scheme provided by SELinux. We will be working with a Red Hat Fedora Core 2 system on Intel kit. The first order of business is of course the installation of the system. In order to ensure that SELinux is installed you must type *linux selinux* (McCarty, 41) at the boot prompt during the installation procedure, then proceed choosing the packages as you usually would, making sure to include the FTP server package containing the vsftpd (Very Secure Ftp Daemon). We will also install SELinux development and policy source packages. They are contained by default on the 4-disc installation media available over the Bit-Torrent or from the Red Hat Fedora Download website.

Make sure to install the **policy-sources rpm**, as otherwise you will only have a functional system, which cannot have its policy modified. This is not one of the packages installed during one of the canned install solutions, so I recommend doing your install and then, as root, getting the policy-sources rpm off of the disk manually via a sequence like:

```
#mount /dev/cdrom /mnt/cdrom
#cd /mnt/cdrom/Fedora/RPMS
#rpm –ivvh /mnt/cdrom/Fedora/RPMS/policy-source{version, arch info}
```

Also needed are the rpms: policy, checkpolicy, libselinux-devel, and libselinux. These rpms can be obtained via a similar sequence to that used to obtain the policy-sources rpm.

Once installed on the system the default ftpd.te does not allow users any access to their own home directories, so we shall add policy statements allowing read and search access to home directories to ftpd.te and reload the SELinux policy. In particular, add the following three lines:

```
allow ftpd_t user_home_dir_t:dir { getattr search read };
allow ftpd_t user_home_dir_t:file { getattr read };
allow ftpd_t user_home_t:file { getattr read };
```

When we left our earlier discussion of the ftpd.te file our daemon could traverse only as far as the “/home” directory, however to retrieve files our

FakeCorp clients will need to first get into their home directory and then read files inside of it. Thus, once the daemon gets to /home in our coming example it will need to get into /home/sansuser, meaning the daemon will need search and getattr again for user home directories, which happens to have a type named user_home_dir_t. Moreover the daemon will need to be able to list the files in the user's home directory, so the daemon will need read permission on the directory structure, where a directory read means the same in this SELinux policy as it does in the standard UNIX DAC read bit on a directory.

Next, since the user will need to be able to download files we give the daemon (acting on the user's behalf) permission to read and getattr files of type user_home_dir_t. Finally, to download the file the daemon needs permission to read the files within the /home/sansuser directory, so we allow the daemon getattr and read access to files of type user_home_t.

Note that we have not allowed write permission for the ftpd type on any of the directory types or file types, meaning that the MAC scheme will not allow the daemon write access to files in these directories. A careful reading of the included ftpd.te will show the interested reader that the daemon has write access to very few areas. Only those areas likely to be used by a FTP daemon are accessible mainly the xferlogs located under /var.

These restricted permissions are important to security of an FTP daemon; we wouldn't want to allow write permission that might be subverted to a system compromise. This type of protection could screen our system from compromises in the realm of CAN-2003-0466 from the CVE list hosted by the MITRE Corporation (9). This candidate for a CVE reveals a buffer overflow possibility in the popular wu-ftpd (Washington University FTPd). We are not using this particular daemon in this demonstration; however it is prudent to understand what ways and means attackers could deploy against your system, so let us analyze this CVE candidate for our SELinux setup. The CVE lists that commands of sufficient length (which would be quite long as viewed) can trigger a buffer overflow, including the FTP commands "STOR, RETR, APPE, DELE, MKD, RMD, STOU, RNTO" (CAN-2003-0466).

Let us look at each of these commands to access if we have gain a layer of protection via our SELinux setup against an attacker logged in via FTP. The first command, STOR is the FTP protocol command to store a file on a remote host, which we do not allow the daemon to accomplish on behalf of a user, so an extra level of protection is afforded by our setup (Anon, p.1). We are similarly protected from a malicious STOU command as it is simply a store unique, but still a store command, and thus denied by our policy. Again by similar logic we are protected from the APPE (append) and DELE (delete) commands, as each requires a file write permission in the user's home directory.

