

**AppArmor**

**Advanced  
User's Guide**

<b>Introduction to AppArmor</b>	<b>4</b>
<i>Conventions Used in This User's Guide</i>	<b>4</b>
Manual Text	4
Examples, Notes and Warnings	5
Command	5
Filename	5
Computer Output	5
Prompts	6
<i>Understanding This User's Guide</i>	<b>6</b>
<i>Getting Started With AppArmor</i>	<b>6</b>
AppArmor Installation	7
<b>Why Immunize Programs?</b>	<b>8</b>
<i>How To Immunize With SubDomain</i>	<b>8</b>
<b>What Should You SubDomain?</b>	<b>9</b>
SubDomaining SetUID Programs	10
SubDomaining Cron Jobs	10
SubDomaining Web Applications	10
SubDomaining Network Agents	12
<b>How to Build SubDomain Profiles</b>	<b>16</b>
<i>Profile Components and Syntax</i>	<b>16</b>
Breaking Down the SubDomain Profile Into Its Parts	16
#include	18
Capability Entries (POSIX.1e)	19
<i>About Using the Command-line Interface</i>	<b>19</b>
<i>Checking the SubDomain Module Status</i>	<b>20</b>
<i>Building the SubDomain Profiles</i>	<b>22</b>
Add or Create a SubDomain Profile	22
Edit a SubDomain Profile	23
Delete a SubDomain Profile.	23
<i>Two Methods of Profiling</i>	<b>24</b>
Standalone Profiling	24
Systemic Profiling	25
<i>Summary of Profiling Tools</i>	<b>26</b>
Autodep	27
Complain or Learning Mode	28
Genprof	30
Logprof	35
Subdomain.vim	40
Unconfined	41
<i>Path Names and Regular Expression Matching</i>	<b>42</b>
<i>File Permission Access Modes</i>	<b>43</b>
Read Mode	43

- Write Mode 44
- Discrete Profile Execute Mode 44
- Unconstrained Execute Mode 44
- Inherit Execute Mode 44
- Link Mode 44

## **Managing Profiled Applications 46**

### *Reacting to Security Events 46*

- Receiving a Security Event Rejection 46
- Changing Application Security 47

### *Maintaining Your Security Profiles 47*

- Backing Up Your Security Profiles 47
- Changing Your Security Profiles 48
- Introducing New Software Into Your Environment 48

## **Profiling Your Web Applications Using ChangeHat Apache 49**

### *What is ChangeHat? 49*

### *Apache ChangeHat 49*

- Tools for Managing ChangeHat Aware Applications 50
- phpsysinfo Hat (Subprofile) 53
- Running Genprof While Using ChangeHat 54

### *Apache Configuration for mod\_change\_hat 55*

- vhosts Directives 55
- Location and Directory Directives 56

## **Immunix Support 58**

### *Updating AppArmor Online 58*

### *Using the Man Pages 59*

### *Troubleshooting Solutions 60*

### *Getting Online Support 61*

### *Using Mailing List Support 61*

## **Glossary 62**

## Chapter 1 Introduction to AppArmor

The Immunix AppArmor is designed to provide application security for both servers and workstations that is easy to use. Immunix secures applications by differentiating between “good” and “bad” behaviors, then permitting the good and preventing the bad. This works to defend your systems *without* resorting to attack signatures, and thus can prevent attacks even if they are exploiting *previously unknown* vulnerabilities.

**AppArmor** is a *mandatory access control* system designed for application firewalling, **AppArmor** lets you specify *per program* which files the program may read, write, and execute.

AppArmor is comprised of:

- A library of AppArmor *profiles* for common Linux applications describing what files the program needs to access.
- A library of AppArmor *profile foundation classes* (profile building blocks) needed for common application activities such as DNS lookup and user authentication.
- A tool suite for developing and enhancing AppArmor profiles, so that you can create new profiles for your own local and custom applications.
- Specially modified application(s) that are AppArmor-*enabled* to provide enhanced security in the form of Immunix’s unique *sub-process confinement*. In the current release, the AppArmor-enabled application is Apache.
- The AppArmor loadable kernel module and associated control scripts to enforce AppArmor policies on your SuSE LINUX Enterprise Server 9 systems.

### Conventions Used in This User’s Guide

#### **Manual Text**

When using GUIs, field names, menu and screen titles, and field values are shown as **File**.

Key names are listed as they appear on your keyboard, as in **Enter** and **Esc** (for Escape).

### ***Examples, Notes and Warnings***

Examples are focused with **Example:** when appropriate. Notes and pertinent information are shown with a **Note:** or **Warning!** flag, as in:

#### **Note:**

Notes highlight information that might help you to better understand previous paragraphs. Warnings provide important information that might seriously affect the integrity of the product or your data.

### ***Command***

Linux commands (and other operating system commands, when used) are represented *this way*. This style should indicate to you that you can type the word or phrase on the command line and press [Enter] to invoke a command.

**Example:** To use `ls` to view the contents in the current directory, you would type `ls` in a terminal window.

### ***Filename***

Filenames, directory names, paths, and RPM package names are represented *this way*. This style should indicate that a particular file or directory exists by that name on your Linux system.

### ***Computer Output***

When you see text *in this style*, it indicates text displayed by the computer on the command line. You will see responses to commands you typed in, error messages, and interactive prompts for your input during scripts or programs shown *this way*.

Use the `ls` command to display the contents of a directory:

```
$ ls
Desktop  about.html  logs
Mail     backupfiles mail
```

## Prompts

A prompt, which is a computer's way of signifying that it is ready for you to input something, will be shown in this style.

**Example:** `puter login:`

## Understanding This User's Guide

For installation instructions, please refer to the separate AppArmor *Installation & Getting Started Guide*.

**Chapter 2: Why Immunize Programs?** describes operation of the Immunix Application Armor.

**Chapter 3: What Should You Immunize?** describes the types and details about a program that should have AppArmor profiles created for them.

**Chapter 4: How to Build AppArmor Profiles** describes how to use the Immunix tools to *immunize* your own programs and 3<sup>rd</sup> party programs that you may have installed on your SuSe Linux. It also helps you to add, edit or delete profiles that have been created for your applications.

**Chapter 5: Managing Profiled Applications** describes how to perform AppArmor profile maintenance, which involves tracking common issues and concerns.

**Chapter 6: Profiling Your Web Applications Using ChangeHat Apache** enables you to create subprofiles that can be applied to several applications, saving you a great deal of time and redundant efforts.

**Chapter 7: Immunix Support** indicates support options for this product.

## Getting Started With AppArmor

The Immunix AppArmor provides you with technologies to protect your applications from their own vulnerabilities by creating AppArmor profiles for SLES 9 server applications.

## ***AppArmor Installation***

After Installing SLES 9, including the required packages, you are ready to install AppArmor. Refer to the *AppArmor Installation Guide* for detailed installation steps.

Once AppArmor is installed, you are ready to use it to manage AppArmor profiles. You can do this by using the command line (refer to “Building AppArmor Profiles” on page 21).

## Chapter 2 Why Immunize Programs?

AppArmor provides Immunization technologies that protect SLES 9 applications from the inherent vulnerabilities they possess. After installing AppArmor, setting up AppArmor profiles and rebooting the PC, your system becomes **immunized** because it begins to enforce the AppArmor security policies. Protecting programs with AppArmor is referred to as **immunizing**. In the following sections, we explain why and how to immunize your programs.

AppArmor sets you up with a collection of default application profiles to protect standard Linux services. To protect other applications, use the Immunix tools to create profiles for the applications that you want protected. This chapter introduces you to the philosophy of Immunizing programs. Proceed to *Chapter 4: How to Build AppArmor Profiles* if you're ready to build and manage AppArmor profiles.

### How To Immunize With AppArmor

AppArmor provides **streamlined access control** for network services by specifying which files each program is allowed to read, write, and execute. This ensures that each program does what it is supposed to do, **and nothing else**.

