Argus Security Solutions
for Linux

PitBull LX Administration Guide

Release 1.1.1

February 15, 2002

Argus Systems Group, Inc.
1809 Woodfield Drive
Savoy, IL 61874 U.S.A

Table of Contents

Chapter 1: Introduction

PitBull LX is a security system for Linux, Solaris and AIX-based systems. It allows the system administrator to secure important data by isolating files, processes, and network communications from undesirable access by other processes. This isolation is called *compartmentalization*.

PitBull LX is simple to understand and easy to administer. It is lightweight, meaning that it does not significantly decrease system performance. It is powerful, however, in its ability to compartmentalize the system. PitBull LX is not an application built on top of the operating system. Instead, PitBull LX access checks are built into the kernel itself. It is not possible to bypass PitBull LX by bypassing an application.

PitBull LX does not replace the standard Linux access controls (*Discretionary Access Controls*, or *DAC*) already in place on a standard system. PitBull LX provides access controls in addition to DAC. To access PitBull LX-protected files, processes must pass both DAC access checks and PitBull LX access checks.

PitBull LX does not interrupt the current operation of your system. This is a fundamental design principle of the product. PitBull LX can be installed on the system, and the system will perform exactly as it did prior to installation. After PitBull LX is installed, the system may be compartmentalized as much or as little as needed.

1.1 Target Audience

This guide is for the administrator of a Linux system to be secured with PitBull LX. The guide assumes a basic knowledge of your operating system, including process relationships, file and directory relationships, differences between the various types of filesystem objects, basic shell operations such as concatenation, and basic commands and arguments.

On the system, there may be users who need to know only a subset of the information contained in this guide. Since the information that users need to know varies greatly from system to system, there is no attempt in this guide to separate "user information" from "administrator information." The administrator will need to determine what the system users need to know about PitBull LX and provide that information to them.

1.2 Conventions

Throughout this guide,

- *Italic print* in body text indicates a new term, usually one specific to PitBull LX.
- **Bold print** in body text indicates important information.
- `Monospace print` indicates literal text as output on the computer display.

- *`Monospace italic print`* indicates variable text as output on the computer display.
- **`Monospace bold print`** indicates literal text that you type into the computer.
- ***`Monospace bold italic print`*** indicates variable text that you type into the computer.

1.3 System Installation

To install PitBull LX, follow the instructions in the Installation Guide provided in the CD-ROM jewel case. Installation requires a license string, which you must obtain from Argus. After the product is installed, the license string may be viewed with the `getlicense(1)` command. If a new license string is required after installation, use `setlicense(1).`

1.4 Required Software

PitBull LX runs on Linux, Solaris or AIX operating systems. See the Installation Guide for details about specific versions of the operating systems that are supported.

1.5 Technical Support

For technical support, contact Argus at:

email:	support@argus-systems.com
phone:	(217) 355-3700
web:	http://www.argus-systems.com (Click SUPPORT)

Chapter 2: PitBull LX Basics

This chapter provides an overview of the PitBull LX security system. This chapter explains components and concepts of the system, but does not provide details or mechanics of their workings. The features are discussed in depth in subsequent chapters, with examples that demonstrate how to use them.

2.1 Standard UNIX Security vs. PitBull LX security

The standard UNIX security system is based around *Discretionary Access Control (DAC)*, meaning that the security settings on a file are based on the owner's discretion. DAC is commonly known as a file's "permission bits." PitBull LX, on the other hand, uses a form of *Domain Based Access Control (DBAC)*, which cannot be circumvented, even by the system administrator.

PitBull LX security works in conjunction with DAC; it is meant to complement, rather than replace, traditional UNIX security.

Like DAC, LX security data can be placed on any filesystem object. In addition, LX security data can be placed on processes, network data, and IPC objects.

2.2 Access Domains and Domain Sets

PitBull LX security is based around the concept of an *access domain*. Conceptually, an access domain is a single compartment that is completely separate from all other access domains.

There are two types of access domains: a *file access domain* and a *network access domain*. File access domains are used to control access to filesystem objects, while network access domains are used to control access to network data.

2.2.1 File Access Domains

A file access domain can be broken down into three separate bits: a read bit, a write bit, and an execute bit.

File access domains are represented as `domain_name(rwx)`, where `domain_name` is the symbolic name for the domain and the grouping `(rwx)` represents the type of access: `r` for read access, `w` for write access, and `x` for execute (search) access.

Example

`fdom1(--x)`	- execute `fdom1` domain
`fdom2(r-x)`	- read and execute `fdom2` domain
`fdom3(rwx)`	- read, write, and execute `fdom3` domain

2.2.2 Network Access Domains

A network access domain is a single entity, and cannot be separated into smaller components. The network access domain represents all types of access to a network object.

Network access domains are represented as `domain_name(net)`, where `domain_name` is the symbolic name for the domain and `(net)` signifies that it applies to network objects.

Example

`eth0(net)`	Access to `eth0` domain
`port80(net)`	Access to `port80` domain

2.2.3 Domain Sets

A *domain set* is a set of access domains grouped according to type of access. For example, all the domains set for read access on a file make up the *read domain set* for that file. There are four types of domain sets: *read*, *write*, *execute*, and *network*. Domain sets are represented by the set type, a colon, and a comma-separated list of domains. For example, `write:fdom1,fdom3` indicates a write domain set in which the `fdom1` and `fdom3` bits are set.

Example

The following set of access domains:

Access Domains

```
fdom1(r-x)
fdom2(rwx)
fdom3(r-x)
ndom1(net)
```

would constitute the following domain sets:

Domain sets

```
  read: fdom1, fdom2, fdom3
 write: fdom2
  exec: fdom1, fdom2, fdom3
   net: ndom1
```

Here is the same information in a cross-reference chart format, so you can see the relationships.

DOMAIN SETS

ACCESS DOMAINS

	READ	WRITE	EXECUTE	NET-WORK
`fdom1`	r		x	
`fdom2`	r	w	x	
`fdom3`	r		x	
`ndom1`				net

2.2.4 Domain Set Relationships and Operations

PitBull LX uses two relationships between domain sets (*dominance* and *intersection)* and two operations on domain sets (*masking* and *combining*) to determine the outcome of various situations.

2.2.4.1 Dominance

One domain set (set1) *dominates* another domain set (set2) if every element of the second domain set is contained in the first domain set (set2 is contained within set1). In other words, set1 dominates set2 if the logical intersection of set1 and set2 is exactly set2. If two sets are equal, they dominate each other.

Examples

Domain Set 1	Domain Set 2	1 dominates 2?

`fdom1`
`fdom4`
`fdom5`
	`fdom1`
	`fdom4`
		YES

Domain Set 1	Domain Set 2	1 dominates 2?

`fdom1`
`fdom4`
`fdom5`
	`fdom1`
	`fdom4`
	`fdom5`
		YES

Domain Set 1	Domain Set 2	1 dominates 2?

`fdom1`
`fdom4`
`fdom5`
		YES

Domain Set 1	Domain Set 2	1 dominates 2?
fdom1	fdom1	NO
fdom4	fdom3	
fdom5	fdom5	

2.2.4.2 Intersection

A domain set (set1) is said to intersect a second domain set (set2) if the two domain sets have at least one domain in common (set1 and set2 share at least one domain). In other words, set1 intersects set2 if the logical intersection of set1 and set2 is not empty.

Examples

Domain Set 1	Domain Set 2	Intersection?
ndom1	ndom1	YES
ndom4	ndom4	
ndom5		

Domain Set 1	Domain Set 2	Intersection
ndom1	ndom2	YES
ndom4	ndom3	
ndom5	ndom4	

Domain Set 1	Domain Set 2	Intersection
ndom1		NO
ndom4		
ndom5		

Domain Set 1	Domain Set 2	Intersection
ndom1	ndom2	NO
ndom4	ndom3	
ndom5		

2.2.4.3 Masking

The mask of two domain sets (set1 and set2) is the set of domains that the two sets have in common. That is, the mask of set1 and set2 is the logical intersection of set1 and set2.

Examples

Domain Set 1	Domain Set 2	Mask
ndom1	**ndom1**	**ndom1**
ndom4	**ndom4**	**ndom4**
ndom5		

Domain Set 1	Domain Set 2	Mask
ndom1	ndom2	**ndom4**
ndom4	ndom3	
ndom5	**ndom4**	

Domain Set 1	Domain Set 2	Mask
ndom1	ndom2	
ndom4	ndom3	
ndom5		

2.2.4.4 Combining

The combination of two domain sets (set1 and set2) is the set of all domains that are present in either set. That is, the combination of set1 and set2 is the logical union of set1 and set2.

Examples

Domain Set 1	Domain Set 2	Combination
ndom1	ndom1	ndom1
ndom4	ndom4	ndom4
ndom5		ndom5

Domain Set 1	Domain Set 2	Combination
ndom1	ndom2	ndom1
ndom4	ndom3	ndom2
ndom5	ndom4	ndom3
		ndom4
		ndom5

Domain Set 1	Domain Set 2	Combination
`ndom1`		`ndom1`
`ndom4`		`ndom4`
`ndom5`		`ndom5`

2.3 Security Flags

In addition to access domains, PitBull LX also employs *security flags* to control the behavior of subjects (acting processes) and objects (files, directories, IPC objects, sockets, target processes). Security flags can be broken down into four categories: *process security flags, file security flags, network security flags*, and *IPC security flags*.

2.3.1 Process Security Flags

Process security flags will restrict the behavior of processes. The process security flags are listed below, along with brief descriptions. The process security flags will be discussed in greater detail in the appropriate sections of this guide.

`ASG_AWARE`

This flag indicates that the process is protected by PitBull LX. This flag is automatically set if any other LX attributes are associated with the process. If this flag is set on a process, the process may be referred to as *LX Aware*.

`ASG_SEC_ACC_FS`

This flag indicates that a process may only access files that are LX Aware. If this flag is set on a process, the process may be referred to as being in *secure file access mode*.

`ASG_SEC_ACC_NET`

This flag indicates that a process may only interact with network data that is LX Aware. If this flag is set on a process, the process may be referred to as being in *secure network access mode*.

`ASG_RES_SYS`

This flag indicates that a process may not perform certain actions on the system that may modify PitBull LX security.

`ASG_RES_MNT`

This flag indicates that a process may not mount or unmount filesystems.

`ASG_RES_RBT`

This flag indicates that a process may not reboot the system.

`ASG_RES_DAC_OVR`

This flag indicates that a process may not override DAC restrictions.

`ASG_RES_SETID`

This flag indicates that a process may not change its user or group IDs.

`ASG_RES_BIND`

This flag indicates that a process may not bind to a privileged port.

2.3.2 File Security Flags

File security flags may modify the behavior of filesystem objects. The file security flags are listed below, along with brief descriptions. The file security flags will be discussed in greater detail in the appropriate sections of this guide.

`ASG_AWARE`

This flag indicates that the file is protected by PitBull LX. This flag is automatically set if any other LX attributes are associated with the file. If this flag is set on a file, the file may be referred to as *LX Aware*.

`ASG_MASK_EXEC`

This flag indicates that a process may have its abilities restricted upon execution of the file.

`ASG_COMB_EXEC`

This flag indicates that a process may have its abilities increased upon execution of the file.

`ASG_INHERIT_DOM`

This flag indicates that a newly created file may inherit attributes of its parent directory. This flag only applies to directories.

2.3.3 Network Security Flags

Network security flags may modify the behavior of network data. The network security flags are listed below, along with brief descriptions. These flags will be discussed in more detail in the appropriate sections of this guide.

`ASG_AWARE`

This flag indicates that the network data is protected by PitBull LX. If this flag is set on network data, the network data may be referred to as *LX Aware*.

`ASG_SEC_ACC_NET`

This flag indicates that the data was written by a process that is LX Aware. If this flag is set on network data, the network data may be referred to as being in *secure network access mode*.

2.3.4 IPC Security Flags

IPC security flags may modify the behavior of System V IPC objects (message queues, shared memory, and semaphores). The IPC security flags are listed below with brief descriptions. The IPC security flags will be discussed in more detail in the appropriate sections of this guide.

`ASG_AWARE`

This flag indicates that the IPC object is protected by PitBull LX. If this flag is set on an IPC object, the object may be referred to as *LX Aware*.

2.4 Object Attributes

2.4.1 Filesystem Object Attributes

Each filesystem object has a read access domain set, a write access domain set, an execute access domain set, and file security flags. These domain sets and file security flags are used for PitBull LX access checks and file security flags may modify certain behaviors of a file. The file's security flags may consist only of the file security flags mentioned earlier. Filesystem objects also have a read execution domain set, a write execution domain set, an execute execution domain set, a net execution domain set, and execution security flags. These execution attributes may affect a process executing certain binaries as discussed later. The execution security flags may consist only of the process security flags mentioned earlier.