Similar protection is provided against malicious RNTO (file rename to), MKD (make directory) and RMD (remove directory) commands since the policy doesn't allow write permission to the directory structure or to files or directories within the home directory. So the good news is that we have an extra level of protection against at least 7 of the 8 possible attack modes, but what about the seventh? The seventh possible FTP command is RETR, a retrieval of a remote file, issued by a user as a get command. This command may still be susceptible to attack, however this risk is greatly mitigated, as unless the SELinux policy is subverted the only domains the malicious user can enter are those granted entrance to the ftpd type.

2.3. Protection Demonstration

Now let use test out our kernel protection on an improperly configured vsftpd daemon, i.e. one that is not disallowing FTP PUT operations. Note that this unsafely configured daemon is for illustrative purposes only; it is certainly unwise to rely solely on the MAC scheme to protect us from malicious FTP daemon usage. Prior to this we must, as a system administrator running in the sysadm_r role go to the directory /etc/security/selinux/src/policy and issue the command *make reload* (Coker Writing, p. 4). The FTP session then goes like this, with the entered text in bold:

```
homehost>ftp 192.168.1.1
Connected to 192.168.1.1
220 (vsftpd 1.2.1)
User (192.168.1.1:(none)):sansuser
Password:
230 Login successful.
ftphost>put badfile
533 Could not create file
```

This is a simple example; let us look a little deeper into the audit message generated by this action, the audit is put into the kernel ring buffer; it can be output via a "dmesg" command. The urbane output in this case is:

```
audit(1105511951.491:0): avc:  denied  { write } for  pid=3029
exe=/usr/sbin/vsftpd name=sansuser dev=hda2 ino=4021922
scontext=root:system_r:ftpd_t tcontext=system_u:object_r:user_home_dir_t
tclass=dir
```

This message tells us that a write operation was denied for the executable /usr/sbin/vsftpd with the userid sansuser to a directory with the type of user_home_dir_t. The avc alerts us that this alert was generated by the Access Vector Cache, part of the SELinux structure. We can also see that the daemon is running as the root user, in the system_r role and is in the ftpd_t domain. Note that the audit message does not include the exact resource or command denied, just that information from within the context of the SELinux support structure, i.e. the user, PID, types and contexts.

Now let us consider one of the above commands possibly exploited for an attack, the MKD, or as entered into the FTP client, mkdir command. We will again attempt to make a directory in an incorrectly configured vsftpd FTP daemon and watch SELinux at work. The conversation follows:

```
homehost>ftp 192.168.1.1
Connected to 192.168.1.1
220 (vsftpd 1.2.1)
User (192.168.1.1:(none)):sansuser
Password:
230 Login successful.
ftphost>mkdir fakedir
550 Create directory operation failed.
```

In the audit messages we get: the message:

```
audit(1105518982.507:0): avc:  denied  { write } for  pid=15421
exe=/usr/sbin/vsftpd name=sansuser dev=hda2 ino=4021922
scontext=root:system_r:ftpd_t tcontext=system_u:object_r:user_home_dir_t
tclass=dir
```

This means that the executable /usr/sbin/vsftpd running in the system role and belonging to the ftpd_t domain was not allowed to write to a directory of type user_home_dir_t. This is our denied MKD command, meaning that unless the DAC or MAC scheme of SELinux has been subverted we are protected against such an attack.

2.4. Conclusion

We have seen that indeed SELinux can help by adding a layer of protection to our FTP host, remember however that this does not mean attackers are without their vectors, it could still be possible to mount an attack subverting or even directly attacking the SELinux support structure. Does SELinux mean that we no longer have to worry about daemon configuration and server hardening? NO, SELinux does add a safety net beneath all of the industry standard security practices, but it does not replace them, neither was it designed to.