AppArmor is **host intrusion prevention** or a mandatory access control scheme that is optimized for servers. Previously, access control schemes were centered around *users* because they were built for large time-share systems. Alternatively, modern network servers largely do not permit users to log in, and instead provide a variety of network services for users such as web, mail, file, print, etc. AppArmor controls the access given to network services and other programs to prevent weaknesses from being exploited.



## Chapter 3 What Should You Immunize?

AppArmor quarantines programs to protect the rest of the system from being damaged by a compromised process. Thus programs that need profiling are those that *mediate privilege*. For example, sometimes a program has access to resources that the person using the program does not have, which is true for the following types of programs:

- **SetUID Programs:** Programs that are `setuid` or `setgid` run as the user or group that owns the program file, rather than the usual case of running as the user and group of the person invoking the program. For instructions on using AppArmor for this type of program, refer to “Immunizing SetUID Programs” on page 10.
- **cron jobs:** Programs that will be run periodically by cron. Such programs read input from a variety of sources, and may run with special privileges, sometimes with as much as root privilege, e.g. cron runs `/usr/bin/updatedb` daily to keep the *slocate* database up to date, with sufficient privilege to read the name of every file in the system. For instructions on using AppArmor for this type of program, refer to “Immunizing Cron Jobs” on page 10.
- **Web Applications:** Programs that can be invoked through a web browser, including CGI PERL scripts, PHP pages, and more complex web applications. For instructions on using AppArmor for this type of program, refer to “Immunizing Web Applications” on page 10.
- **Network Agents:** Programs (servers and clients) that have open network ports. User clients such as mail clients and web browsers, surprisingly, mediate privilege. These programs run with the privilege to write to the user’s home directories, and process input from potentially hostile remote sources, such as hostile web sites and e-mailed malicious code. For instructions on using AppArmor for this type of program, refer to “Immunizing Network Agents” on page 12.

Conversely, unprivileged programs do *not* need to be profiled. For instance, a shell script might invoke the `cp` program to copy a file. Since `cp` does not have its own profile, it *inherits* the profile of the parent shell script, and thus can copy any files that the parent shell script’s profile can read and write.

### ***Immunizing SetUID Programs***

To find `setuid` programs, you can inspect your file system. For instance, this command will find files that are `setuid root`:

```
find / -user root -perm -4000 -print
```

### ***Immunizing Cron Jobs***

To find programs that will be run by `cron`, you need to inspect your local `cron` configuration. Unfortunately, `cron` configuration is rather complex, and so there are numerous files to inspect. Periodic `cron` jobs are run from these files:

```
/etc/crontab  
/etc/cron.d/*  
/etc/cron.daily/*  
/etc/cron.hourly/*  
/etc/cron.monthly/*  
/etc/cron.weekly/*
```

For `root`'s `cron` jobs, you can edit the tasks with “`crontab -e`”, and list `root`'s `cron` tasks with “`crontab -l`”. You must be `root` for these to work.

### ***Immunizing Web Applications***

To find web applications, you should investigate your web server configuration. The Apache web server is highly configurable, and web applications can be stored in many directories, depending on your local configuration. SuSE Linux, by default, stores web applications in `/srv/www/cgi-bin/`. To the maximum extent possible, each web application should have an AppArmor profile.

Because CGI programs are to be *executed* by the Apache web server, the profile for Apache itself `usr.sbin.httpd2-prefork` (for Apache 2 on SuSE Linux) must be modified to add execute permissions to each of these programs. For instance, adding the line “`/srv/www/cgi-bin/my_hit_counter.pl rpx,`” would grant Apache permission to execute the PERL script `my_hit_counter.pl` and require that there be a dedicated profile for

`my_hit_counter.pl`. If `my_hit_counter.pl` does not have a dedicated profile associated with it, then the rule should say `"/srv/www/cgi-bin/my_hit_counter.pl rix,"` to cause `my_hit_counter.pl` to inherit the `usr.sbin.httpd2-prefork` profile.

Some users may find it inconvenient to specify execute permission for every CGI script that Apache may invoke. Instead, the administrator can grant controlled access to collections of CGI scripts. For instance, adding the line `"/srv/www/cgi-bin/*.{pl,py,pyc} rix,"` will allow Apache to execute all files in `/srv/www/cgi-bin/` ending in `.pl` (PERL scripts) or `.py` or `.pyc` (Python scripts). As above, the `"ix"` part of the rule causes the Python scripts to inherit the Apache profile, which is appropriate if you do not want to write individual profiles for each Python script.

**Note:** If you want the Sub-process confinement module (`mod_change_hat`) functionality when web applications handle Apache *modules*, (`mod_perl` and `mod_php`), insert the AppArmor sub-process confinement module into the Apache web server.

The AppArmor installer installs this modified Apache web server along with `change_hat`. AppArmor for Apache is provided by the `mod_change_hat` Apache module. To take advantage of the sub-process confinement, refer to "Apache ChangeHat" on page 57.

Profiling web applications that use `mod_perl` and `mod_php` require slightly different handling. In this case, the "program" is a script interpreted directly by the module within the Apache process, so no `exec` happens. Instead, the Immunix version of Apache calls `change_hat()` naming a subprofile (a "hat") corresponding to the name of the URI being requested.<sup>1</sup> For `mod_perl` and `mod_php` scripts, this will be the name of the PERL script or the PHP page requested. So for example, adding this subprofile to `foo` will allow the

1. The name presented for the script to be executed may not be the URI, depending on how Apache has been configured for where to look for module scripts. If you have configured your Apache to place scripts in a different place, then the different names will show up in Syslog when AppArmor complains about access violations. See "Managing Profiled Applications" on page 45.

localtime.php page to execute and access the local system time:

```
/usr/sbin/httpd2-prefork^/cgi-bin/localtime.php {  
    /etc/localtime                                r,  
    /srv/www/cgi-bin/localtime.php                r,  
    /usr/lib/locale/**                            r,  
}
```

If no subprofile has been defined, then the Immunix version of Apache applies the `DEFAULT_URI` hat. This subprofile is basically sufficient to display an HTML web page. The `DEFAULT_URI` hat that Immunix provides by default is:

```
/usr/sbin/suexec2 ixr,  
/var/log/apache2/** rwl,  
/home/*/public_html/**          r,  
/srv/www/htdocs/**              r,  
/srv/www/icons/*.{gif,jpg,png}  r,  
/usr/share/apache2/**            r,
```

If you want a single AppArmor profile for *all* web pages and CGI scripts served by Apache, then editing the `DEFAULT_URI` subprofile is a good approach.

### **Immunizing Network Agents**

To find network server daemons that should be profiled, you should inspect the open ports on your machine, consider the programs that are answering on those ports, and provide profiles for as many of those programs as possible. If you provide profiles for *all* programs with open network ports, then for all possible network threats, the attacker cannot get to the file system on your machine without passing through an AppArmor profile policy.

Scanning your server for open network ports can be done manually from outside the machine using a scanner such as `nmap`, or from inside the machine using `netstat`, and then inspecting the machine to determine which programs are answering on the discovered open ports.

## Using Unconfined To Inspect Open Ports

An automated method for finding network server daemons that should be profiled is to use the Immunix-provided tool **unconfined**. Using the command “netstat -nlp,” the **unconfined** tool inspects your open ports from inside your computer, detects the programs associated with those ports, inspects the set of AppArmor profiles that you have loaded. Unconfined then reports these programs along with the AppArmor profile associated with each program, or reports “none” if the program is not confined.

**Note:** If you create a new profile, you must restart the program that has been profiled in order for unconfined to detect and report the new profiled state.