2.4.2 Process Attributes

Each process has an effective read access domain set, effective write access domain set, effective execute access domain set, and an effective network access domain set. These domain sets are used for PitBull LX access checks. A process also has a maximum read access domain set, maximum write access domain set, maximum execute access domain set, and a maximum network access domain set. These are "upper bounds" for the effective domain sets of the process. Processes also have a set of security flags. The process' security flags may consist only of the process security flags mentioned earlier.

2.4.3 Network Data Attributes

Network data has a network access domain set associated with it. This domain set is used for PitBull LX network access checks. Network data also has a set of security flags. The security flags may consist only of the network security flags mentioned earlier.

2.4.4 IPC Object Attributes

Each System V IPC (shared memory, message queues, and semaphores) object has a read access domain set and write access domain set. These domain sets are used for PitBull LX access checks. A System V IPC object also has a set of security flags. The security flags can consist only of the IPC security flags mentioned earlier.

FIFOs and UNIX Domain Sockets exist as filesystem objects and therefore operate with the attributes of a filesystem object.

2.5 Network Rules

PitBull LX allows an administrator to specify a set of *network rules* which can be used to prevent unauthorized processes from reading and writing restricted data that passes across a network. An LX network rule is similar to a firewall rule, except that packets are labeled with LX network domains when a successful match is made. Processes attempting to read or write network data must have a domain set that intersects with the network data before the read/write is allowed. Network rules are discussed in detail in a later chapter.

Chapter 3: Access Checks

This chapter describes how PitBull LX attributes are used to affect access operations on the system. These access checks and restrictions apply to all processes, regardless of superuser ability.

3.1 Filesystem Access Checks

Processes may be subject to PitBull LX access checks when accessing filesystem objects. This includes access for read, write, and execute, as well as modifying attributes of the file. The rules for these access checks are presented below and assume that the process already has the applicable DAC access.

3.1.1 Read, Write, and Execute Access

The basic rule is that LX performs access checks only under the following two conditions. If neither of these conditions is true, there is no LX access check:

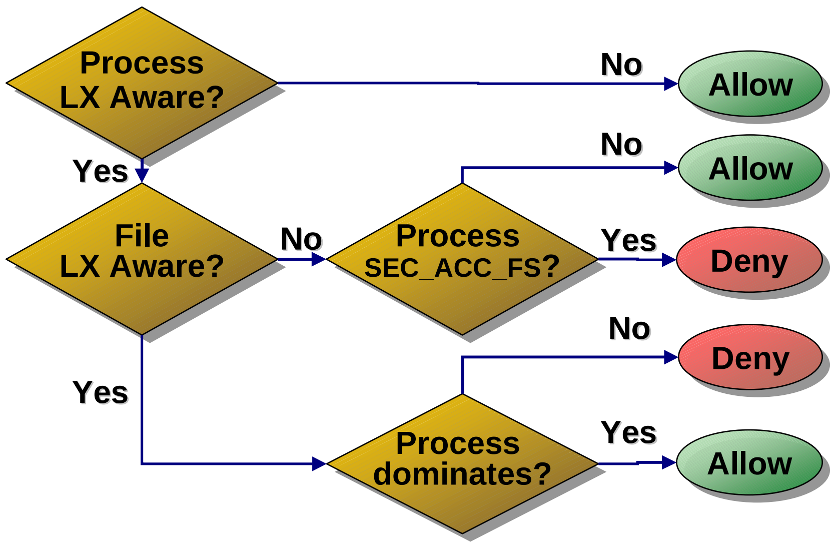
1) Both subject and object are LX Aware.

2) The subject is in secure file access mode.

If one or both of these conditions are true, LX access checks are performed as follows:

- A process that is LX Aware and in secure file access mode is denied access to a file that is not LX Aware.
- A process that is LX Aware is subject to a PitBull LX attribute check when accessing a file that is LX Aware. In order for an access attempt to succeed, the process' domain set must dominate the file's domain set for that access type (read, write, or execute).

The following chart is intended as a quick reference for the information provided above.



Example

These examples represent various processes attempting to access various files. The file attributes shown are the read, write, and execute access domain sets, as well as the file security flags. The process attributes shown are the effective read, write, and execute domain sets, as well as the process security flags. These examples assume that the process has met the DAC requirements for the specified type of access.

Process Attributes		File Attributes		Access	
read:	`fdom1,fdom2`	read:	`fdom1`	read:	`YES`
write:	`fdom1`	write:	`fdom1,fdom2`	write:	`NO`
exec:	`fdom1`	exec:	`fdom1`	exec:	`YES`
flags:	`ASG_AWARE`	flags:	`ASG_AWARE`		

Process Attributes		File Attributes		Access	
read:	`fdom1`	read:		read:	`YES`
write:	`fdom1`	write:		write:	`YES`
exec:	`fdom1`	exec:		exec:	`YES`
flags:	`ASG_AWARE`	flags:			

Process Attributes	File Attributes	Access
read:	read: `fdom1`	read: YES
write:	write: `fdom1,fdom2`	write: YES
exec:	exec: `fdom1`	exec: YES
flags:	flags: `ASG_AWARE`	

Process Attributes	File Attributes	Access
read: `fdom1`	read:	read: NO
write: `fdom1`	write:	write: NO
exec: `fdom1`	exec:	exec: NO
flags: `ASG_AWARE, ASG_SEC_ACC_FS`	flags:	

3.1.2 Attribute Modification Access

Modifying attributes (times, owner, group, mode, access domains, security flags) of a filesystem object requires write access as described above. Modifying LX attributes also requires ownership of the file.

3.1.3 Mount Access

Covering a directory by mounting a filesystem on it requires write access as described above.

3.2 Network Access Checks

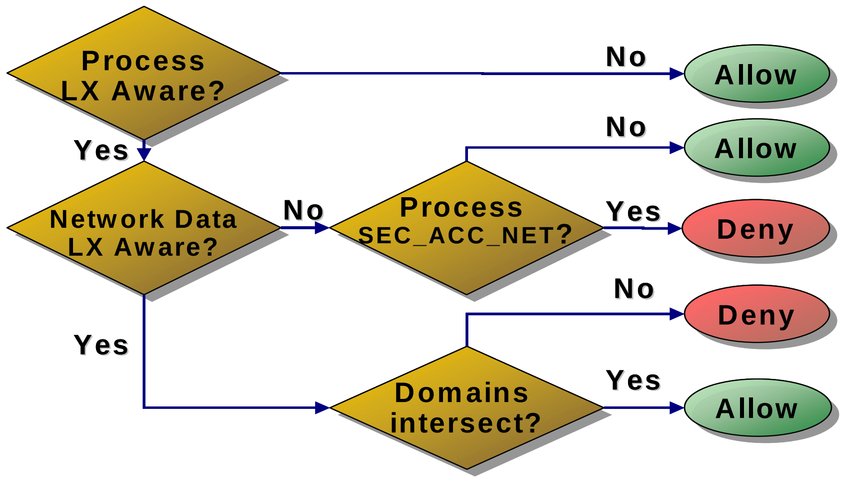
Processes may also be subject to PitBull LX access checks when accessing network data. This includes reading and writing data, as well as making or accepting a network connection. The rules for these access checks are presented below.

3.2.1 Read Access

When network data arrives in the system, it is matched against the set of network rules loaded into the kernel. When a matching rule is found, the rule’s LX domain is applied to the data, and the data becomes LX Aware. If a rule is not found, the data will not be LX Aware. When an LX Aware process attempts to read the data, the checks are similar to those done for filesystems. It is important to note that, in the case of filesystems, dominance is required. In the case of network read access, however, intersection is needed. The process and the network data must share at least one (1) domain. The differences between network data read access and filesystem access are as follows:

- Network read access checks deal with accessing data from the network, as opposed to filesystems.
- The process' effective network domains must intersect the data's network domain set (dominance is not required, only intersection).

The following chart is intended as a quick reference for the information provided above.

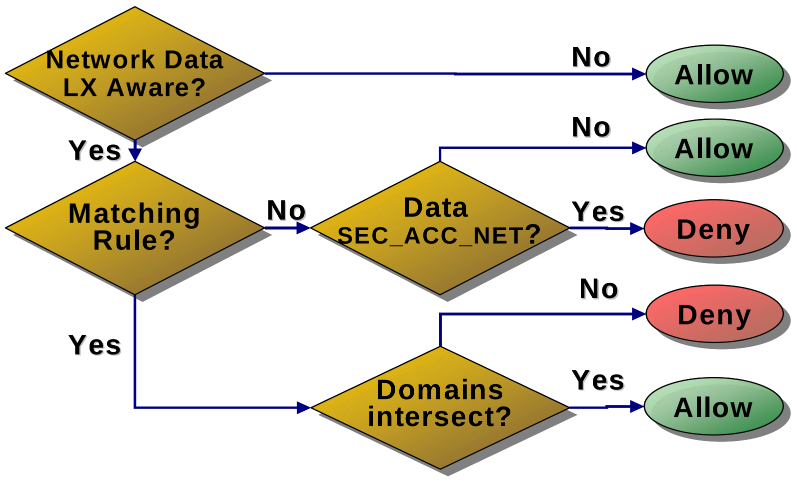


3.2.2 Write Access

When a process attempts to write data to the network, the data receives the process' effective network domain set and network flags. Before network data leaves the system, it is checked against the network rule set. The following checks are performed:

- Network data that is not LX Aware will not be subject to a PitBull LX attribute check when leaving the machine regardless of the presence of a matching rule.
- Network data that is LX Aware will be subject to a PitBull LX attribute check when leaving the machine if there is a matching rule. The data's network domain set must intersect with the rule's network domain set to be put on the network.
- Network data that is LX Aware and is in secure network access mode (which is inherited from the sending process) will be subject to a PitBull LX attribute check when leaving the machine if there is a matching rule. The data's network domain set must intersect with the rule's network domain set to be put on the network.
- Network data that is LX Aware and in secure network access mode will be denied access to the network if there is no matching rule.
- If the data is LX Aware and is not in secure network access mode and there is no matching rule, then access will be allowed.

The following chart is intended as a quick reference for the information provided above.



3.2.3 Connection Establishment

The establishment of a network connection by a connection-oriented service (such as TCP) is considered by PitBull LX to be the same as data transmission. Thus, it is possible that the set of network rules will prevent a connection from being established.

3.2.4 Network Access Examples

Reading Process Attributes	Incoming Data Attributes	Read Access
net: `ndom1,ndom2`	net: `ndom2,ndom4`	YES
flags: `ASG_AWARE`	flags: `ASG_AWARE`	

In the above example, the reading process and the incoming data both have the `ndom2` network domain, so the read is allowed.

Reading Process Attributes	Incoming Data Attributes	Read Access
net: `ndom1,ndom2`	net:	NO
flags: `ASG_AWARE` `ASG_SEC_ACC_NET`	flags:	

Here, although the incoming data does not have any network domains, the process is not allowed to read data that is not LX Aware, because the process has the `ASG_SEC_ACC_NET` flag. The read will fail.

Writing Process Attributes	Matched Rule Attributes	Write Access
net: `ndom2`	`(no rule found)`	YES
flags: `ASG_AWARE`		

Above, no rule could be found to match the outgoing data. Because the writing process does not have the `ASG_SEC_ACC_NET` flag, the data is allowed to continue out over the network.

Writing Process Attributes	Matched Rule Attributes	Write Access
net: `ndom1`	net: `ndom2,ndom4`	NO
flags: `ASG_AWARE`	flags: `ASG_AWARE`	

The writing process and the rule are both `ASG_AWARE`, so a check must be performed. Because the two have no network domains in common, the write is not allowed.

Writing Process Attributes	Matched Rule Attributes	Write Access
net:	net: `ndom1,ndom2`	YES
flags:	flags: `ASG_AWARE`	

Since the process is not LX Aware, no access checks are performed.

Writing Process Attributes	Matched Rule Attributes	Write Access
net: `ndom1`	net:	NO
flags: `ASG_AWARE`	flags: `ASG_AWARE`	

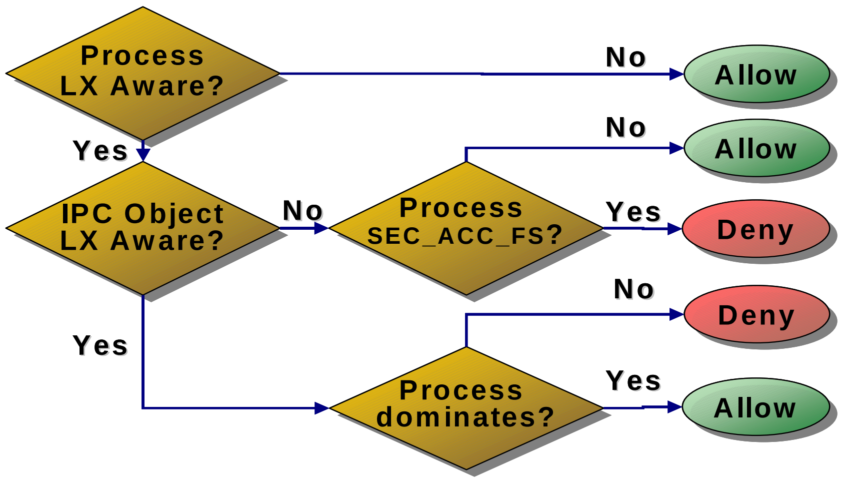
Though the writing process dominates the rule, because they do not intersect, there is no write access allowed to the network. The domains must intersect for write access to be allowed.