I put forth that SELinux has reached the point of viability for usage in Business - Class applications, but with some caveats. SELinux requires more sophistication of its administrators presently than other operating systems, but for those intrepid enough SELinux offers some excellent security benefits at an extremely low price (free as in speech). And with many major vendors beginning to port products to Linux, and increased media scrutiny on subjects like worm, computer viruses and identity theft it seems likely that SELinux can play a part in future more secure deployments of business critical applications like databases and file servers.

SELinux has not yet reached the point of readiness for government or some fields of business that require more detail in audit messages. However, the community of developers has thus far proven themselves very competent. SELinux seems to be a technology whose evolution will continue for some time to come.

3. Bibliography

Coker, Faye. "Getting Started with SE Linux HOWTO: the new SE Linux" Home Page. 17 Mar. 2004. <http://www.lurking-grue.org/WritingSELinuxPolicyHOWTO.pdf>.

Coker, Faye. "Writing SE Linux policy HOWTO" Home Page. 17 Mar. 2004. <http://www.lurking-grue.org/WritingSELinuxPolicyHOWTO.pdf>.

McCarty, Bill. SELINUX: NSA's Open Source Security Enhanced Linux, 1st ed. Sebastopol, CA: O'Reilly Media, 2004.

Morris, James. "Filesystem Labeling in SELinux" LINUX Journal. 1 Oct 2004 <http://www.linuxjournal.com/article.php?=7426>.

Immunix, Inc. Immunix vs. SELinux: A comparison of Host Application Security Solutions. Portland, Or, 2004. <http://www.immunix.com/pdfs/immunix_selinux.pdf>.

Rajnic, Susan. "An Introduction to the NSA's Security-Enhanced Linux: SELinux <http://www.sans.org/rr/whitepapers/linux/232.php>.

Smalley, Stephen. Configuring the SELinux Policy. NAI Labs January 2003. <http://ftp.funet.fi/pub/mirrors/www.nsa.gov/slinux/doc/policy2.pdf>.

Wade, Karsten. Fedora Core 2 SELinux FAQ, Red Hat Inc, 2004. <http://fedora.redhat.com/docs/selinux-faq-fc2/index.html#id3248024>.

"CAN-2003-0466 (under review)." Common Vulnerabilities and Exposures Database. 2003.

<http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0466>.

Anonymous . "Raw FTP Command List" <http://www.nsftools.com/tips/RawFTP.htm>.

Niewiadomski, Janusz and Purczynski, Wojciech. "wu-ftpd fb_realpath() off by one bug" July 31,2003. <http://ipsec.pl/vulnerabilities/isec-0011-wu-ftpd.txt>.

4. APPENDIX

4.1. /etc/security/selinux/src/policy/file_contexts/program/ftpd.fc

```
# ftpd
/usr/sbin/in\.ftpd          --          system_u:object_r:ftpd_exec_t
/usr/sbin/proftpd           --          system_u:object_r:ftpd_exec_t
/usr/sbin/muddleftpd        --          system_u:object_r:ftpd_exec_t
/usr/sbin/ftpwho            --          system_u:object_r:ftpd_exec_t
/usr/kerberos/sbin/ftpd --          system_u:object_r:ftpd_exec_t
/usr/sbin/vsftpd            --                          system_u:object_r:ftpd_exec_t
/etc/proftpd\.conf          --          system_u:object_r:ftpd_etc_t
/var/run/proftpd/proftpd-inetd -- system_u:object_r:ftpd_var_run_t
/var/run/proftpd/proftpd.scoreboard -- system_u:object_r:ftpd_var_run_t
/var/log/muddleftpd\.log.* --     system_u:object_r:xferlog_t
/var/log/xferlog.*          --          system_u:object_r:xferlog_t
/var/log/xferreport.*     --          system_u:object_r:xferlog_t
/etc/cron\.monthly/proftpd --     system_u:object_r:ftpd_exec_t
```