Below you will find sample unconfined output:

- The first portion is a number. This number is the PID of the listening program.
- The second portion is a string, which represents is the absolute path of the listening program
- The final portion indicates the profile confining the program, if any.

```
2325 /sbin/portmap not confined
3702 /usr/sbin/sshd confined by '/usr/sbin/sshd
(enforce)'
4040 /usr/sbin/ntpd confined by '/usr/sbin/ntpd
(enforce)'
4373 /usr/lib/postfix/master confined by
'/usr/lib/postfix/master (enforce)'
4505 /usr/sbin/httpd2-prefork confined by
'/usr/sbin/httpd2-prefork (enforce)'
5274 /sbin/dhcpd not confined
5592 /usr/bin/ssh not confined
7146 /usr/sbin/cupsd confined by '/usr/sbin/cupsd
(complain)'
```

**Notes:**

Requires root privilege, and should not itself be run from within a AppArmor profile.

**unconfined** does not distinguish between one network interface and another, and so it will report *all* unconfined processes, even those

that may be listening to an internal LAN interface.

Finding user network client applications is dependent on your user's preferences. The `unconfined` tool will detect and report network ports opened by client applications, but *only* those client applications that are running at the time the `unconfined` analysis is performed. This is a problem because network services tend to be running all the time, while network client applications tend to only be running when the user is interested in them.

Applying AppArmor profiles to user network client applications is also dependent on user's preferences, and Immunix is intended for servers rather than workstations, and so we leave profiling of user network client applications as an exercise for the user.

To aggressively confine desktop applications, the `unconfined` command supports a `paranoid` option, which will report all processes running and the corresponding AppArmor profiles that may or may not be associated with each process. The `unconfined` user can then decide whether each of these programs needs a AppArmor profile.

Additional profiles may be traded with other Immunix users and with the Immunix security development team on the Immunix user's mailing list at the following URL:

<http://mail.wirex.com/mailman/listinfo/immunix-users>

## Chapter 4 How to Build AppArmor Profiles

This chapter explains how to build and manage AppArmor profiles. You are ready to build AppArmor profiles once you select the programs to be profiled. For help with this, refer to “What Should You Immunize?” on page 9.

### Profile Components and Syntax

This section details the syntax or makeup of AppArmor profiles. An example illustrating this syntax is presented on “Breaking Down the AppArmor Profile Into Its Parts” on page 15.

#### ***Breaking Down the AppArmor Profile Into Its Parts***

AppArmor profile components are called AppArmor *rules*. Currently there are two main types of AppArmor rules, *path* entries and *capability* entries. **Path entries** specify what the process can access in the filesystem and **Capability entries** provide a more fine-grained control over what a confined process is allowed to do through other system calls that require privileges. Includes are a type of meta-rule or directives that pull in path and capability entries from other files.

The easiest way of explaining what a profile is comprised of and how to create one is to show the details of a sample profile. Consider, for example, the following profile for the program, `/sbin/klogd`:

```
# profile to confine klogd
/sbin/klogd
{
#include <abstractions/base>

    capability sys_admin,

    /boot/* r,
    /proc/kmsg r,
    /sbin/klogd r,
    /var/run/klogd.pid lw,
}
```

**The first line:** The first line is a comment.

**The second line:** The second line indicates the absolute path of the program to be confined. In this example, whenever a program named `/sbin/klogd` executes, it will be confined by this profile.

**Subsequent lines within the brackets {}:** The rest of the lines take one of several forms:

- **#include** directives that pull in components of AppArmor profiles to simplify profiles.
- **Capability Entries** statements that enable each of the 32 POSIX.1e capabilities.
- **Path Entries** in which the first part specifies the absolute path of a file (possibly including regular expression globbing), and the second part indicates permissible access modes (**r**: read, **w**: write, and **x**: execute).

**Spaces or Tabs:** A white space of any kind (spaces or tabs) can precede path names or separate the path name from the access modes. White space between the access mode and the trailing comma is optional.

When a profile is created for a program, the program can *only* access the files, modes, and POSIX capabilities specified in the profile. These



restrictions are in *addition* to the native Linux access controls.

**Example:** To gain the capability `CAP_CHOWN`, the program must have **both** access to `CAP_CHOWN` under conventional Linux access controls (typically, be a root-owned process) **and** have “capability chown” in its profile. Similarly, to be able to write to the file `/foo/bar` the program must have both the correct user-ID and mode bits set in the file attributes (see the `chmod` and `chown` man pages) and have “`/foo/bar w`” in its profile.

Attempts to violate AppArmor rules are recorded in `syslog`. In many cases, AppArmor rules will prevent an attack from working because necessary files are not accessible, and in all cases AppArmor confinement bounds the damage that the attacker can do to the set of files permitted by AppArmor.

### **#include**

`#includes` are directives that pull in components of other AppArmor profiles to simplify profiles. Include files procure access permissions for programs. By using an include, you can give the program access to directory paths or files that are also required by other programs. Using includes can reduce the size of a profile.

By default, the `#include` statement appends the beginning of the path-name to `/etc/subdomain.d/` which is where it expects to find the include file. Unlike other profile statements (but similar to C programs), `#include` lines do not end with a comma.

Immunix provides two classes of `#includes`, **Abstractions** and **Program Chunks** to assist you in profiling your applications.

### **Abstractions**

Abstractions are `#includes` that are grouped by common application tasks. These tasks include access to authentication mechanisms, access to nameservice routines, common graphics requirements, and system accounting, among others. Files listed in these abstractions are specific to the named task; programs that require one of these files will usually require some of the other files listed in the abstraction file (depending upon local configuration as well as the specific require-

ments of the program). Abstractions can be found in the following directory:

```
/etc/subdomain.d/abstractions/
```

### Program Chunks

The program chunks are access controls for specific programs that a system administrator may wish to control based on local site policy. Each chunk is used by a single program; these are provided to ease local-site modifications to policy and updates to policy provided by Immunix. Administrators can modify policy in these files to suit their own needs, leaving the program profiles unmodified, which will simplify the task of merging policy updates from Immunix into enforced policy at each site.

The access restrictions in the program chunks are typically very liberal, and are designed to allow your users access to their files in the least intrusive way possible, while still allowing system resources to be protected. An exception to this rule is the postfix\* series of program chunks; these profiles are used to help abstract the location of the postfix binaries. You will probably not want to reduce the permissions in the postfix\* series. Program Chunks can be found in the following directory:

```
/etc/subdomain.d/program-chunks/
```

### Capability Entries (POSIX.1e)

Capabilities statements are simply the word “capability” followed by the name of the POSIX.1e capability as defined in the capabilities(7) man page, which you can get to by typing `man capabilities` in a terminal window.

## About Using the Command-line Interface

Using SLES 9 operating systems, you have the ability to build and manage AppArmor profiles using the command-line interface. Some operating systems have a GUI interface available. If you require this option, please contact your sales representative for more information.

The command-line interface requires knowledge of Linux commands

and using terminal windows. To use the **Command-line Interface** for building and managing AppArmor profiles, refer to “Building AppArmor Profiles” on page 21.

The Command-line Interface offers access to a few tools that are not available using the other AppArmor managing methods. These tools are:

- **complain (or learning mode):** Sets profiles into complain mode. Set it back to enforce mode when you want the system to begin enforcing the rules of the profiles not just logging information. For more information on this tool, refer to “Complain or Learning Mode” on page 27
- **enforce:** Sets profiles back to enforce mode and the system begins enforcing the rules of the profiles not just logging information. For more information on this tool, refer to “Enforce Mode” on page 28
- **unconfined:** Performs a server audit to find processes that are running and listening for network connections and reports whether they are profiled or not.
- **autodep:** Generates a profile skeleton for a program and loads it into the SubDomain module in complain mode.

## Checking the SubDomain Module Status

The SubDomain module can be in one of three states:

**Unloaded:** The SubDomain module is not loaded into the kernel.

**Running:** The SubDomain module is loaded into the kernel, and is enforcing AppArmor program policies.

**Stopped:** The SubDomain module is loaded into the kernel, but there are *no* policies being enforced.

You can detect which of the three states that SubDomain is in by inspecting `/subdomain/profiles`. If **`cat /subdomain/profiles`** reports a list of profiles, then SubDomain is running. If it is empty and returns nothing then SubDomain is stopped. If the file does not exist, then SubDomain is unloaded.