3.3 IPC Access Checks

Processes may also be subject to PitBull LX access checks when accessing IPC objects (semaphores, shared memory and message queues). This includes access for reading and writing, as well as modifying attributes of the object.

3.3.1 System V IPC Read and Write Access

The rules for reading and writing to IPC objects are identical to the rules for file-system access checks. The following chart is intended as a quick reference for the information provided above.



Examples

The object attributes shown are the read and write access domain sets, as well as the IPC security flags. The process attributes shown are the effective read and write domain sets, as well as the process security flags. These examples assume that the process has met the DAC requirements for the specified type of access.

Process Attributes	Object Attributes	Access
read: `fdom1`	read: `fdom1`	read: YES
write: `fdom1`	write: `fdom1,fdom2`	write: NO
flags: `ASG_AWARE`	flags: `ASG_AWARE`	

Above, the process read domain set dominates that of the object (they are equal), so read access is allowed. However, the object write domain set includes `fdom2`, which is not in the process write domain set, so write access is disallowed.

Process Attributes	Object Attributes	Access
read: `fdom1`	read:	read: YES
write: `fdom1`	write:	write: YES
flags: `ASG_AWARE`	flags:	

Since the IPC object is not LX Aware and the process is not in secure file access mode, no access checks are performed.

Process Attributes	Object Attributes	Access
read:	read: `fdom1`	read: YES
write:	write: `fdom1,fdom2`	write: YES
flags:	flags: `ASG_AWARE`	

Since the process is not LX Aware, no access checks are performed.

Process Attributes	Object Attributes	Access
read: `fdom1`	read:	read: NO
write: `fdom1`	write:	write: NO
flags: `ASG_AWARE, ASG_SEC_ACC_FS`	flags:	

The process is in secure file access mode, so the object would have to be LX Aware for any access to be granted.

3.3.2 FIFOs and UNIX Domain Socket Access

FIFOs and UNIX Domain Sockets exist as filesystem objects. Therefore, they obey the standard filesystem object access checks. UNIX Domain Sockets also use the network read and write checks once they are opened.

3.4 Restricted Actions

PitBull LX also provides the ability to restrict processes from performing certain actions on the system, through the use of process security flags. The actions that can be restricted are listed below, grouped by the process security flags.

`ASG_SEC_ACC_FS`

This flag removes a process' ability to access files that are not LX Aware.

`ASG_SEC_ACC_NET`

This flag removes a process' ability to access network data that is not LX Aware.

`ASG_RES_SYS`

This flag removes a process' ability to perform the following actions:

- Change the process' root directory using `chroot(2)`.
- Terminal hijack using the `TIOCSTI` vector of `ioctl(2)`.
- Create block and character devices using `mknod(2)`.
- Create a loadable module entry using `create_module(2)`.
- Set kernel parameters using `sysctl(2)`
- Initialize a loadable module entry using `init_module(2)`.
- Delete a loadable module entry using `delete_module(2)`.
- Change input/output port permissions using `ioperm(2)`.
- Change input/output port privilege level using `iopl(2)`.
- Set the system time/date using `stime(2)`, `adjtimex(2)`, and `set-timeofday(2)`.

`ASG_RES_MNT`

This flag removes a process' ability to do the following:

- Mount filesystems using `mount(2)`.
- Unmount filesystems using `umount(2)`.
- Export and unexport filesystems using `nfsservctl(2).`
- Add and remove clients using `nfsservctl(2)`.
- Manage swap space using `swapon(2)` and `swapoff(2)`.

`ASG_RES_RBT`.

This flag removes a process' ability to reboot the system using `reboot(2)`.

`ASG_RES_DAC_OVR`

This flag removes a superuser process' ability to override DAC restrictions. This includes overriding DAC when accessing files, changing file attributes, and accessing System V IPC objects.

`ASG_RES_SETID`

This flag removes a process' ability to change user IDs or group IDs using `setregid(2)`, `setgid(2)`, `setreuid(2)`, `setuid(2)`, `setre-suid(2)`, `setresgid(2)`, `setfsuid(2)`, `setfsgid(2)`, and `setgroups(2)`.

`ASG_RES_BIND`.

This flag removes a process' ability to bind to a privileged TCP or UDP port using `bind(2)`.

Chapter 4: Working with File Attributes

This section contains examples of how to manipulate access domains, security flags, execution domains, and execution flags on filesystem objects. The examples are designed to be performed on your system as the text is read. It may also be instructional to depart from these examples and experiment with the commands.

All of the examples assume that the shell process is not LX Aware. Some of the examples may require a shell process running as root. If so, the example will mention that fact.

4.1 Setting and Viewing Access Domains

To place access domains on files, use the `setfdom(1)`command. You can view a file's access domains using the `getfdom(1)` and `secls(1)` commands. Refer to the manual pages for complete descriptions of the options for these commands.

To begin, create a test file containing the ASCII text "test file."

```
% echo “test file” > test
```

View the file's access domain sets:

```
% getfdom test
test
 Access Read:
 Access Write:
 Access Exec:
```

The access domains for the file are empty.

Set the new file's access domains.

```
% setfdom –S –r fdom1,fdom2 –w fdom2 test
```

With the `–S` (Set) option, `setfdom(1)` assigns new domains and removes any existing ones. The `–r` option specifies which read domains to set, and the `–w` option specifies the write domains to set.

View the results.

```
% getfdom test
test
 Access Read:  fdom1,fdom2
 Access Write: fdom2
 Access Exec:
```

`getfdom(1)` reports the access domain sets of the target file. The read bits of `fdom1` and `fdom2` are set, as well as the write bit of `fdom2`.

Add new file domains to existing ones with the `–A` option of `setfdom(1)`, and view the results.

```
% setfdom –A –r fdom3 –w fdom1 –x fdom3 test

% getfdom test
test
 Access Read:  fdom1,fdom2,fdom3
 Access Write: fdom1,fdom2
 Access Exec:  fdom3
```

The read bit of `fdom3`, the write bit of `fdom1`, and the execute bit of `fdom3` are now set.

Delete selected domains from the file with `setfdom(1)`’s `–D` option.

```
% setfdom –D –r fdom1 –w fdom1 test

% getfdom test
test
 Access Read:  fdom2,fdom3
 Access Write: fdom2
 Access Exec:  fdom3
```

The read and write bits of `fdom1` are no longer set on the test file.

Display only the write domain set of the file.

```
% getfdom –w test
test
 Access Write: fdom2
```

With the `–w` option, a list of all domains that have the write bit set appears. Similarly, the `–r` and `–x` options return a list of all domains with the read and execute bits set, respectively.

4.2 Setting and Viewing Security Flags

Use the `setfsf(1)` command to set security flags on files.

```
% setfsf –S ASG_AWARE test
```

Use `getfsf(1)` to view security flags on files. `getfsf(1)` shows only the security flags, and no other attributes.

```
% getfsf test
test
   ASG_AWARE
```

You can also use the `secls(1)` command to view security flags on files. The `secls(1)` command shows all file attributes, including DAC attributes. Using the `-f` option shows only the flags for the file.

```
% secls -f test
-rw-r--r-- 1 root root 10 Oct 02 08:56 test
 File Security Flags:
   ASG_AWARE
 File Execution Flags:
   None
```

4.3 Setting and Viewing Execution Domains

To place execution domains on files, use the `setfdomex(1)`command. You can view a file’s execution domains using the `getfdomex(1)` and `secls(1)` commands. Refer to the manual pages for complete descriptions of the options for these commands.

View the file's execution domain sets:

```
% getfdomex test
test
 Execution Read:
 Execution Write:
 Execution Exec:
 Execution Net:
```

Use the `setfdomex(1)` and `getfdomex(1)`commands to modify and retrieve the file's execution domain set. Only a superuser process that is not LX Aware may use the `setfdomex(1)` command successfully.

```
# setfdomex –S –r fdom0 –w fdom3 –n ndom1,ndom4 test

# getfdomex test
test
 Execution Read:  fdom0
 Execution Write: fdom3
 Execution Exec:
 Execution Net:   ndom1,ndom4
```

`setfdomex(1)` and `getfdomex(1)` have exactly the same arguments as `setfdom(1)` and `getfdom(1)`, except the execution domain commands allow you to set and view network domain sets in addition to file domain sets.

4.4 Setting and Viewing Execution Flags

The `setfsfex(1)` and `getfsfex(1)` commands set and retrieve the execution flags from the files. They behave similarly to `setfsf(1)` and `getfsf(1)`.

Use `setfsfex(1)` and `getfsfex(1)` to set and view execution flags on files. The `setfsfex(1)` command requires a shell process running as superuser that is not LX Aware.

```
# setfsfex -S ASG_SEC_ACC_NET test

# getfsfex test
test
 ASG_AWARE,ASG_SEC_ACC_NET
```

The `ASG_SEC_ACC_NET` flag restricts a process to accessing only network data that is LX Aware. The system automatically adds `ASG_AWARE` to the execution flag set.

As an alternative to `getfdom(1)` or `getfdomex(1)`, use `secls(1)` to view file security attributes.

```
% secls test
-rw-r--r-- 1 root root 10 Oct 02 08:56 test
 Access File Domains:
    (rw-)fdom2
    (r-x)fdom3
  Execution File Domains:
    (r--)fdom0
    (-w-)fdom3
  Execution Network Domains:
    (net)ndom1
    (net)ndom4
  File Security Flags:
    ASG_AWARE
  File Execution Flags:
    ASG_AWARE,ASG_SEC_ACC_NET
```

`secls(1)` displays a complete listing of all PitBull LX attributes on the file, as well as the output you would see using the `ls –l` command. You can see that the system automatically added the `ASG_AWARE` flag to the access and execution flags when you set access and execution domains.

4.5 File Attribute Inheritance

You can set domains on files directly with the `setfdom(1)` and `setfdomex(1)` commands. The system also assigns attributes to a file based on the attributes of the process that creates the file. Any file created by an LX Aware process inherits attributes from the process. The file inherits the process' **effective** domain sets (not the maximum set) as its access domain sets. However, the only security flag inherited is the `ASG_AWARE` security flag. The exception being the `ASG_INHERIT_DOM` flag if the file is a directory and the parent directory has the flag.

Examples

Process	Created File
Access Domains	Access Domains
`(r-x)fdom1`	`(r-x)fdom1`
`(rw-)fdom2`	`(rw-)fdom2`
`(net)ndom0`	
	Execution Domains
Flags	Security Flags
`ASG_AWARE`	`ASG_AWARE`
`ASG_SEC_ACC_FS`	
	Execution Flags

Process	Created File
Access Domains	Access Domains
Flags	Execution Domains
	Security Flags
	Execution Flags

If an LX Aware process opens and writes or moves an existing file, the file's attributes are not changed. However, if an LX Aware process copies a file, the copy inherits the attributes of the copying process, not those of the original file.

If a directory has the `ASG_INHERIT_DOM` flag set, any files created in the directory will inherit attributes of the directory. This flag can be placed on any file, but it only has an effect if placed on a directory.

A newly created file will inherit security attributes from both its parent directory, if the directory has the `ASG_INHERIT_DOM` flag set, and from the creating process, if the process is LX Aware. If this is the case, the new file's attributes are assigned as follows:

- The new file's security flags will be set to `ASG_AWARE.`
- If the new file is a directory, it will also inherit the `ASG_INHERIT_DOM` flag.
- The new file's access domain sets will be set to the mask of the process' effective domain sets and directory's access domain sets.
- No execution domains or execution flags will be set.

A newly created file will also inherit security attributes from its parent directory, if it has the `ASG_INHERIT_DOM` flag set, even if the process is not LX Aware. If this is the case, the new file's attributes are assigned as follows:

- The new file's security flags will be set to `ASG_AWARE.`
- If the new file is a directory, it will also inherit the `ASG_INHERIT_DOM` flag.
- The new file's access domain sets will be set to the parent directory's access domain sets.
- No execution domains or execution flags will be set.

You may wish to experiment on your own with various combinations of settings to become familiar with these interactions.