4.2. /etc/security/selinux/src/policy/domains/program/ftpd.te

```
#DESC Ftpd - Ftp daemon
#
# Authors:  Stephen Smalley <sds@epoch.ncsc.mil> and Timothy Fraser
#           Russell Coker <russell@coker.com.au>
# X-Debian-Packages: proftpd-common bsd-ftpd ftpd vsftpd
#

#################################
#
# Rules for the ftpd_t domain
#
type ftp_port_t, port_type;
type ftp_data_port_t, port_type;
```

```
daemon_domain(ftpd, `, auth_chkpwd')
etc_domain(ftpd)
typealias ftpd_etc_t alias etc_ftpd_t;

can_network(ftpd_t)
can_ypbind(ftpd_t)
allow ftpd_t self:unix_dgram_socket { sendto create_socket_perms };
allow ftpd_t self:unix_stream_socket create_socket_perms;
allow ftpd_t self:process { getcap setcap setsched };
allow ftpd_t self:fifo_file rw_file_perms;

allow ftpd_t bin_t:dir search;
can_exec(ftpd_t, bin_t)
allow ftpd_t { sysctl_t sysctl_kernel_t }:dir search;
allow ftpd_t sysctl_kernel_t:file { getattr read };

allow ftpd_t urandom_device_t:chr_file { getattr read };

ifdef(`crond.te', `
system_crond_entry(ftpd_exec_t, ftpd_t)
can_exec(ftpd_t, { sbin_t shell_exec_t })
')

allow ftpd_t ftp_data_port_t:tcp_socket name_bind;

ifdef(`inetd.te', `
domain_auto_trans(inetd_t, ftpd_exec_t, ftpd_t)
', `
define(`ftpd_is_daemon')
')

ifdef(`ftpd_is_daemon', `
rw_dir_create_file(ftpd_t, var_lock_t)
allow ftpd_t ftp_port_t:tcp_socket name_bind;
can_tcp_connect(userdomain, ftpd_t)
', `
ifdef(`inetd.te', `
ifdef(`tcpd.te', `domain_auto_trans(tcpd_t, ftpd_exec_t, ftpd_t)')

# Use sockets inherited from inetd.
allow ftpd_t inetd_t:fd use;
allow ftpd_t inetd_t:tcp_socket rw_stream_socket_perms;

# Send SIGCHLD to inetd on death.
allow ftpd_t inetd_t:process sigchld;
') dnl end inetd.te
')dnl end (else) ftp_is_daemon

ifdef(`ftp_shm', `
allow ftpd_t tmpfs_t:file { read write };
allow ftpd_t { tmpfs_t initrc_t }:shm { read write unix_read unix_write
associate };
')

# Use capabilities.
allow ftpd_t self:capability { chown fowner fsetid setgid setuid
net_bind_service sys_chroot sys_nice sys_resource };

# Append to /var/log/wtmp.
allow ftpd_t wtmp_t:file { getattr append };

# allow access to /home
allow ftpd_t home_root_t:dir { getattr search };

# Create and modify /var/log/xferlog.
type xferlog_t, file_type, sysadmfile, logfile;
file_type_auto_trans(ftpd_t, var_log_t, xferlog_t, file)

# Execute /bin/ls (can comment this out for proftpd)
# also may need rules to allow tar etc...
can_exec(ftpd_t, ls_exec_t)

allow initrc_t ftpd_etc_t:file { getattr read };
allow ftpd_t { etc_t etc_runtime_t }:file { getattr read };
allow ftpd_t proc_t:file { getattr read };

dontaudit ftpd_t sysadm_home_dir_t:dir getattr;
dontaudit ftpd_t krb5_conf_t:file { write };
allow ftpd_t krb5_conf_t:file { getattr read };
ifdef(`automount.te', `
allow ftpd_t autofs_t:dir { search };
')
allow ftpd_t self:file { read };
tmp_domain(ftpd)
ifdef(`ftp_home_dir', `
ifdef(`nfs_home_dirs', `
allow ftpd_t nfs_t:dir r_dir_perms;
allow ftpd_t nfs_t:file r_file_perms;
')dnl end if nfs_home_dirs
')dnl end if ftp_home_dir
```