The SubDomain module can be loaded and unloaded with the stan-

standard Linux module commands such as `modprobe`, `insmod`, `lsmod`, and `rmmmod`, but this approach is **not** recommended. Rather, it is recommended that you manage SubDomain through the script `/etc/init.d/subdomain` which can perform the following operations:

### **`/etc/init.d/subdomain start`**

Has different behaviors depending on the SubDomain module state. If it was unloaded, then `start` loads the module and starts it, putting it in the running state. If it was stopped, then `start` causes the module to re-scan the AppArmor profiles usually found in `/etc/subdomain.d` and puts the module in the running state. If the module was already running then `start` reports a warning and takes no action.

### **`/etc/init.d/subdomain stop`**

Stops SubDomain module (if it was running) by removing all profiles from kernel memory, effectively disabling all access controls, putting the module into the stopped state. If SubDomain was either unloaded or already stopped, then `stop` is a silent no-op.

### **`/etc/init.d/subdomain restart`**

Causes SubDomain to rescan the profiles usually found in `/etc/subdomain.d` without unconfining running processes, adding new profiles, and removing any profiles that had been deleted from `/etc/subdomain.d`.

### **`/etc/init.d/subdomain kill`**

Unconditionally removes the SubDomain module from the kernel. This is **unsafe**, because unloading modules from the Linux kernel is unsafe. This command is provided only for debugging and emergencies, when the module might have to be removed.

**Note:** AppArmor is a *powerful* access control system, and it is possible to lock yourself out of your own machine to the point where you have to boot the machine from rescue media (such as disc 1 of SLES 9) to regain control.

To prevent such a problem, always ensure that you have a running, unconfined, `root` login on the machine being configured when you restart the SubDomain module. If you damage your system to the point

where logins are no longer possible (for example, by breaking the profile associated with the SSH daemon) you can repair the damage using your running `root` prompt, and restarting SubDomain.

## Building AppArmor Profiles

AppArmor profile definitions are stored in the directory `/etc/subdomain.d/` as plain text files.

**Warning!** All files in the `/etc/subdomain.d/` directory are interpreted as profiles. Renaming files in that directory is not an effective way of preventing profiles from being loaded. You must remove profiles from this directory to manage them effectively.

You can use a text editor, such as `vim`, to access and make changes to these profiles. The following options contain detailed steps for building profiles:

- **Add or Create AppArmor Profiles:** Refer to “Add or Create a AppArmor Profile” on page 21
- **Edit an AppArmor Profile:** Refer to “Edit an AppArmor Profile” on page 22
- **Delete an AppArmor:** Refer to “Delete AppArmor Profile” on page 22

Use `vim` to view and edit your profile by typing `vim` at a terminal window. To enable the syntax coloring, when you edit an AppArmor profile in `vim`, use the command `“:syntax on”` and then `“:set syntax=subdomain”`. For more information on `vim` and syntax coloring, refer to “Subdomain.vim” on page 39.

**Note:** After making changes to a AppArmor file, use the `/etc/init.d/subdomain restart` command, described in the previous section. This command causes the SubDomain module to re-read the AppArmor files. For a detailed description of the syntax of these files, refer to “How to Build AppArmor Profiles” on page 15.

### ***Add or Create a AppArmor Profile***

To add or create an AppArmor profile for an application, you can use a systemic or standalone profiling method, depending on your needs.

Both methods are explained in detail here:

- **Standalone Profiling:** This method is suitable for profiling small applications that have a finite run time, such as user client applications like mail clients. “Standalone Profiling” on page 23.
- **Systemic Profiling:** This method is suitable for profiling large numbers of programs all at once, and for profiling applications that may run for days, weeks, or continuously across reboots, such as network server applications like web servers and mail servers. “Systemic Profiling” on page 24.

### ***Edit an AppArmor Profile***

The following steps tell you what you need to do in order to edit an AppArmor profile. To better understand what makes up a profile, refer to “Profile Components and Syntax” on page 15.

1. If you are not currently signed in as `root`, type `su` in a terminal window.
2. Enter the `root` password, when prompted.
3. To go to the AppArmor directory, type `cd /etc/subdomain.d/`
4. Type `ls` to view all the AppArmor profiles currently installed.
5. Open the profile you want to edit in a text editor, such as `vim`.
6. Make the necessary changes, then save the profile.
7. Restart AppArmor by typing `/etc/init.d/subdomain restart` in a terminal window.

### ***Delete AppArmor Profile***

The following steps tell you what you need to do in order to delete a AppArmor profile.

1. If you are not currently signed in as `root`, type `su` in a terminal window.
2. Enter the `root` password, when prompted. `domain.d/`.
3. Type `ls` to view all the AppArmor profiles that are currently installed.
4. Delete the profile exiting profile by typing `rm profilename`.

5. Restart the SubDomain module by typing `/etc/init.d/subdomain restart` in a terminal window.

## Two Methods of Profiling

Given the syntax for AppArmor profiles in “Profile Components and Syntax” on page 15, one *could* create profiles without using the tools. However, the effort involved would be substantial. To avoid such a hassle, use the AppArmor tools to automate the creation and refinement of profiles.

There are two ways to approach creating AppArmor profiles, along with tools to support both methods.

- A **standalone method** (for more information, refer to “Standalone Profiling” on page 23), suitable for profiling small applications that have a finite run time, such as user client applications like mail clients.
- A **systemic method** (for more information, refer to “Systemic Profiling” on page 24), suitable for profiling large numbers of programs all at once, and for profiling applications that may run for days, weeks, or continuously across reboots, such as network server applications like web servers and mail servers.

Automated profile development becomes more manageable with the Immunix tools:

### 1. Decide which profiling method suits your needs.

- **Perform a static analysis.** Run either `genprof` or `autodep`, depending on the profiling method you have chosen.
- **Enable dynamic learning.** Activate *learning* mode for all profiled programs.

## Standalone Profiling

Standalone profile generation and improvement is managed by a program called `genprof`. This method is easy, because `genprof` takes care of everything, but limited, because it requires `genprof` to run for the entire duration of the test run of your program, i.e. you cannot reboot the machine while you are still developing your profile.

To use `genprof` and the standalone for profiling, refer to “Genprof” on page 29.

## Systemic Profiling

This method is called Systemic Profiling because it updates *all* of the profiles on the system at once, rather than focusing on the one or few being targeted by `genprof` or Standalone Profiling.

With Systemic Profiling, building and improving profiles are somewhat less automated, but more flexible. This method is suitable for profiling long-running applications whose behavior continues after rebooting, or a large numbers of programs to be profiled all at once.

Building an AppArmor profile for a group of applications is as follows:

1. **Create profiles for the individual programs that make up your application.** Even though this approach is systemic, AppArmor still only monitors those programs with profiles, and their children. Thus, to get AppArmor to consider a program, you must at least have Autodep create an *approximate* profile for it. To create this approximate profile, refer to “Autodep” on page 25.

2. **Put relevant profiles into *learning/complain* mode.** Activate *learning/complain* mode for all profiled programs by typing the following:

```
complain /etc/subdomain.d/*
```

When in learning mode, access requests are not blocked even if the profile dictates that they should be. This enables you to run through several tests (as shown in Step 3) and learn the access needs of the program so it runs properly. With this information, you can decide how secure to make the profile.

Refer to “Complain or Learning Mode” on page 27 for more detailed instructions on how to use learning/complain mode.

3. **Exercise Your Application.** Run your application, and exercise its functionality. How *much* to exercise the program is up to you, but you will need the program to access each file representing its access needs. Because the execution is not being supervised by `genprof`, this step can go on for days or weeks, and can span complete system reboots.



4. In Systemic profiling, you run `logprof` directly instead of letting `genprof` run it for us as in the Standalone Profiling. The general form of `logprof` is:

```
logprof [ -d /path/to/profiles ] [ -f /path/to/logfile ]
```

Refer to “Logprof” on page 34 for more information on using Logprof.