Examples

Process	Parent Directory	Created File
Access Domains	Access Domains	Access Domains
`(r-x)fdom1`	`(-w-)fdom2`	`(-w-)fdom2`
`(rw-)fdom2`	`(r--)fdom4`	
`(net)ndom0`		
Flags	Flags	Flags
`ASG_AWARE`	`ASG_AWARE`	`ASG_AWARE`
`ASG_SEC_ACC_FS`	`ASG_INHERIT_DOM`	

Process	Parent Directory	Created File
Access Domains	Access Domains	Access Domains
	`(-w-)fdom2`	`(-w-)fdom2`
	`(r--)fdom4`	`(r--)fdom4`
Flags	Flags	Flags
	`ASG_AWARE`	`ASG_AWARE`
	`ASG_INHERIT_DOM`	

Additional Practice

This section provides additional practice and examples with attribute inheritance, access, and domain dominance. You must be the superuser to perform many of these operations; you may wish to become the superuser before starting this series. It is also assumed that your shell is not LX Aware.

Create a new test file, copy the program `more(1)` to the current directory and use it to display the contents of the test file:

```
    # echo “test file” > test.txt

    # cp /bin/more .

    # ./more test.txt
    test file
```

Set a read domain set on the new file, and see if `./more` still can read it.

```
    # setfdom –S –r fdom2 test.txt

    # ./more test.txt
    test file
```

Even though `test.txt` has an access read domain set, the `./more` process is not LX Aware, so it can read the file.

Set the ASG_MASK_EXEC flag so that ./more will be LX Aware when executed. This is covered in the next chapter.

```
# setfsf –S ASG_MASK_EXEC ./more

# secls ./more
-rwxr-xr-x 1 root root 23600 Oct 02 11:35 more
 Access File Domains:
   None
 Execution File Domains:
   None
 Execution Network Domains:
   None
 File Security Flags:
   ASG_AWARE,ASG_MASK_EXEC
 File Execution Flags:
   ASG_AWARE
```

The ASG_AWARE flag was set automatically in both the security and execution flags, since a file must have ASG_AWARE before it can have any other security flags. The ./more process will now be LX Aware on execution.

Try to see the contents of the test file again:

```
# ./more test.txt
test.txt: Permission denied
```

This time, ./more was unable to open the test file. Both the ./more process and the test file are LX Aware, and the test file requires a read access bit (fdom2) that ./more does not have.

There are several ways to allow ./more to read the contents of test.txt. You could remove the access read domain from test.txt's access domain set, or you could give ./more the domain it needs to read test.txt. For this example, set the ./more binary so that the process receives the fdom2 read bit when the file is executed. This way, ./more's read domain set will dominate test.txt's read domain set:

```
# setfdomex –S –r fdom2 ./more

# ./more test.txt
test file
```

The ASG_MASK_EXEC flag is already set on the ./more binary, and your shell process is not LX Aware, so the ./more process inherits all security settings from the ./more binary.

Dominance checks apply whenever an LX Aware process attempts to access an LX Aware file. For example, if a vi(1) editor process is LX Aware, its read and write domain sets must dominate both the read and write domain sets of the file to be edited (assuming the file has the ASG_AWARE flag). inetd, if run LX Aware, must have a read domain set that dominates the access read domain set of its configuration file, and inetd's execute domain set must dominate the access execute domain set of any LX Aware server it wishes to execute (again, assuming the configuration file and the target servers have the `ASG_AWARE` flag).

Chapter 5: Working with Process Attributes

Processes, like files, may have zero or more domains, each of which can have any combination of read, write, or execute permissions. You can set domains directly on files with the commands discussed previously, but you cannot set domains directly on processes.

A process may be programmed to change its own domains. For security, each process has an upper limit on what domains it can give itself. Therefore, processes have two domain sets. The effective domain set is the currently active set, used in access determination. The maximum domain set is the upper bound of all possible domains the process can have. A process can add or remove domains from its effective set, and it can remove domains from its maximum set, but it cannot add domains to its maximum domain set. A process cannot change the effective or maximum domain set of another process.

There are also no commands or system calls to remove security flags from a process. Once a process has a flag, it cannot be removed. However, a process can be programmed to add security flags to itself (including the `ASG_AWARE` flag, which enables a process to go from being LX Unaware to LX Aware). One result of this is that when a process is LX Aware (the `ASG_AWARE` flag is set), that process and all its children will always be LX Aware.

Documentation for the programming aspects of PitBull LX can be found in the PitBull LX man pages, sections 2 and 3.

5.1 Viewing Access Domains

Use the `getpdom(1)` and `secps(1)` commands to view a process' access domains.

See that you are running a shell process that is not LX Aware, and that no PitBull LX attributes are set.

```
% getpdom $$
10249
 Read_eff:
 Write_eff:
 Exec_eff:
 Net_eff:
 Read_max:
 Write_max:
 Exec_max:
 Net_max:

% secps $$
10249 root root
 Effective File Domains
   None
 Effective Net Domains
   None
```

```
Maximum File Domains
  None
Maximum Net Domains
  None
Security Flags
  None
```

5.2 Viewing Security Flags

There are no commands to alter the security flags on a process. The system assigns security flags to processes based strictly on rules of inheritance.

Recall that with domains, a process may be programmed to reduce or add domains that are included in the process' maximum domain set. A process may also be programmed to alter its flag set, but the process can only **add** flags. It cannot remove any of its security flags. As a result, if a process starts life as an LX Aware process, it will always be LX Aware. Also, if a process gives itself the `ASG_AWARE` flag, it will be LX Aware for the rest of its existence, as well as any child processes it creates.

Use `getpsf(1)` to view process security flags.

```
% getpsf $$
10249
 None
```

You can also use `secps(1)` to view process security flags, along with all other PitBull LX security attributes.

5.3 Process Attribute Inheritance

PitBull LX security attributes cannot be set on processes directly. Other than using a program written to change its own security attributes, the only means by which a process' security can change is through the execution of an LX Aware file.

In general, there are two rules of inheritance for processes:

- When a process creates a child, the child process inherits all PitBull LX attributes of the parent. That is, in terms of LX security settings, the child is identical to the parent process.
- When a process executes a file, the LX security attributes on the process may change, depending on the file's LX execution domains and flags.

5.4 Setting Process Attributes Using File Security Flags

The `ASG_COMB_EXEC` flag combines the process' access domains and the file's execution domains. The combination is placed on the process before the new process image begins running. However, the process' security flags are masked with the file's execution flags, and the mask of the two sets of flags is placed on the process. This increases the access of the process.

The `ASG_MASK_EXEC` flag behaves in the opposite manner: the process' access domains are masked with the file's execution domains, and the process' security flags are combined with the file's execution flags. The new domains and flags are then placed on the process before the new process image begins executing. This decreases the access of the process.

The behavior of the two flags is defined through the following rules:

- `ASG_COMB_EXEC` causes a process, in general, to become more powerful by adding security domains while removing security flags.
- `ASG_MASK_EXEC` causes a process, in general, to become less powerful by removing security domains and adding security flags.
- If an LX Unaware process executes a file with the `ASG_MASK_EXEC` flag, the process will receive the file's execution domains and execution flags. The only other method by which a process may become LX Aware is to write a program which adds the `ASG_AWARE` flag to the process.
- If an LX Aware process executes a file that is not LX Aware, the process will retain its LX security attributes after the exec.
- When both flags are placed on an executable file, the process that executes the file will have its domains and security flags changed to exactly the file's execution domains and execution flags, regardless of the process' initial security attributes.

Examples

The following examples all assume the process has DAC access to the file to be executed.

Process before executing file	File	Process after executing file
Access Domains	Access Domains	Access Domains
`(--x)fdom1`	`(r-x)fdom1`	`(-w-)fdom2`
`(-w-)fdom2`	`(r--)fdom2`	
`(r--)fdom4`		
`(net)ndom0`		
Flags	Security Flags	Flags
`ASG_AWARE`	`ASG_AWARE`	`ASG_AWARE`
`ASG_RES_MNT`	`ASG_MASK_EXEC`	`ASG_RES_MNT`
		`ASG_RES_SYS`
	Execution Domains	
	`(r--)fdom1`	
	`(rw-)fdom2`	
	Execution Flags	
	`ASG_AWARE`	
	`ASG_RES_SYS`	

In the above example, the executable file has the `ASG_MASK_EXEC` security flag. Thus, when the process executes the file, the process' domains are changed to the mask of the process' access domains and the file's execution domains, and the process' security flags are changed to the combination of the process' security flags and the file's execution flags. Note that if the process did not have the execute bit of domain `fdom1`, it would not have been able to execute the file.

Process before executing file	File	Process after executing file
Access Domains	Access Domains	Access Domains
`(--x)fdom1`	`(r-x)fdom1`	`(r-x)fdom1`
`(-w-)fdom2`	`(r--)fdom2`	`(rw-)fdom2`
`(r--)fdom4`		`(r--)fdom4`
`(net)ndom0`		`(net)ndom0`
Flags	Security Flags	Flags
`ASG_AWARE`	`ASG_AWARE`	`ASG_AWARE`
`ASG_RES_MNT`	`ASG_COMB_EXEC`	
	Execution Domains	
	`(r--)fdom1`	
	`(rw-)fdom2`	
	Execution Flags	
	`ASG_AWARE`	
	`ASG_RES_SYS`	

The preceding example is identical to the example further above, except that the executable file now has the `ASG_COMB_EXEC` flag. After the new process image has begun running, its access domains are the combination of the process' initial domains and the file's execution domains, and its final security flags are the mask of the process' initial security flags and the file's execution flags.

Process before executing file	File	Process after executing file
Access Domains	Access Domains	Access Domains
`None`	`(r--)fdom1`	`(r-x)fdom1`
		`(net)ndom1`
Flags	Security Flags	Flags
`None`	`ASG_AWARE`	`ASG_AWARE`
	`ASG_MASK_EXEC`	`ASG_RES_SYS`
	Execution Domains	
	`(r-x)fdom1`	
	`(net)ndom1`	
	Execution Flags	
	`ASG_AWARE`	
	`ASG_RES_SYS`	

The above example demonstrates the execution of a file with the `ASG_MASK_EXEC` flag by a process that is not LX Aware. The process receives the file's execution domains and flags. Note that if an LX Unaware process executes a file with the `ASG_COMB_EXEC` flag and without the `ASG_MASK_EXEC` flag set, the process will not inherit any security attributes because the process' security flags are masked with the file's execution flags, resulting in an empty set.

Two important features regarding these two flags are:

- The `ASG_COMB_EXEC` and `ASG_MASK_EXEC` flags can be set only by a superuser process that is not LX Aware.
- `ASG_MASK_EXEC` and `ASG_COMB_EXEC` only work on binary executables.

Example

In this example you will copy the `sh(1)` shell to your current working directory and set attributes on it to demonstrate how security flags work. Since you will be setting `ASG_COMB_EXEC` and `ASG_MASK_EXEC`, and using `setfdomex(1)` and `setfsfex(1)` commands, make sure you are running as the superuser, and that your shell process is not LX Aware.

Start by copying the file `/bin/sh` to the current directory:

```
% cp /bin/sh .
```

Execute the shell and use `getpdom(1)` to find the shell's attributes. Be sure to specify`./sh`, so that the local copy is always used.

```
# ./sh

# getpdom $$
1234
 Read_eff:
 Write_eff:
 Exec_eff:
 Net_eff:
 Read_max:
 Write_max:
 Exec_max:
 Net_max:
```

No attributes appear. Neither the previous shell process nor the `./sh` executable has any security attributes for the new shell to inherit.

Exit from the shell.

```
# exit
exit
```

Set an execution domain on the file (the read bit of `fdom1`), but do not set any security flags. Run the shell, and view the shell process's security attributes. Then exit the shell. Note that when you add a domain to a file, it is automatically set `ASG_AWARE`.

```
# setfdomex –S –r fdom1 ./sh

# secls ./sh
-rwxr-x-r-x 1 root root 61 Oct 02 10:59 ./sh
 Access File Domains:
   None
 Execution File Domains:
   (r--)fdom1
 Execution Network Domains:
   None
File Security Flags:
   ASG_AWARE
File Execution Flags:
   ASG_AWARE

# ./sh

# getpdom $$
1240:
 Read_eff:
 Write_eff:
 Exec_eff:
 Net_eff:
 Read_max:
 Write_max:
 Exec_max:
 Net_max:
```

```
# exit
exit
```

There are no domains on the local `./sh` process.

Now assign the `ASG_MASK_EXEC` security flag to the `./sh` binary and check the settings on the file. This flag will assign all of the file's execution domains and execution flags to the process.

```
# setfsf –S ASG_MASK_EXEC ./sh

# secls ./sh
-rwxr-xr-x 1 root root 61 Oct 02 10:59 ./sh
 Access File Domains:
   None
 Execution File Domains:
   (r--)fdom1
 Execution Network Domains:
   None
 File Security Flags:
   ASG_AWARE,ASG_MASK_EXEC
 File Execution Flags:
   ASG_AWARE
```

Run the shell and view the settings.

```
# ./sh

# getpdom $$
1250:
 Read_eff:  fdom1
 Write_eff:
 Exec_eff:
 Net_eff:
 Read_max:  fdom1
 Write_max:
 Exec_max:
 Net_max:
```

The local copy of `sh` now has the execution bit of `fdom1`. The effective and max domain sets are set equal on binary execution unless otherwise hardcoded into a binary.