5. **Repeat Steps 3-4.** Iterate Step 3 and Step 4 to generate optimum profiles. An iterative approach captures smaller data sets that can be trained and reloaded into the policy engine. Subsequent iterations will generate fewer messages and run faster.
6. **Edit the Profiles.** You may wish to review the profiles that have been generated. You can open and edit the profiles in `/etc/subdomain.d/` using `vim`. For help using `vim` to its fullest capacity, refer to “Subdomain.vim” on page 39.
7. **Return to “enforce” mode.** This is when the system goes back to enforcing the rules of the profiles not just logging information. This can be done manually, by removing the “`flags=(complain)`” text from the profiles, or automatically, using the `enforce` command, which works identically to the `complain` command, but edits the profiles to be in enforce mode.

To assure that *all* profiles are taken out of complain mode and put into enforce mode, type:

```
enforce /etc/subdomain.d/*
```

8. **Re-scan all profiles.** To have SubDomain re-scan all of the profiles and change the enforcement mode in the kernel, type:

```
/etc/init.d/subdomain restart
```

## Summary of Profiling Tools

All of the Immunix-provided AppArmor profiling utilities are provided by the `subdomain-utils` RPM package, and most are stored in `/usr/sbin`. Here is a brief summary of each tool:

### Autodep

When you run the Autodep program, it creates an approximate profile for the program or application you are autodepping. You can generate approximate profiles for binary executables and interpreted script pro-

grams. The resulting profile is called "approximate" because it does not necessarily contain all of the profile entries that the program needs to be properly confined by AppArmor. The minimum autodep approximate profile will at least have a base include directive, which contains basic profile entries needed by most programs. For certain types of programs, autodep will generate a more expanded profile. The profile is generated by recursively calling ldd(1) on the executables listed on the command line.

To generate an approximate profile, type `autodep program`. The "program" argument can be either the simple name of the program, and autodep will find it by searching the shell path variable, or it can be a fully qualified path. The program itself can be of any kind (ELF binary, shell script, PERL script, etc.) and autodep will still generate an approximate profile, to be improved through the dynamic profiling that follows. The resultant approximate profile is written to the `/etc/subdomain.d` directory using the AppArmor profile naming convention of naming the profile after the absolute path of the program, replacing the front slash (/) characters in the path with period (.) characters. The general form of autodep is to type the following in a terminal window when logged in as root:

```
autodep [ -d /path/to/profiles ] [program1  
program2...]
```

If you don't enter the program name or names, you will be prompted for them. `/path/to/profiles` overrides the default location of `/etc/subdomain.d`.

To begin profiling, you must create profiles for each main executable service that is part of your application (anything that may start up without being a child of another program that already has a profile). Finding all such programs is dependent on the application in question. Here are several strategies for finding such programs:

- **Directories:** If all of the programs you wish to profile are in a directory, and there are no other programs in that directory, then the simple command "`autodep /path/to/your/programs/*`" will create nominal profiles for all programs in that directory.
- **using ps:** You can run your application, and use the standard Linux

`ps` command to find all processes running. You then need to manually hunt down the location of these programs, and run the `autodep` program for each one. If the programs are in your path, then `autodep` will find them for you. If they are not in your path then the standard Linux command `locate` may be helpful in finding your programs.

### ***Complain or Learning Mode***

The *complain* or *learning* mode AppArmor tool detects violations of AppArmor profile rules, such as the profiled program accessing files not permitted by the profile. The violations are permitted, but also logged. To improve the profile, turn complain mode on, run the program through a suite of tests to generate log events that characterize the program's access needs, then post-process the log with the AppArmor tools to transform log events into improved profiles.

Manually activating *complain* mode (using the command line) adds a flag to the top of the profile so that `"/bin/foo {"` becomes `"/bin/foo flags=(complain) {"`. To use complain mode, open a terminal window and type one of the following lines as a root user.

- If the example program (`program1`) is in your path, type:  

```
complain [program1 program2 ...]
```
- If the program is not in your path, you should specify the entire path, as follows:  

```
complain /sbin/program1
```
- If the profiles are not in `/etc/subdomain.d`, type the following to override the default location:  

```
complain /path/to/profiles/ program1
```
- Alternately, you can specify the profile for `program1`, as follows:  

```
complain /etc/subdomain.d/sbin.program1
```

Each of the above commands will activate complain mode for the profiles/programs listed. The command can either list programs or profiles. If the program name is not fully qualified, then `complain`

searches \$PATH for the program. So for instance “`complain /usr/sbin/*`” will find profiles associated with all of the programs in `/usr/sbin` and put them into complain mode, and “`complain /etc/subdomain.d/*`” will put all of the profiles in `/etc/subdomain.d` into complain mode.

### **Enforce Mode**

The *enforce* mode AppArmor tool detects violations of AppArmor profile rules, such as the profiled program accessing files not permitted by the profile. The violations are logged and NOT permitted. Turn complain mode on when you want the AppArmor profiles to control the access of the program that is profiled. the default mode is for enforce mode to be turned on. *enforce* toggles with *complain* mode.

Manually activating *enforce* mode (using the command line) adds a flag to the top of the profile so that “`/bin/foo {`” becomes “`/bin/foo flags=(enforce) {`”. To use complain mode, open a terminal window and type one of the following lines as a root user.

- If the example program (`program1`) is in your path, type:  

```
enforce [program1 program2 ...]
```
- If the program is not in your path, you should specify the entire path, as follows:  

```
enforce /sbin/program1
```
- If the profiles are not in `/etc/subdomain.d`, type the following to override the default location:  

```
enforce /path/to/profiles/ program1
```
- Alternately, you can specify the profile for `program1`, as follows:  

```
enforce /etc/subdomain.d/sbin.program1
```

Each of the above commands will activate enforce mode for the profiles/programs listed.

If you don't enter the program or profile name(s), you will be prompted to enter one. `/path/to/profiles` overrides the default location of `/etc/subdomain.d`.

The argument can be either a list of programs or a list of profiles. If the program name does not include its entire path, then `enforce` searches `$PATH` for the program. For instance, “`enforce /usr/sbin/*`” will find profiles associated with all of the programs in `/usr/sbin` and put them into enforce mode, and “`enforce /etc/subdomain.d/*`” will put all of the profiles in `/etc/subdomain.d` into enforce mode.

### Genprof

Genprof (or Generate Profile) is AppArmor’s profile generating utility. It Autodeps the specified program, creating an *approximate* profile (if a profile doesn’t already exist for it), sets it to complain mode, reloads it into AppArmor, marks the syslog, and prompts the user to execute the program and exercise its functionality.

```
genprof [ -d /path/to/profiles ] program
```

If you were to create a profile for the the Apache web server program `httpd2-prefork`, you would type the following at a root shell prompt:

1. `/etc/init.d/apache2 stop`
2. Next, type `genprof httpd2-prefork`

Now Genprof will do the following:

- Resolve the full path of `httpd2-prefork` based on your shell’s path variables. You can also specify a full path. On SuSE LINUX Enterprise Server 9, the full path is:  
**`/usr/sbin/httpd2-prefork`**
- Check to see if there is an existing profile for `httpd2-prefork`. If there is one already, then `genprof` will update it. If not, then `genprof` creates one using the `autodep` program described in “Summary of Profiling Tools” on page 25. **Note:** there is a naming convention relating the full path of a program to its profile file name so that the various AppArmor profiling tools can consistently manipulate them. The convention is to replace `/` with `.` so that the profile for `/usr/sbin/httpd2-prefork` is stored in **`/etc/subdomain.d/usr.sbin.httpd2-prefork`**

- Put the profile for this program into "learning" or "complain" mode so that profile violations are logged, but are permitted to proceed. A log event looks like this:

```
Oct  9 15:40:31 SubDomain: PERMITTING r access to
/etc/apache2/httpd.conf      (httpd2-prefork(6068)
profile      /usr/sbin/httpd2-prefork      active
/usr/sbin/httpd2-prefork)
```

- Mark syslog with a beginning marker of log events to consider.
3. The tool prompts you to run the application to be profiled in another terminal window. Perform as many of the application functions as possible so learning mode can log the files and directories the program requires access to in order to function properly. For example, in a new terminal window type `/etc/init.d/apache2 start`
  4. You are given the following menu choices which can be used after you have executed the program functionality:
    - Select "S" from the `genprof` menu to run `logprof` against the system log from where it was marked when `genprof` was started, and reloads the profile.
    - Select "F" from the `genprof` menu to exit.
  5. If you selected "S" in the previous step and system events exist in the log, AppArmor will parse the learning mode log files. This will generate a series of questions which you must answer to guide `genprof` in generating the security profile.