Use `getpsf(1)` to view process security flags only.

```
# getpsf $$
1250
 ASG_AWARE
```

The process is now LX Aware.

Use `secps(1)` to view all the process's security attributes, then exit the shell.

```
# secps $$
1250 root root
 Effective File Domains
   (r--)fdom1
 Effective Net Domains
   None
 Maximum File Domains
   (r--)fdom1
 Maximum Net Domains
   None
 Security Flags
   ASG_AWARE

# exit
exit
```

5.5 Setting Process Attributes Using lxexec

`lxexec(1)` is provided as a means to run programs that are LX Aware without modifying the target program binary's execution domains and flags. The desired domains and flags are specified on the command line, and the target program is run as specified.

Example

From an LX Unaware shell:

```
# lxexec –r fdom1 –w fdom3 -f ASG_RES_SYS -- /bin/sh

# secps $$
1265 root root
 Effective File Domains
   (r--)fdom1
   (-w-)fdom3
 Effective Net Domains
   None
 Maximum File Domains
   (r--)fdom1
   (-w-)fdom3
 Maximum Net Domains
   None
 Security Flags
   ASG_AWARE,ASG_RES_SYS

# exit
exit
```

Here, `lxexec` executes the `sh` shell with the read bit of `fdom1` and the write bit of `fdom3`, as well as the `ASG_RES_SYS` process security flag. Once the shell is running, `secps` reveals that these are, indeed, the new attributes of the process.

If you are using `lxexec` to execute a binary that has either the `ASG_MASK_EXEC` or `ASG_COMB_EXEC` flags set, the behavior of those flags is also honored. For a binary that has only the `ASG_MASK_EXEC` flag set, the resulting process' domains will be a mask of the binary's execution domains and the command line arguments to `lxexec`. The resulting process' security flags will be a combination of the binary's execution flags and the command line arguments to `lxexec`. For a binary that has only the `ASG_COMB_EXEC` flag set, the resulting process' domains will be a combination of the binary's execution domains and the command line arguments to `lxexec`. The resulting process' security flags will be a mask of the binary's execution flags and the command line arguments to `lxexec`. If the binary has both the `ASG_COMB_EXEC` and `ASG_MASK_EXEC` flags set, the resulting process' domains and security flags will be equal to the binary's execution domains and execution flags (the use of `lxexec` will have no effect on the resulting process).

If your process has LX attributes before you run `lxexec`, then you cannot specify any domains on the command line that your process does not already have nor can you omit any flags on the command line that your process already has. If you attempt this, then `lxexec` will not be able to execute the target program specified on the command line. This is true even if the target program has the `ASG_MASK_EXEC` or `ASG_COMB_EXEC` flags. For these reasons, `lxexec` is most effective when run by an LX unaware process.

Chapter 6: Working with Network Data Attributes

Each network rule consists of two parts: a set of properties that are to be compared against data passing through the network stack, and a set of LX network domains. For a rule to match the data, the rule's properties must exactly match the properties of the data (examples of data properties are: source and destination address, ports, interfaces, protocols, etc.). For example, if a rule specifies a particular destination machine for incoming data, then data bound to that machine will be tagged with LX network domains specified in that rule. If no rule can be found that exactly matches the data's properties, no domains are assigned. The kernel will not select the "closest" rule.

When network data passes up or down through the layers of the network stack, it is tagged with LX security domains. For incoming data, the domain and the `ASG_AWARE` flag is placed on the data according to the set of network rules that have been loaded into the kernel. If no matching rule can be found, the `ASG_AWARE` flag will not be placed on the data. A check is then made when a process attempts to access the data, according to the rules established in Chapter 3.

The process for outbound data is similar. When the data is sent, the data will inherit the process' network domains and network security flags. Before it is transmitted over the network, a check is made, again according to the rules established in Chapter 3.

6.1 Netrules Configuration File

The default network rules configuration file is `/etc/argus/netrules`.

6.2 Setting and Viewing Netrules

`setnetrule(1)` loads a network rule set into the kernel. If it is given no options, the command will use the default rule file `/etc/argus/netrules`. An alternate rule file may be specified.

`getnetrule(1)` is used to retrieve the current network rule set from the kernel and write it to the standard output. It writes the rule set in a format that can be edited and used by `setnetrule(1)`.

6.3 Netrules structure

A netrules file can contain both rules and comments. A comment is a line that begins with a '#' character. Each rule in the file has a format of:

```
[proto:<p>] [<specification>:<s>] [<extension>:<e>]
domain:<d>
```

The fields may be in any order.

The *`proto`* field specifies which protocol the rule is to match. Valid protocols are `tcp`, `udp`, `icmp`, or a protocol number (as listed in `/etc/protocols`). Additionally, *`p`* may be the `*` character, which denotes all protocols. If the character immediately following the `:` character is `!`, followed by a valid protocol designation, the rule will match all protocols except the specified protocol. If the `proto` field is not specified for a given rule, that rule will match all protocols (the same behavior as the `*` designation). If the protocol is specified as `!*`, the rule will not match any protocol.

The *`specification`* field provides additional granularity for rule-matching. If a particular specification is not given, the rule will not check for that particular information, and thus will match by default. Valid specifications are:

`inif:[!]`*`name`*

> This specifies the network interface name through which data is received. If the interface name ends with a `+` character, the rule will match with any interface name that begins with the string before the `+` character. If a `!` character is the first character in the name, the result of a match will be reversed.

`outif:[!]`*`name`*

> This specifies the network interface name through which data is sent. Otherwise, the behavior is the same as `inif`.

`saddr:[!]`*`addr`*`[`*`/mask`*`]`

> This specifies the source address of the data. The address must be supplied as either a hostname or a protocol address. The mask is the IPv4 net-mask to use. It can be specified in either the traditional or modern form (e.g. `/255.255.255.0` or `/24`, respectively). If the `!` character is supplied, it reverses the result of a match. If the *`saddr`* specification is not given, the rule will apply to all source addresses.

`daddr:[!]`*`addr`*`[`*`/mask`*`]`

> This specifies the destination address of the data. Otherwise, `daddr` behaves the same as the `saddr` specification.

The `extension` field is a protocol-specific specification. They behave exactly the same as specifications, except that they are only valid for particular protocols. If more than one protocol has been specified (such as through the use of `proto:*`), the extensions will be used only for the protocols they apply to, and ignored where they are invalid. Valid extensions are:

`sport:[!]`*`port`*`[:`*`uport`*`]`

> This specifies the range of source ports for this rule. If the `!` character is supplied, the result of a match is reversed. *`port`* specifies a specific port to which this rule applies. If *`uport`* is also given, then the rule will apply to a range of ports from *`port`* up to, and including, *`uport`*. `sport` applies only to the TCP and UDP protocols.

`dport:[!]`*`port`*`[:`*`uport`*`]`

This specifies the range of destination ports for this rule. Otherwise, it is identical to the *`sport`* extension.

The `domain` field is the only field that is required in a rule. It is a comma-separated list of network domains as specified in `/etc/argus/ndommap`. If the `!` character immediately follows the `:` character, the list of domains is inverted. If the `*` character is used, the rule will contain every network domain. If no domains are given, the rule will have no network domains.

6.4 Defining Netrules

Similar to some firewall products, PitBull LX network rules should be written with the most specific rules first, and the rules that will be matched most often should come before rules that will be less-often matched, so that the average number of checks against a given network packet before a rule is found is minimized. For example, on a machine that is primarily used as a web server, a rule that checks for incoming connections to port 80 should be positioned before any checks for incoming connections to other services.

Note: the following examples all assume the local machine is named `localserv`, and the remote machine is named `remoteserv`.

Example

The following will place the network domain `ndom3` on all incoming packets destined to the local `telnet` port. It makes use of the fact that the `telnet` protocol uses TCP and that the well-known `telnet` port is 23:

```
proto:tcp daddr:localserv dport:23 domain:ndom3
```

The following associates `ndom2` with all `ftp` requests to the standard `ftp` port, either inbound or outbound:

```
proto:tcp dport:21 domain:ndom2
```

The following rule associates all domains except `ndom4` with all traffic passing through any device starting with `eth`:

```
inif:eth+ outif:eth+ domain:!ndom4
```

The following establishes `ndom5` as the default domain. If a network rule with no protocol, specification, or extension field, such as the following, is placed in a network rules file, all network data will match it:

```
domain:ndom5
```

More Thorough Example

The well-known TCP port for mail transfer is 25. If the network domain reserved for the mail system is called `netmail`, then a network rule for incoming mail would be:

```
proto:tcp daddr:localserv dport:25 domain:netmail
```

The reasoning behind the rule is as follows:

- By specifying TCP as the protocol, the rule will not match packets destined to UDP port 25 (although, chances are, nothing will be listening on that port).
- By specifying the destination address as the local machine, only incoming traffic is matched (if `daddr` were omitted, the rule would match both incoming and outgoing traffic).
- By specifying the destination port, the rule will match only network traffic that will be accepted by the mail transfer agent.
- The domain placed on the incoming network traffic is `netmail`. This assures that only a process with the `netmail` network domain will be able to read the data.

The mail transfer agent (such as `sendmail` or `qmail`) must have the `netmail` network domain to be able to read the incoming mail. Any LX Aware process without `netmail` will be unable to read the data, even if it has successfully established a TCP connection.

If the mail transfer agent does not have the `ASG_AWARE` security flag, it is running in an unsecured state, and will be able to read the network data regardless of the security restrictions placed on the data by PitBull LX.

In the case of a program sending mail to the remote server, the rule

```
    proto:tcp daddr:remoteserv dport:25 domain:netmail
```

can be used. Its fields are similar to those above, except that it requires an LX Aware process to have the `netmail` domain when sending mail to the remote server.

A more general rule can also be created:

```
    proto:tcp dport:25 domain:netmail
```

The rule will give the same behavior as above, but now, without the `daddr` field, the rule will apply to traffic that is both inbound and outbound, to and from any server. Thus, no LX Aware process without the `netmail` domain will be able to send or receive data via TCP to port 25.

Chapter 7: Customizing Domain Mapping Files

The system makes domain comparisons based on bits set in data fields in memory. You cannot read these bits directly. The system provides two files that provide the human-readable names for the file and network domains: `/etc/argus/fdommap` for file domains, and `/etc/argus/ndommap` for network domains. When the system compares domains, it ignores these files and bases its comparison on the actual bits set. However, PitBull LX commands use the files to provide textual names for domains in their output.

7.1 Domain Map Files

The system administrator may assign text-string labels, up to 32 characters, to any or all of the file domain and network domain bit-numbers. The format of the entries in these files is discussed below.

These labels are not stored with the objects (files and processes) themselves. Instead they are translated by the PitBull LX applications as necessary. Changes to the `fdommap` and `ndommap` files are therefore immediately apparent to users.

Because these human-readable labels are presented to users when PitBull LX applications print lists of domains, it is important to protect these files with both DAC and PitBull LX security measures. It is recommended that these files be given a user ID and group ID for a user who is not able to normally log in, such as `bin`. Write permission should only be given to the administrator. You can do this by not allowing DAC write permission to any user, and by adding a unique file access write domain to both of these files and the superuser shell.

7.2 Map File Entry Format

The two files contain a fixed number of entries to provide human-readable labels to their domains. There must be exactly one entry per domain. There are 1,024 file access domains and 512 network domains. Therefore there are exactly 1,024 entries (0 through 1023, inclusive) in `fdommap`, and 512 entries (0 through 511, inclusive) in `ndommap`. Each entry has a default label upon installation, `fdom0` through `fdom1023` and `ndom0` through `ndom511`. For both files, domains 0 through 4 are marked valid on system installation, and the rest are marked invalid.

Not every line in the file needs to contain a translation entry, and not all text on a line is considered part of an entry. Any text following a pound (#) character is considered a comment and is ignored by the translation routines. Blank lines, lines with white space at the head of the line, and comment-only lines are also ignored.

The entries in both files follow the same format:

```
bit-number:valid:label
```

There must be no white space before or within any of the fields. The *`bit-number`* field must begin at the first character of the line, and must contain only numbers

(`0-9`). There must not be any white space between the bit-number and the field separation character (:). The *`valid`* field indicates the status of the textual name. This field may contain either a `v` for valid or an `i` for invalid. There must not be any white space between the valid field and the field separation character (:). The *`label`* field can contain only numbers, lowercase letters, uppercase letters and the hyphen (`-`) or underscore (`_`) characters. There may be any amount of white space after the label. Comments are also allowed after an entry. Labels that are longer than 32 characters will be truncated to 32 characters. However, if you specify labels longer than 32 characters, you may inadvertently create two labels that appear different but truncate to identical 32-character labels. It is wise to restrict your labels to 32 characters.