**Note:** If requests to add hats appear, proceed to "Running Genprof While Using ChangeHat" on page 62.

The questions will fall into two categories:

- *A resource is requested by a profiled program that is not in the profile (see **Figure 1** below)*
- *Or a program is executed by the profiled program and the security domain transition has not been defined (see **Figure 2** below).*

Each of these cases results in a series of questions that you must answer to add the resource to the profile or to add the program into the profile. The following two figures show an example of each case. Subsequent steps describe your options in answering these questions.

**Figure 1:** The Learning Mode exception requires you to allow or deny access to a specific resource.

```
Reading log entries from /var/log/messages.  
Updating subdomain profiles in /etc/subdomain.d.  
  
Profile: /usr/sbin/xinetd  
Execute: /usr/sbin/vsftpd  
  
[(I)nherit] / (P)rofile / (U)nconstrained / (D)eny  
/ Abo(r)t / (F)inish
```

Dealing with execute accesses is complex. You must decide which of the three kinds of execute permissions you intend to grant the program

- **Inherit (ix):** The child inherits the parent's profile, i.e. runs with the same access controls as the parent. This mode is useful when a confined program needs to call another confined program without gaining the permissions of the target's profile, or losing the permissions of the current profile. This mode is often used when the child program is a "helper application", such as the `/usr/bin/mail` client using the `less` program as a pager, or the Mozilla web browser using the `acrobat` program to display PDF files.
- **profile (px):** The child runs using its own profile, which must be loaded into the kernel. If the profile is *not* present, then attempts to execute the child will fail with permission denied. This is most useful if the parent program is invoking a global service, such as DNS lookups or sending mail via your system's MTA.
- **unconfined (ux):** The child runs **completely unconfined** without any AppArmor profile being applied to the executed resource.

**Figure 2:** The Learning Mode exception requires you to define execute permissions for an entry.

```

Adding /bin/ps ix to profile.

Profile:  /usr/sbin/xinetd
Path:    /etc/hosts.allow
New Mode: r

[1 - /etc/hosts.allow]

[(A)llow] / (D)eny / (N)ew / (G)lob / Glob w/(E)xt
/ Abo(r)t / (F)inish

```

The above menu shows AppArmor **suggesting directory path entries** that have been accessed by the application you are profiling. It may also **require you to define execute permissions for entries**.

AppArmor provides one or more path names or includes. By clicking the option number, choose from one or more of the following options, then proceed to Step 6.

**Note:** All of these options are not always presented in the AppArmor menu.

- **#include:** An include is the section of a AppArmor profile that refers to an *include file*. Include files procure access permissions for programs. By using an include, you can give the program access to directory paths or files that are also required by other programs. Using includes can reduce the size of a profile. It's good practice to select #includes when suggested.
- **Globbed Version:** This is accessed by clicking the **Glob** button as described in the next step. For information on globbing syntax, refer to "Path Names and Regular Expression Matching" on page 41.
- **Actual Path Name:** This is the literal path that the program needs access to so that it can run properly.



6. Once you select the path name or `#include`, you can process it as an entry into the AppArmor profile by clicking **Allow** or **Deny**. If you are not satisfied with the directory path entry as it is displayed, you can also **Glob** or **Edit** it.

The following options are available to process the learning mode entries and to build the profile:

- **Press Enter:** Accepts the entry highlighted with []. Press the **Enter** key if you want to allow access to the selected directory path.
- **"Allow":** Select Allow if you want to grant the program access to the specified directory path entries. AppArmor suggests file permission access. For more information on this, refer to "File Permission Access Modes" on page 42
- **"Deny":** Select Deny to prevent the program from accessing the specified directory path entries. AppArmor will then move on to the next event.
- **"New":** Prompts you to enter your own rule for this event, allowing you to specify whatever form of regular expression you want. If the expression you enter does not actually satisfy the event that prompted the question in the first place, AppArmor will ask you for confirmation and let you re-enter the expression.
- **"Glob":** Select Glob once once to modify the directory path (by using wildcards) to include all files in the suggested entry directory. When you select Glob twice, access will be granted to all files and subdirectories beneath the one shown.

For more information on globbing syntax, refer to "Path Names and Regular Expression Matching" on page 41.

- **Glob w/"Ext":** When you select Glob w/Ext, the original directory path is modified while retaining the filename extension. With one click, `/etc/apache2/file.ext` becomes `/etc/apache2/*.ext`, adding the wildcard (asterisk) in place of the file name. This will allow the program to access all files in the suggested directory that end with the ".ext" extension. When you select it twice, access will be granted to all files (with the particular extension) and subdirectories beneath the one shown.
- **"Edit":** Select **Edit** to edit the highlighted line. The new line will appear at the bottom of the list.
- **Abort":** Aborts `logprof`, dumping all rule changes entered so far and leaving all profiles unmodified.
- **"Finish":** Closes `logprof`, saving all rule changes entered so far and modifying all profiles.

7. To view and edit your profile using `vim`, type `vim /etc/subdomain.d/profilename` in a terminal window. To enable the syntax coloring, when you edit a AppArmor profile in `vim`, use the command `“:syntax on”` and then `“:set syntax=subdomain”`. For more information on `vim` and syntax coloring, refer to “Subdomain.vim” on page 39.

### Logprof

Logprof is an interactive tool used to review the learning/complain mode output found in the AppArmor syslog entries, then generate new entries in AppArmor security profiles.

When you run `logprof`, it begins to scan the log files produced in learning/complain mode, and if there are new security events that are not covered by the existing profile set, the user is prompted with suggestions for modifying the profile. The learning/complain mode traces program behavior and enters it in syslog. Logprof uses this information to observe program behavior.

If a confined program forks and execs another program, `logprof` will see this and ask the user which execution mode should be used when launching the child process. The following execution modes are options for starting the child process: **ix**, **px**, or **ux**. If a separate profile exists for the child process, the default selection will be **px**. If one **doesn't** exist, the profile will default to **ix**. Child processes with separate profiles will be autodep'd and loaded into the SubDomain module, if it's running.

When `logprof` exits, profiles are updated with the changes. If AppArmor is running, the updated profiles are reloaded and if any processes that generated security events are still running in the null-complain-profile, those processes are set to run under their proper profiles.

To run `logprof`, you have to type `logprof` into a terminal window while logged in as `root`. The following options can also be used for `logprof`:

- **logprof -d** /path/to/profile/directory/

Use this option to specify the full path to the location of the profiles if the profiles are not located in the standard directory, `/etc/subdomain.d/`.

- **logprof -f** /path/to/logfile/  
Use this option to specify the full path to the location of the logfile if the logfile is not located in the default directory, /var/log/messages/.
- **logprof -m** "string marker in logfile"  
Use this option to mark the starting point for logprof to look in the system log. logprof will ignore all events in the system log before the specified mark is seen. If the mark contains spaces, it must be surrounded with quotes to work correctly. This option would look like this: `logprof -m "Jan 19 13:09:51"`

Logprof scans through the log, asking you how to handle each logged event. Each question presents a numbered list of AppArmor rules that *could* be added by pressing the number of the item on the list.

By default, logprof looks for profiles in /etc/subdomain.d and scans the log in /var/log/messages so in many cases, just running "logprof" as root will do the right thing.

However, there will be times when you need to search archived log files, such as if the program exercise period exceeds the log rotation window (when the log messages file is archived and the new log file is started). If this is the case, you can type:

```
zcat -f `ls -ltr /var/log/messages*` | logprof -f -
```

### Logprof Example 1

Following is an example of how logprof will address httpd2-pre-fork accessing the file /etc/group. The example uses [] to indicate the default option.

In this example, the access to /etc/group is part of httpd2-pre-fork accessing nameservices. The appropriate response is **1**, which pulls in a pre-defined set of AppArmor rules. Selecting **1** to #include the nameservice package forestalls all of the future questions pertaining to DNS lookups, and also makes the profile less brittle, in that any changes to DNS configuration and the associated nameservice profile package can be made once, rather than needing to revise many pro-

files.

```

Profile:  /usr/sbin/httpd2-prefork
Path:     /etc/group
New Mode: r

[1 - #include <abstractions/nameservice>]
 2 - /etc/group

[(A)llow] / (D)eny / (N)ew / (G)lob / Glob w/(E)xt /
Abo(r)t / (F)inish

```

Keystroke responses to this question are:

- **Press Enter:** Accepts the entry highlighted with []. Press the **Enter** key if you want to allow access to the selected directory path.
- **"A"llow:** Select Allow if you want to grant the program access to the specified directory path entries. AppArmor suggests file permission access. For more information on this, refer to "File Permission Access Modes" on page 42
- **"D"eny:** Select Deny to prevent the program from accessing the specified directory path entries. AppArmor will then move on to the next event.
- **"N"ew:** Prompts you to enter your own rule for this event, allowing you to specify whatever form of regular expression you want. If the expression you enter does not actually satisfy the event that prompted the question in the first place, AppArmor will ask you for confirmation and let you re-enter the expression.
- **"G"lob:** Select Glob once once to modify the directory path (by using wildcards) to include all files in the suggested entry directory. When you select Glob twice, access will be granted to all files and subdirectories beneath the one shown.

For more information on globbing syntax, refer to "Path Names and Regular Expression Matching" on page 41.

- **Glob w/"E"xt:** When you select Glob w/Ext, the original directory path is modified while retaining the filename extension. With one click, `/etc/apache2/file.ext` becomes `/etc/apache2/*.ext`, adding the wildcard (asterisk) in place of the file name. This will allow the program to access all files in the suggested directory that end with the ".ext" extension. When you select it twice, access will be

granted to all files (with the particular extension) and subdirectories beneath the one shown.

- **"E"dit:** Select **Edit** to edit the highlighted line. The new line will appear at the bottom of the list.
- **"Abo"r"t:** Aborts `logprof`, dumping all rule changes entered so far and leaving all profiles unmodified.
- **"F"inish:** Closes `logprof`, saving all rule changes entered so far and modifying all profiles.

## Logprof Example 2

In an example from profiling `vsftpd`, we see this question:

```
Profile:  /usr/sbin/vsftpd
Path:     /y2k.jpg
New Mode: r

[1 - /y2k.jpg]

(A)llow / [(D)eny] / (N)ew / (G)lob / Glob w/(E)xt
/ Abo(r)t / (F)inish
```

Several items of interest appear in this question. First, note that `vsftpd` is asking for a path entry at the top of the tree, even though `vsftpd` by default on SuSE LINUX Enterprise Server 9 serves FTP files from `/srv/ftp`. This is because `httpd2-prefork` uses `chroot`, and for the portion of the code inside the `chroot` jail, `AppArmor` sees file accesses in terms of the `chroot` environment, rather than the global absolute path.

The second item of interest is that we may want to grant FTP read access to *all* of the JPEG files in the directory, and so we could use the **Glob w/"E"xt** and use the suggested path of `"/* .jpg"`. Doing so will collapse all previous rules granting access to individual `.jpg` files, and forestall any future questions pertaining to access to `.jpg` files.

Finally, you may want to grant more general access to FTP files. If you select "G"lob last entry, then `logprof` will replace the suggested path of `/y2k .jpg` with `/*`. Or you may want to grant even more access to the entire directory tree, in which case you could use the "N"ew path

option, and enter `/**.jpg` (which would grant access to all .jpg files in the entire directory tree) or `/**` (which would just grant access to all files in the directory tree).

The above deal with read accesses. Write accesses are similar, except that it is *good policy* to be more conservative in your use of regular expressions for write accesses.

Dealing with execute accesses is more complex. You must decide which of the three kinds of execute permissions you intend to grant:

- **inherit (ix):** The child inherits the parent's profile, i.e. runs with the same access controls as the parent. This mode is useful when a confined program needs to call another confined program without gaining the permissions of the target's profile, or losing the permissions of the current profile. This mode is often used when the child program is a "helper application", such as the `/usr/bin/mail` client using the `less` program as a pager, or the Mozilla web browser using the `acrobat` program to display PDF files.
- **profile (px):** The child runs using its own profile, which must be loaded into the kernel. If the profile is *not* present, then attempts to execute the child will fail with permission denied. This is most useful if the parent program is invoking a global service, such as DNS lookups or sending mail via your system's MTA.
- **unconfined (ux):** The child runs **completely unconfined** without any AppArmor profile being applied to the executed resource.

In this example, we are profiling the `/usr/bin/mail` mail client<sup>1</sup> and `logprof` has discovered that `/usr/bin/mail` executes `/usr/bin/less` as a helper application to "page" long mail messages, and presents us with this prompt:

```
/usr/bin/nail -> /usr/bin/less
(I)nherit / (P)rofile / (U)nconstrained / (D)eny
```

The program `/usr/bin/less` appears to be a simple one for scrolling through text that is more than one screen long, and that is in fact what `/usr/bin/mail` is using it for. However, `less` is actually a large and powerful program that makes use of many other helper appli-

---

1. The actual executable file for `/usr/bin/mail` turns out to be `/usr/bin/nail` which is not a typographical error.

cations, such as `tar` and `rpm`.<sup>1</sup>

We notably do *not* want to automatically invoke `rpm` when reading mail messages (that leads directly to a Microsoft Outlook style virus attack, because `rpm` needs the power to install and modify system programs) and so in this case the best choice is to use “I”nherit. This will result in the less program executed *from this context* running under the profile for `/usr/bin/mail`. This has two consequences:

- We will need to add all of the basic file accesses for `/usr/bin/less` to the profile for `/usr/bin/mail`.
- We can *avoid* adding the helper applications such as `tar` and `rpm` to the `/usr/bin/mail` profile, so that when `/usr/bin/mail` runs `/usr/bin/mail/less` in this context, the less program is *far less dangerous* than it would be without AppArmor protection.

In other circumstances, we may instead want to use the “P”rofile option. This has two effects on `logprof`:

- The rule written into the profile is `px`, which forces the transition to the child’s own profile.
- `logprof` constructs a profile for the child and starts building it, in the same way that it built the parent profile, by ascribing events for the child process to the child’s profile and asking the `logprof` user questions as above.

Finally, we might want to grant the child process *very* powerful access by specifying “U”nconfined. This writes “ux” into the parent profile, so that when the child runs, it runs without any AppArmor profile being applied at all. This means running with no protection, and should only be used when absolutely required.

### **Subdomain.vim**

A syntax coloring file for the vim text editor highlights various features of a AppArmor profile with colors. Using vim and the AppArmor syntax mode for vim, you can see the semantic implications of your profiles with color highlighting. Use vim to view and edit your profile by typing

---

1. Run less on a tar ball or an rpm file and it will show you the inventory of these containers.

vim at a terminal window.