A file access domain mapping file may contain entries like this.

```
0:v:sys    # System critical files
1:v:html   # Webserver related files
2:v:cgi    # CGI scripts and configuration files
3:v:sql    # SQL database tables and configuration
4:i:fdom4
```

A network domain mapping file may contain entries like this.

```
0:v:eth0     # Ethernet device
1:v:port80   # Webserver traffic
2:v:port443  # Secure webserver traffic
3:i:ndom3
4:i:ndom4
```

7.3 Adding and Removing Domain Names

The `chdommap(1)` command allows an administrator to add and remove domain names in an automated fashion. When adding a domain name, the utility searches for the first invalid entry in the appropriate map file, enters the new name, and marks the entry as valid. When removing a domain name, the utility searches for the name and marks the corresponding entry as invalid. Using this command to delete a domain name will not remove references to it on the system. Therefore, care must be taken when manipulating the domain map files. It is important to keep in mind that, since `chdommap` finds the first invalid domain and replaces it with the specified domain name, it will overwrite previously deleted domains. The following example illustrates using the command and its effects on the domain map file:

Partial content listing of `/etc/argus/fdommap` before using `chdommap`:

```
0:v:sys    # System critical files
1:v:html   # Webserver related files
2:v:cgi    # CGI scripts and configuration files
3:v:sql    # SQL database tables and configuration
4:i:fdom4
5:i:fdom5
```

Use `chdommap(1)` to add a new file domain named `new_domain.`

```
% chdommap –A –f new_domain
```

Partial content listing of /etc/argus/fdommap after adding a new file domain using chdommap:

```
0:v:sys    # System critical files
1:v:html   # Webserver related files
2:v:cgi    # CGI scripts and configuration files
3:v:sql    # SQL database tables and configuration
4:v:new_domain
5:i:fdom5
```

Use chdommap(1) to effectively delete the file domain named “new_domain”.

```
% chdommap –D –f new_domain
```

Partial content listing of /etc/argus/fdommap after removing a file domain using chdommap(1):

```
0:v:sys    # System critical files
1:v:html   # Webserver related files
2:v:cgi    # CGI scripts and configuration files
3:v:sql    # SQL database tables and configuration
4:i:new_domain
5:i:fdom5
```

The ‘i’ indicates that new_domain has been invalidated and, effectively, deleted.

7.4 Modifying Domain Names Manually

If you assign a file access domain to a file, and then rename that domain in /etc/argus/fdommap, the new domain name appears when you run getfdom(1).

Create a new file and make a file domain assignment. View the file domains.

```
% touch test3

% chdommap –A –f fdom11
% setfdom –S –r fdom11 test3

% getfdom test3
test3
 Access Read:  fdom11
 Access Write:
 Access Exec:
```

Change the label of the fdom11 entry in /etc/argus/fdommap.

For example, you could change:

```
11:v:fdom11
```

To this:

```
11:v:newfdom11
```

View the results.

```
% getfdom test3
test3
 Access Read:  newfdom11
```

```
        Access Write:
        Access Exec:
```

7.5 Verifying Domain Map Files

Before changing the system domain maps, it is a good idea to make a copy of the default files, make changes to the copy, and verify that there are no errors in the copy. After verification, it is safe to replace the system default file with the changed copy. The utility `chkdommap(1)`verifies a domain map file by checking for correct formatting. Any deviation from the correct format is reported in detail to standard output.

`chkdommap(1)` accepts one of two options. Use `-f` *`fdom_file`* to verify a file containing a file domain map, and `-n` *`ndom_file`* to verify a file containing a network domain map. If no options are specified, `chkdommap(1)` checks the format of both `/etc/argus/fdommap` and `/etc/argus/ndommap`.

Chapter 8: Setting Security Attributes upon Login Using PAM

By design, users who login to a freshly installed PitBull LX system have no PitBull LX attributes on their shell process. If you want to assign attributes to users, you must configure the system to make those assignments during login. You can configure your system so that any user's process is LX Aware on login and thus subject to PitBull LX security.

PitBull LX uses a PAM (Pluggable Authentication Module) module to implement attribute assignment on login. This mechanism helps guarantee PitBull LX enforcement on your system. With the PitBull LX PAM module, you can:

- Set attributes on a per-user basis.
- Set different attributes for console logins and network logins, for the same user.
- Set attributes through any or all of the PAM-aware login mechanisms
- The PitBull LX PAM module may be added to only those applications that need it.

Your operating system must use PAM-aware login applications such as `ssh`, `login`, and `ftp` to make use of this feature. Most current Unix/Linux distributions ship with PAM-aware applications.

This chapter of the guide assumes that you are familiar with the PAM Administrator's or System Administration Guide for your system. Consult such documentation before attempting to integrate the PitBull LX PAM module into your system.

Downloadable documentation on PAM administration and programming is available, as well as PAM source code, on the Linux-PAM web site: http://www.us.kernel.org/pub/linux/libs/pam.

For a PAM-aware login application to assign PitBull LX attributes to users who login, you must configure the application to use the PitBull LX PAM module, and configure the PitBull LX attributes configuration file `/etc/argus/users` to specify the attributes to be set on users.

You must configure PAM so that the application calls PitBull LX's PAM module. An example of configuring PAM to use `pam_argus.so` for the `login` command follows.

Add the following line to `/etc/pam.d/login`:

```
   auth   required /lib/security/pam_argus.so
```

By default, the pam_argus modules will attempt to set the file access domains specified in `/etc/argus/users` for a user on that user's pseudo terminal. In order for the access domains to be set on a user's pseudo terminal, the

`/dev/pts` filesystem must be enabled in the Linux kernel. If the `/dev/pts` file-system has been enabled and pam_argus is unable to set the access domains on the user's pseudo terminal, then the login authorization will fail and the user will not be allowed to login. This behavior can be overridden by adding **`secure_pts=no`** as an option to the pam_argus module.

An example entry with the `secure_pts=no` option is:

```
auth required /lib/security/pam_argus.so secure_pts=no
```

It is recommended to either have `requisite` or `required` as the control type. Any other points of entry that you want configured with LX attributes will need to have the same line added.

8.1 The User Attribute Configuration File

The attribute configuration file, `/etc/argus/users`, specifies the attributes to be set on users. Each entry in the file specifies the attributes for a user logging in through the network, at the console, or both. The format for configuration entries is as follows.

```
username:login-path:access-domains:security-flags
```

All four fields are **required** for each configuration entry. Any amount of white space may precede or follow a configuration entry, and blank lines are ignored. All text following a pound (#) character is considered a comment and ignored. Lines that contain only a comment are ignored.

Use the backslash character (\ ) to continue an entry on the following line. Any number of spaces or tabs may precede the backslash character. Comments may follow the line-wrap character.

Here are the meanings of the fields:

`username`	This is the user's UNIX login name. You can also place `ARGUS_DEFAULT` in this field. If so, the attributes in this entry are set for any user who is not otherwise specified in the `users` file.

`login-path` This field contains one of the three following entries:

`console`

Attributes will be applied if the user is logging in via a console device (the login session corresponds to a `/dev/tty*` terminal device).

`net`

Attributes will be applied if the user is logging in via a non-console device (the login session corresponds to non-console terminal devices, such as `/dev/pts/0).`

`any`

Attributes will be applied for the user regardless of corresponding terminal device.

`console` and `net` entries take precedence over `any` entries. For example, if a user logging in at a console has both a `console` entry and an `any` entry, the user will get the attributes specified in the `console` entry.

`access-domains` This field contains a comma-seperated list of file domains and network domains to be set for the matching login session. No white space is allowed between items in the list. Any number of access domains may be specified in the following formats:

File Access Domain label([r|-][w|-][x|-])

Network Access Domain label(net)

Read, write and/or execute file access domain bits may be set by specifying the `r`, `w`, and/or `x` flags in the domain specification. A dash (`-`) signifies that the corresponding read/write/execute bit is not to be set for this session. File domains and network domains may be listed in any order. An empty list means that no domains are to be set for this session.

If you change the name of a domain in a domain map file after placing the name in the PAM file, you must also change the name in the PAM file. Otherwise, the system will read the old name in the PAM file and will not be able to translate it to a number, and the user will not be able to login.

security-flags This field contains a comma-separated list of security flags to be set for the matching login session. No white space is allowed between items in the list. You can enter only flags that are valid for processes.

Flags may be specified in any order. If no flags should be set for this session, an empty list may be given.

Consider the following example configuration lines:

```
ARGUS_DEFAULT:any:sys(r-x),eth0(net):ASG_AWARE, \
  ASG_SEC_ACC_FS,ASG_RES_SYS, \
  ASG_RES_MNT,ASG_RES_RBT

root:net:sys(r-x),root(r-x),eth0(net): \
  ASG_AWARE,ASG_SEC_ACC_FS,ASG_RES_SYS

root:console::# superuser console logins are not
              # ASG_AWARE
```

With this configuration, a user logging in as `root` will have different PitBull LX attributes for console sessions and network sessions. If the user logs in through the network, he will have the read and execute bits for the `sys` and `root` file domains, the `eth0` network domain, and the `ASG_AWARE`, `ASG_SEC_ACC_FS`, and `ASG_RES_SYS` flags.

If the user logs in through a console, the resulting shell process will not have any file domains, network domains, or flags (provided that they were not set through another mechanism, such as the `ASG_MASK_EXEC` flag set on the login shell, etc.).

The console line prevents the ARGUS_DEFAULT line from being applied to users logging in as root at the console. If this line did not appear in the configuration, the ARGUS_DEFAULT line would be applied.

In processing the `/etc/argus/users` file, if there are duplicate login name and login path rules for a user, the system assigns the first rule from the top.

8.2 Adding to or Modifying the User Login Attribute Configuration

The `chup(1)` command is used to add to or modify the user login attributes. For a complete description of the options for this command refer to the manual page.

To add the ‘`sys`’ read and execute domains to the root user when logging in over the network:

```
# chup -A -d ‘sys(r-x)’ net root
```

To remove the ASG_SEC_ACC_NET flags from the `webadmin` user logging in from a console and put the date in comment above that entry:

```
# chup -D -f ‘ASG_SEC_ACC_NET’ -c “‘date‘” console
webadmin
```

8.3 Verifying the User Login Configuration File Format

Use `chkup(1)` command to verify the user login configuration file format. `chkup` will report any problems with the formatting of the configuration file to standard output. Refer to the manual page for complete descriptions for this command.

To check the format of the default user login configuration file (`/etc/argus/users`).

```
% chkup
```

The check the format of the specified user login configuration file.

```
% chkup filename
```

Chapter 9: Creating and Using Integrity Databases

You can save the file attributes and PitBull LX security attributes of any set of files on your system in a file called an integrity database. This database may be used to verify the integrity of the system's files or restore the security attributes after a backup or in the event of a failure.

9.1 Integrity Database Format

Integrity database files are editable ASCII text files. Each integrity database entry consists of pipe(`|`)-delimited fields giving:

- The full pathname of the file.
- The file type (directory, regular file, symbolic link, etc.)
- The file's permission bits in octal format.
- The name of the file's owner.
- The name of the file's group.
- The MD5 checksum of the file, if it is a regular file.
- The major and minor device numbers, if it is a block or character device file.
- The PitBull LX file access domains.
- The PitBull LX file access security flags.
- The PitBull LX file execution domains.
- The PitBull LX file execution security flags.

The human-readable domain labels are used in the access and execution domain fields. Therefore, if a domain label is made invalid or its name is changed after an integrity database is created for files that use that domain, those entries will produce an error and be skipped. However, this problem can be corrected by replacing all occurrences of the old domain label in the database(s) with the new domain label.

The format of an integrity database is discussed further in the `integrity(5)` manual page.

9.2 Creating an Integrity Database

The `mkinteg(1)` utility may be used to create an integrity database. `mkinteg(1)` reads filenames from standard input and outputs an integrity database entry for each filename to either standard output or a filename given with the `-o` option. Standard UNIX commands, such as `find(1)`, may be used to create an integrity file for a select group of files:

```
% find . | mkinteg -o /tmp/backup.sec
% find /etc | mkinteg >> /tmp/backup.sec
```

If `mkinteg(1)` is given the name of a nonexistent file or is unable to read the security information of a given file, it will abort and display an error message. Likewise, if `mkinteg(1)` is unable to create the output file given with the `-o` option, it will abort and display an error message.

9.3 Using and Maintaining an Integrity Database

The `chkinteg(1)` utility may be used to:

- Compare the file's current attributes with those stored in an integrity database, interactively prompting you to update the database file or restore the attributes of the file.
- Restore attributes to a group of files from an integrity database.
- Update an integrity database after changing a file's attributes.
- Update an integrity database's MD5 checksums for files whose contents have changed.

`chkinteg(1)` reads integrity database entries from standard input, from a file whose name is given with the `-f` option, or from all files in a directory whose name is given with the `-d` option. For example, the following `chkinteg(1)` commands are valid:

```
# grep '/etc/' /etc/argus/integrity | chkinteg
# chkinteg -cif /etc/argus/integrity
# chkinteg -U -d /etc/argus/integ.d
```

If the `-i` option is used, `chkinteg(1)` may be used interactively to verify the file and security attributes for the files in the given integrity database(s); the `-f` or `-d` option must be used with the `-i` option. If there is any difference between the current attributes of a file and the attributes in its integrity entry, the user will be given a list of actions which can be taken:

- repair (restore the attributes of) the file.
- repair all remaining files.
- update the integrity database entry for this file.
- update the integrity database entries for all remaining files.
- take no action for this file.
- take no action for any of the remaining files.

The `-c` option must be given if the checksums are to be generated and compared to database(s).

If the `-r` option is used, `chkinteg(1)` will repair files that have integrity entries given from standard input or from integrity databases given with the `-f` or `-d` options. Each file that has different attributes from those given in its integrity entry will have its attributes reverted to those in the entry. Note: checksums, file types and major/minor device numbers cannot be restored through this action.

If the `-u` option is used, `chkinteg(1)` will update integrity databases that are given with the `-f` or `-d` options. For each file that has different attributes than the attributes in its integrity entry, the integrity entry will be changed to match the current attributes. The file's MD5 checksum will not be updated unless the `-c` option is given.

If the `-U` option is used, `chkinteg(1)` will update only the checksums of integrity entries listed in databases given with the `-f` or `-d` options. For each file that has a different checksum than the one given in its integrity entry, its integrity entry will have its checksum field updated to the current value.

If the `-t` option is given, `chkinteg(1)` will use the directory whose name is given as the directory where temporary files can be created. This directory will be created if it doesn't already exist.

If the `-R` option is given, `chkinteg(1)` will prepend the given directory name to the filenames in the integrity entries in the integrity databases supplied with the `-f` or `-d` options or from standard input.

If `chkinteg(1)` encounters an error in an entry, such as a listed file that does not exist or the inability to retrieve security information for a listed file, that entry will be skipped, a message will be printed to standard error and `chkinteg(1)` will continue processing entries.

Appendix A: Example System Security Setup

This example shows the basic essentials on how to use PitBull LX to secure a system and two subsystems, HTTP and email. The example will take you through a highly restrictive setup that can later be opened up as needed.

It is important, when planning your security system, to keep in mind the basic fundamentals of PitBull security. Security is only as good as the planning put into it.

This example may differ from your own system architecture. Keep in mind that this example may not work completely on your own system and plan changes accordingly.

A.1 Base System Security

A.1.1 Turn off Unused Services

While planning, identify what services system users would be accessing. To simplify securing the system and to limit the exposure of the system, begin securing the machine by turning off all services that are not needed, For example, if your users do not need to login via telnet, then disable telnet. Otherwise, leaving the telnet server on provides an unnecessary entry point for intruders.

A.1.2 Establish System Domains

Access domains are used to compartmentalize the system and secure files and directories. These domains keep unauthorized users from having access to filesystem objects that they may compromise.

This scenario uses the following file domains:

`sys`	Protect system files and binaries
`httpd`	Protect configuration files and binary programs for HTTP
`httpd_logs`	Protect the HTTP subsystem’s log files
`httpd_cache`	(if apache CacheRoot enabled) protect `/var/cache/httpd`, without allowing `httpd` to write to the entire `httpd` compartment
`mailqueue`	Restrict access to the mail queue to the `sendmail` process
`mailspool`	Restrict access to the mail spool to the `sendmail` process

The following network domains are used, as well:

`net_http`	Assure `httpd` accesses only HTTP network traffic, and that no other processes can access this traffic
`net_sendmail`	Assure that `sendmail` accesses only `sendmail` network traffic, and that no other processes can access this traffic
`net_imapd`	Assure that `imapd` accesses only `imapd` network traffic, and that no other processes can access this traffic
`net_popd`	Assure that `popd` accesses only `popd` network traffic and that no other processes can access this traffic.

Create these domains with the `chdommap` command.

```
# chdommap –A –f sys
# chdommap –A –f httpd
# chdommap –A –f httpd_logs
# chdommap –A –f httpd_cache
# chdommap –A –f mailqueue
# chdommap –A –f mailspool
# chdommap –A –n net_httpd
# chdommap –A –n net_sendmail
# chdommap –A –n net_imapd
# chdommap -A -n net_popd
```

File domain names may be checked with:

```
vi /etc/argus/fdommap
```

Network domain names may be checked with:

```
vi /etc/argus/ndommap
```

The example configuration assumes that all processes will be LX Aware, except for the superuser shell process.

A.1.3 Protect System Files

A.1.3.1 Security Policy

- Only the superuser, in LX Unaware mode, should be able to change system files.
- Individual files should be given the minimal access requirements for a standard users needs.

A.1.3.2 Securing System Files

In this scenario, you need to prevent users from writing to most system files, but allow them to read and execute them. At the same time, you need to allow trusted administrators to write to these files.

No user should ever have the `sys(-w-)` domain. By placing the `sys(-w-)` domain on system files, users passing DAC checks will be able to read and execute the files, but only LX Unaware processes (i.e., the superuser) will be able to write the files.

Begin at the root file system. As the superuser, place the `sys(-w-)` access domain on the `/` directory.

```
# setfdom -S -w sys /
```

Use the `secls command to verify the settings.`

```
# secls -d /
drwxr-xr-x 17 root root 4096 Jan 03 09:26 /
 Access File Domains:
   (-w-)sys
 Execution File Domains:
   None
 Execution Network Domains:
   None
 File Security Flags:
   ASG_AWARE
 File Execution Flags:
   None
```

Apply the same settings to the `/etc` directory. Use the recursive option, `-R`, to set `sys(-w-)` on all files and subdirectories in `/etc` as well.

```
# setfdom -R -S -w sys /etc
```

Next, you need to make sure that any files created in these directories in the future all have the `sys(-w-)` domain set. To do this, set the `ASG_INHERIT_DOM` flag on the directory and all subdirectories.

```
# find /etc -type d | xargs setfsf -S
ASG_AWARE,ASG_INHERIT_DOM
```

Check to see what was just set on the `/etc directory.`

```
# secls -d /etc
drwxr-xr-x 30 root root 4096 Jan 04 15:45 /etc
 Access File Domains:
   (-w-)sys
 Execution File Domains:
   None
 Execution Network Domains:
   None
 File Security Flags:
   ASG_AWARE,ASG_INHERIT_DOM
 File Execution Flags:
   None
```

Also secure the remaining system directories and the files contained under them with `sys(-w-).` It is possible to string the arguments together instead of making a separate command for each directory.

```
# setfdom -R -S -w sys /usr /var /bin /sbin /lib

/boot /root /mnt /lost+found

# find /usr /var /bin /sbin /lib /boot /root /opt /mnt
/lost+found /usr/local -type d | xargs setfsf -S
ASG_AWARE,ASG_INHERIT_DOM.
```

Your system may not have the same directory structure as this example. Also, you may need to alter some of the settings to better suit your security needs. For example, you may wish to allow users to write into the directory `/usr/local`. If so, you would need to remove the `sys(-w-)` domain from that directory.

Most of the files in the `/dev` directory should have read, write, and execute access denied. This protects against damage from devices through `ioctl()` calls. Set the `sys(rwx)` domain on the `/dev` directory and all files under it.

There are some files under the `/dev` directory that users will need to have read and execute access to. Reduce the protection on those particular devices to `sys(-w-)`.

```
# setfdom -R -S -r sys -w sys -x sys /dev

# setfdom -S -r "" -w sys -x "" /dev

# find /dev -type d | xargs setfsf -S
ASG_AWARE,ASG_INHERIT_DOM

# setfsf -S ASG_AWARE /dev/pts

# setfdom -S –r "" -w sys –x "" /dev/argus

# setfdom -S –r "" -w sys –x "" /dev/random

# setfdom -S –r "" -w sys –x "" /dev/urandom
```

Your users also need full access to `/dev/null`, `/dev/zero`, `/dev/pts` and `/dev/tty`. The -C option for `setfdom` will clear all file domains on a file

```
# setfdom -C /dev/null

# setfdom -C /dev/zero

# setfdom -C /dev/pts

# setfdom -C /dev/tty
```

Make the `/tmp` directories LX Aware so that processes running with the `ASG_SEC_ACC_FS` flag can access them. However, many different processes need to be able to read and write into these directories, so it is better to leave off any access domains. Do the same with `/var/run` and /`var/lock`, as a number of processes must access them.

```
# setfsf -S ASG_AWARE /tmp

# setfsf -S ASG_AWARE /var/tmp

# setfsf -S ASG_AWARE /usr/tmp

# setfdom -C /var/tmp
# setfdom -C /usr/tmp

# setfdom -R -C /var/run

# setfsf -R -S ASG_AWARE /var/run

# setfdom -R -C /var/lock

# setfsf -R -S ASG_AWARE /var/lock
```

Finally, protect the user directories with `sys(-w-)`. Users will still need to access their home directories, but they will not need to alter the parent directory. Therefore, set the `sys(-w-)` domain on the main user directory, but not the file security flag `ASG_INHERIT_DOM`.

```
# setfdom -S –w sys /home
```

A.2 Securing OS Subsystems

Securing subsystems consists of compartmentalizing (or isolating) the subsystems to prevent a breach in the security of one of the subsystems from affecting the other subsystems. The subsystems in this scenario are HTTP and email.

A.2.1 HTTP Security Policy

- Only the superuser, in LX Unaware mode, should be able to change vital HTTP files.
- Certain processes need access to HTTP directories. HTTP must be compartmentalized so that a given process may access only what it needs to run successfully.

A.2.1.1 Securing HTTP

To secure the HTTP subsystem, you must know what files make up this subsystem. On Red Hat 6.2, the HTTP subsystem is the Apache web server. The Apache RPM contains configuration files, binaries, libraries, and support files. Find out what files the RPM has installed on the system by locating the RPM and querying it for what it contains.

```
# rpm -qa | grep apache
apache-1.3.12-2

# rpm -qis apache-1.3.12-2

Name        : apache    Relocations: (not relocate-
able)
Version     : 1.3.12         Vendor: Red Hat, Inc.
Release     : 2         Build Date: Wed 01 Mar 2000
12:37:55 PM CST
```

```
Install date: Wed 03 Jan 2001 09:27:08 AM CST  Build
Host: porky.devel.redhat.com
Group       : System Environment/Daemons    Source
RPM: apache-1.3.12-2.src.rpm
Size        : 975071                        License:
Freely distributable and usable
Packager    : Red Hat, Inc. <http://bugz-
illa.redhat.com/bugzilla>
Summary     : The most widely used Web server on the
Internet.
Description :
Apache is a powerful, full-featured, efficient and
freely-available
Web server. Apache is also the most popular Web
server on the
Internet.

Install the apache package if you need a Web server.
normal        /etc/httpd/conf
normal        /etc/httpd/conf/access.conf
normal        /etc/httpd/conf/httpd.conf
normal        /etc/httpd/conf/magic
normal        /etc/httpd/conf/srm.conf
normal        /etc/httpd/logs
normal        /etc/httpd/modules
......
```

(The output of the command has been truncated for brevity.)

Earlier, you created the two file access domains to use here: `httpd` and `httpd_logs.` You also created the network domain: `net_httpd.`

First, secure the HTTP subsystem by protecting the Apache binary with the `sys(-w-)` domain.

```
# setfdom -S -w sys /usr/sbin/httpd
```

When the `httpd` binary is executed, the resulting process image should run in a secure state where it can read, write, and execute only files that are in its compartment. The `httpd` process will need the ability to read and execute files within the `httpd` and `httpd_logs` compartments, and the ability to write within the `httpd_logs` compartment. Set the network access domain on the `httpd` process to `net_httpd`. Using this network domain setting allows the process to read and write network data only within the `net_httpd` network domain. Shortly, you will protect the network data so that the `httpd` process can read and write no other network data.

Use `setfdomex` to set the execution domains that the `httpd` process will need.

```
# setfdomex -S -r httpd,httpd_logs -w httpd_logs -x
httpd,httpd_logs -n net_httpd /usr/sbin/httpd
```

The resulting process image should have at most the execution domains that have been set on it, so set the `ASG_MASK_EXEC` file security flag on the binary.

```
# setfsf -S ASG_MASK_EXEC /usr/sbin/httpd
```

For the `httpd` process, file access should be restricted to files and network data that are LX aware. Therefore, set the `ASG_SEC_ACC_FS` and `ASG_SEC_ACC_NET` flags. Also, restrict the httpd process from calling the reboot system call with the `ASG_RES_RBT` flag, and from mounting and unmounting file systems with the `ASG_RES_MNT` flag.

```
# setfsfex -S
ASG_SEC_ACC_FS,ASG_SEC_ACC_NET,ASG_RES_RBT,
ASG_RES_MNT /usr/sbin/httpd
```

At this point, the main files are protected. Now the other files that make up the `http` subsystem need to be protected.

Earlier, you used the `rpm` command to list all the files the Apache RPM installed on the system. That list provides a convenient way of locating and assessing which files will need protection.

There are a number of files that were installed into the `/etc` `/usr/bin,` `/usr/lib,` `/usr/man` and `/usr/sbin` directories. Those files were already secured with the `sys(-w-)` domain, and since `httpd` will have no need to write to these files, their current security settings are appropriate.

The next collection of files was installed in the httpd home directory. To allow the `httpd` process to read and execute the files contained in these directories, set the appropriate access domains on them.

```
# setfdom -R -S -r httpd -w httpd -x httpd /home/
httpd

# find /home/httpd -type d | xargs setfsf -S
ASG_AWARE,ASG_INHERIT_DOM
```

The `httpd` logs will need to be writable by the `httpd` process, so you need to replace the `sys(-w-)` domain already in place on the directory. You have already placed the `httpd_logs` domain on the `httpd` binary. Now place it on the directory. A separate `httpd_logs` domain is used so you can grant a process write access to the log files without granting access to any other part of the `httpd` domain. As with previous directories, the `ASG_INHERIT_DOM` flag ensures that any new files created in the directory also have the same security settings.

```
# setfdom -R -S -r httpd_logs -w httpd_logs -x
httpd_logs /var/log/httpd

# setfsf -S ASG_AWARE,ASG_INHERIT_DOM /var/log/httpd
```

The other directory is `/var/cache/httpd`. Assume you chose to enable the Apache CacheRoot directive. Apply `httpd_cache(rwx)` to both the `/var/cache/httpd` directory and the `httpd` binary's execution domain so you can allow a process to write to the cache without being able to write to the entire `httpd` compartment.

```
# setfdom –R –S –r httpd_cache –w httpd_cache –x
httpd_cache /var/cache/httpd

# setfdomex -A -r httpd_cache -w httpd_cache -x
httpd_cache /usr/sbin/httpd
```

You have now secured the HTTP subsystem. Restart the `httpd` process and double-check that it is running and has the proper security settings.

You will receive a `/dev/null` error but this is perfectly normal.

```
# /etc/rc.d/init.d/httpd restart
# ps x | grep httpd
6124 ?        S      0:00 httpd

# secps 6124
6124 root root
 Effective File Domains
   (r-x)httpd
   (rwx)httpd_logs
 Effective Net Domains
   (net)net_httpd
 Maximum File Domains
   (r-x)httpd
   (rwx)httpd_logs
 Maximum Net Domains
   (net)net_httpd
 Security Flags
ASG_AWARE,ASG_SEC_ACC_FS,
ASG_SEC_ACC_NET,ASG_RES_RBT,ASG_RES_MNT
```

A final note on maintaining the integrity of the `httpd` compartment: the `httpd` and `httpd_logs` access domains should not be used outside of the HTTP subsystem, and the `net_httpd` network domain should not be placed on any network data that is not intended for the HTTP subsystem.

A.2.1.2 HTTP Netrules

Non-LX Aware data entering a system can pose risks to security. In order to keep network traffic from coming in without a domain or an `ASG_AWARE` flag, netrules must be set on the incoming ports.

Most HTTP traffic runs through port 80. By setting the following netrule, everything that comes through or leaves on port 80 will be tagged with the `net_httpd` domain, making it LX Aware.

Add netrules by using an editor (such as `vi`) to open `/etc/argus/netrules`. At the bottom of the file, add the following:

```
#httpd
proto:tcp sport:80 domain:net_httpd
proto:tcp dport:80 domain:net_httpd
```

Once these rules have been added to the `netrules` file, load the new network data into the kernel.

```
# setnetrule
```

Verify the new rules.

```
# getnetrule
```

A.2.2 Securing the email Subsystem

The email subsystem is made up of a mail transfer agent (`sendmail`) and mail clients (`pine`, `mail`, etc.). These components of the email subsystem perform different tasks, but from the perspective of security they share one important item: they all need access to the users' mailboxes. This common need for access forms the basis of the PitBull LX email compartment.

All of the executables that make up the email subsystem need to be protected with the `sys(-w-)` domain. The processes also need to have their network access protected. This requires the creation of different network rules for each of them. Since the users need to have access to their mail, they also need access to the mailspool directory.

You must modify the `users` file to have `mailspool(rwx)` on every LX Aware user.

A.2.2.1 Email Security Policy

- Only the superuser, in LX Unaware mode, should be able to change vital `sendmail` files
- The `/var/spool/mail` and `/var/spool/mqueue` directories should be protected so that only `sendmail` is able to access them.

A.2.2.2 Securing Sendmail

The process of securing `sendmail` is similar to the process used for the HTTP subsystem. The first step is to create the execution environment for the `sendmail` executable. The `sendmail` executable will have the `mailqueue` and `mailspool` execution domains and the `net_sendmail` network domain.

First, set the execution domains on the binaries, and then set the security flags you want the process image to have. Use `ASG_SEC_ACC_FS` and `ASG_SEC_ACC_NET` to restrict the sendmail daemon to accessing only files and network data with the `ASG_AWARE` flag. Also, disallow mount, unmount, and reboot options with the `ASG_RES_MNT` and `ASG_RES_RBT` flags. `ASG_RES_SYS` is used to limit the amount of access the process has to the system. Finally, set both the `ASG_MASK_EXEC` and `ASG_COMB_EXEC` security flags to cause the process to have the execution domains and flags equal to those set on the executable, regardless of the security settings of the process that executes `sendmail`.

```
# setfdomex -S -r mailqueue,mailspool -w mail-
queue,mailspool -x mailspool -n net_sendmail /usr/
sbin/sendmail

# setfsfex -S
ASG_SEC_ACC_FS,ASG_SEC_ACC_NET,ASG_RES_SYS,ASG_RES_M
NT,ASG_RES_RBT /usr/sbin/sendmail

# setfsf -S ASG_MASK_EXEC,ASG_COMB_EXEC /usr/sbin/
sendmail
```

Set up the mail spool directory so that only the `sendmail` process can access it. To do this, set the file access domains on the `/var/spool/mail` directory to `mail-spool(rwx)` with `setfdom`.

```
# setfdom -S -r mailspool -w mailspool -x mailspool
/var/spool/mail
```

To ensure that any directories created under `/var/spool/mail` inherits the settings placed on it, use `setfsf` to set the `ASG_INHERIT_DOM` flag on `/var/spool/mail`.

```
# setfsf -S ASG_INHERIT_DOM /var/spool/mail
```

`sendmail` queues all outgoing mail until it can be delivered under the directory `/var/spool/mqueue`. Regular users have no need to directly access this directory, so only `sendmail` should have access to it. The `mailqueue` access domain provides this ability. The procedure is the same as with the `/var/spool/mail` directory.

```
# setfdom -S -r mailqueue -w mailqueue /var/spool/
mqueue

# setfsf -S ASG_AWARE,ASG_INHERIT_DOM /var/spool/
mqueue
```

All mail sent by `sendmail` is logged in the file `/var/log/sendmail.st` . `sendmail` needs to write to this file, but other users and processes will not. Setting the access domain on this file to `mailqueue(-w-)` will serve to accomplish this.

```
# setfdom -S -w mailqueue /var/log/sendmail.st
```

Add netrules by using an editor (such as `vi`) to open `/etc/argus/netrules`. At the bottom of the file, add the following:

```
#sendmail
proto:tcp dport:25 domain:net_sendmail
proto:tcp sport:25 domain:net_sendmail
```

Once these rules have been added to the `netrules` file, load the new network data into the kernel.

```
#  setnetrule
```

Verify the new rules.

```
#  getnetrule
```

You have now secured the `sendmail` daemon. Restart the daemon and verify all security settings, as was done with Apache.

A.2.2.3 IMAP Security Policy

- IMAP requires access to users' mailboxes under the `/var/spool/mail` directory only.
- The only time the `imapd` daemon should be run is when spawned by `inetd`, so its access domains should be set to `sys(rwx)`.

A.2.2.4 Securing IMAP

IMAP provides a convenient way for your users to connect to your server to read, transfer, and delete mail from their mailboxes. You'll be using the `mail-spool(rwx)` domains because the files and directories under `/var/spool/mail` have already been protected with them.

```
# setfdom -S -r sys -w sys -x sys /usr/sbin/imapd

# setfdomex -S -r mailspool -w mailspool -x mailspool
-n net_imapd /usr/sbin/imapd

# setfsfex -S
ASG_SEC_ACC_FS,ASG_SEC_ACC_NET,ASG_RES_SYS,ASG_RES_M
NT,ASG_RES_RBT /usr/sbin/imapd

# setfsf -S ASG_AWARE,ASG_MASK_EXEC /usr/sbin/imapd
```

`imapd` has now been properly secured with PitBull LX.

A.2.2.5 IMAP Netrules

Secure port 143, which `imapd` uses, with the `net_imapd` domain.

Add netrules by using an editor (such as `vi`) to open `/etc/argus/netrules`. At the bottom of the file, add the following:

```
#imapd
proto:tcp dport:143 domain:net_imapd
proto:tcp sport:143 domain:net_imapd
```

Once these rules have been added to the `netrules` file, load the new network data into the kernel.

```
#  setnetrule
```

Verify the new rules.

```
#  getnetrule
```

A.2.2.6 Securing POP

The procedure for securing the POP daemon on your system is the same as for the IMAP daemon. Use `net_popd` instead of the `net_imapd` network domain.

```
# setfdom -S -r sys -w sys -x sys /usr/sbin/ipop3d

# setfdomex -S -r mailspool -w mailspool -x mailspool
-n net_popd /usr/sbin/ipop3d

# setfsfex -S
ASG_SEC_ACC_FS,ASG_SEC_ACC_NET,ASG_RES_SYS,ASG_RES_M
NT,ASG_RES_RBT /usr/sbin/ipop3d

# setfsf -S ASG_AWARE,ASG_MASK_EXEC /usr/sbin/ipop3d
```

The email subsystem is now secure and ready for use.

A.2.2.7 POP Netrules

Secure port 110, which `popd` uses, with the `net_popd` domain.

Add netrules by using an editor (such as `vi`) to open `/etc/argus/netrules`. At the bottom of the file, add the following:

```
#popd
dport:110 domain:net_popd
sport:110 domain:net_popd
```

Once these rules have been added to the `netrules` file, load the new network data into the kernel.

```
#  setnetrule
```

Verify the new rules.

```
# getnetrule
```

A.2.2.8 Setting up User Accounts

Review the LX documentation on how to set up the proper access domains on users logging into your system, so that they will have the required access domains on their login shells.

Appendix B: PitBull LX Command Summary

Here is a brief description of the PitBull LX commands. For details on a command, view the online man page.

Path	Command	See Page	Description
`/sbin/`	`chdommap`	46	Edit the file access domain mapping file or the network access domain mapping file
`/sbin/`	`chkdommap`	48	Verify a domain map by checking for correct formatting
`/bin/`	`getfdom`	23	View access domains on a file
`/bin/`	`getfdomex`	25	View execution domains on a file
`/bin/`	`getfsf`	24	View the security flags on a file
`/bin/`	`getfsfex`	25	View execution flags on a file
`/bin/`	`getlicense`	2	Retrieve license string from a running kernel
`/bin/`	`getnetrule`	41	View the currently loaded network rule set
`/bin/`	`getpdom`	31	View domains on a process
`/bin/`	`getpsf`	32	View the security flags on a process
`/sbin/`	`lxexec`	39	Execute a binary with specified LX attributes
`/bin/`	`mkinteg`	56	Backup filesystem security attributes
`/bin/`	`secls`	23	View attributes on a file
`/bin/`	`secps`	31	Get process security information
`/bin/`	`chkinteg`	56	Check attributes of filesystem objects
`/bin/`	`setfdom`	23	Set access domains on a file
`/bin/`	`setfdomex`	25	Set execution domains on a file
`/bin/`	`setfsf`	24	Set security flags on a file

`/bin/`	`setfsfex`	25	Set execution flags on a file
`/bin/`	`setlicense`	2	Set license string
`/bin/`	`setnetrule`	41	Load a network rule set into the kernel
`/sbin/`	`chup`	52	Create or modify user login configuration entry for file
`/sbin/`	`chkup`	53	Check format for the user login configuration file

This chart indicates which commands are used to set and view which security attributes.

Attribute	Set	View
Domain mapping files	`chdommap`	`chkdomap to check validation (no view)`
File access domains	`setfdom`	`getfdom, secls`
Execution domains	`setfdomex`	`getfdomex, secls`
File security flags	`setfsf`	`getfsf, secls`
Execution flags	`setfsfex`	`getfsfex, secls`
Process domains	`(see note)`	`getpdom, secps`
Process security flags	`(see note)`	`getpsf, secps`
Network rules	`setnetrule`	`getnetrule`
License	`setlicense`	`getlicense`
Filesystem Integrity	`mkinteg`	`chkinteg to check validation (no view)`
User Policy	`chup`	`chkup to check validation (no view)`

Note: Process domains may not be set on a running process, though the `lxexec` command may start a new process with process domains and security flags.