To enable the syntax coloring, when you edit a AppArmor profile in vim, use the command `:syntax on` and then `:set syntax=subdomain`. Alternately, you can place these lines in your `~/.vimrc` file:

```
syntax on
set modeline
set modelines=5
```

When vim opens the profile, first enter `:syntax on` and then `:set syntax=subdomain` and vim will color the lines of the profile for you:

**blue:** #include lines that pull in other AppArmor rules, and comments that begin with #

**white:** ordinary read access lines

**brown:** capability statements and complain flags

**yellow:** lines that grant write access.

**green:** lines that grant execute permission, either `ix` or `px`

**red:** lines that grant *unconfined* access, `ux`

**red background:** syntax errors that will not load properly into the SubDomain module.

**Note:** There is a security risk to using these lines in your `.vimrc` file, as it causes vim to trust the syntax mode presented in files you are editing. It may enable an attacker to send you a file to be opened with vim that may do something unsafe.

man `subdomain.vim`, man `vim`, and `:help syntax` from within the vim editor for particulars. The SubDomain module syntax is stored in `/usr/share/vim/vim61/syntax/subdomain.vim`

### **Unconfined**

Examines open network ports on your system, compares that to the set of profiles loaded on your system, and reports network services that do not have SubDomain profiles. `unconfined` requires root privilege, and that it not be confined by a AppArmor profile.

`unconfined` must be run as root to retrieve the process executable



link from the `proc` filesystem. This program is susceptible to the following **race** conditions:

- an unlinked executable will be mishandled
- an executable started before a AppArmor profile is loaded will not appear in the output, despite running without confinement
- a process that dies between the `netstat(8)` and further checks will be mishandled.

**Note:** This program only lists processes using TCP and UDP. In short, this program is unsuitable for forensics use and is provided only as an aid to profiling all network-accessible processes in the lab.

For more information on the science and security of AppArmor, refer to the following papers:

- “SubDomain: Parsimonious Server Security”. Crispin Cowan, Steve Beattie, Greg Kroah-Hartman, Calton Pu, Perry Wagle, and Virgil Gligor. Describes the initial design and implementation of SubDomain. Published in the proceedings of the USENIX LISA Conference, December 2000, New Orleans, LA.

**Note:** The above paper is now out of date, describing syntax and features that are different from the present SubDomain/AppArmor. This paper should be used only for scientific background, and not for technical documentation.

- “Defcon Capture the Flag: Defending Vulnerable Code from Intense Attack”. Crispin Cowan, Seth Arnold, Steve Beattie, Chris Wright, and John Viega. A good guide on strategic and tactical use of SubDomain to solve severe security problems in a very short period of time. Published in the Proceedings of the DARPA Information Survivability Conference and Expo (DISCEX III), April 2003, Washington DC.

## Path Names and Regular Expression Matching

Regular Expression Matching, or Globbing, is when you modify the directory path using wildcards to include a group of files or subdirectories. File resources may be specified with a globbing syntax similar to that used by popular shells, such as `csh(1)`, `bash(1)`, `zsh(1)`.

- \* Can substitute for any number of characters, except '/'  
**For example:** an arbitrary number of path elements, including entire directories.
  - \*\* Can substitute for any number of characters, including '/'  
**For example:** an arbitrary number of path elements, including entire directories.
  - ? Can substitute for any single character, except '/'
- [abc]** This will substitute for the single character a, b, or c  
**For example:** a rule that matches `/home[01]/*.plan` that allows a program to access .plan files for users in both /home0 and /home1.
- [a-c]** This will substitute for the single character a, b, or c
- {ab,cd}** This will expand to one rule to match ab, one rule to match cd.  
**For example:** A rule that matches `/{usr,www}/pages/**` to grant access to web pages in both /usr/pages and /www/pages.

## File Permission Access Modes

File permission access modes consists of combinations of the following six modes:

- r: Reade Mode
- w: Write Mode
- px: Discrete Profile Execute Mode
- ux: Unconstrained Execute Mode
- ix: Inherit Execute Mode
- l: Link Mode

### **Read Mode**

Allows the program to have read access to the resource. Read access is required for shell scripts and other interpreted content, and determines if an executing process can core dump or be attached to with `ptrace(2)`. (`ptrace(2)` is used by utilities such as `strace(1)`, `ltrace(1)`, and `gdb(1)`.)

***Write Mode***

Allows the program to have write access to the resource. Files must have this permission if they are to be unlinked (removed).

***Discrete Profile Execute Mode***

This mode requires that a discrete security profile is defined for a resource executed at a AppArmor domain transition. If there is no profile defined then the access will be denied. Incompatible with Inherit and Unconstrained execute entries.

***Unconstrained Execute Mode***

Allows the program to execute the resource without any AppArmor profile being applied to the executed resource. Requires listing execute mode as well. Incompatible with Inherit and Discrete Profile execute entries.

This mode is useful when a confined program needs to be able to perform a privileged operation, such as rebooting the machine. By placing the privileged section in another executable and granting unconstrained execution rights, it is possible to bypass the mandatory constraints imposed on all confined processes. For more information on what is constrained, see the `subdomain(7)` man page.

***Inherit Execute Mode***

Prevent the normal AppArmor domain transition on `execve(2)` when the profiled program executes the resource. Instead, the executed resource will inherit the current profile. Incompatible with Unconstrained and Discrete Profile execute entries. This mode is useful when a confined program needs to call another confined program without gaining the permissions of the target's profile, or losing the permissions of the current profile. This mode is infrequently used.

***Link Mode***

Allows the program to be able to create and remove a link with this name (including symlinks). When a link is created, the file that is being linked to MUST have the same access permissions as the link being

created (with the exception that the destination does not have to have link access.) Link access is required for unlinking a file.

## Chapter 5 Managing Profiled Applications

After creating profiles and Immunizing your applications, the SLES 9 system will be more efficient and better protected if you perform AppArmor profile maintenance, which involves tracking common issues and concerns.

Applications that are confined by AppArmor security profiles will generate messages when applications execute in unexpected ways, or outside of their specified profile. These messages can be monitored by

- **Monitoring Your Secured Applications**, see page 45.
- **Maintaining Your Security Profiles**, see page 55.

### Monitoring Your Secured Applications

Applications that are confined by AppArmor security profiles will generate messages when applications execute in unexpected ways, or outside of their specified profile. These messages can be monitored by event notification, generating periodic reports, or integration into a 3rd party reporting mechanism. The following sections provide detail on how to use these features and where to find additional resources.

- **Setting Up Event Notification**, see page 45.
- **Reports**, see page 49.
- **Reacting to Security Events**, see page 45.

### Setting Up Event Notification

Security event notification is an AppArmor feature that informs a specified email recipient when systemic AppArmor activity occurs.

When you enter an email address, you are notified via email when AppArmor Security events occur. You can enable three types of notifications, which are:

- **Terse:** Terse notification summarizes the total number of system events without providing details. For example:

```
dhcp-101.up.wirex.com has had 10 security events since Tue  
Oct 12 11:10:00 2004
```

- **Summary Notification:** The Summary notification displays the logged AppArmor security events and lists the number of individual occurrences, including the date of the last occurrence. For example:

```
SubDomain: PERMITTING access to capability 'setgid'  
(httpd2-prefork(6347) profile /usr/sbin/httpd2-prefork  
active /usr/sbin/httpd2-prefork) 2 times, the latest at Sat  
Oct 9 16:05:54 2004.
```

- **Verbose Notification:** The Verbose notification displays unmodified, logged AppArmor security events. It tells you every time an event occurs and writes a new line in the Verbose log. These security events include the date and time the event occurred, when the application profile permits access as well as rejects access, and the type of file permission access that is permitted or rejected. Verbose Notification also reports several messages that the logprof tool (see “Logprof” on page 34) uses to interpret profiles. For example:

```
Oct 9 15:40:31 SubDomain: PERMITTING r access to  
/etc/apache2/httpd.conf (httpd2-prefork(6068) profile  
/usr/sbin/httpd2-prefork active /usr/sbin/httpd2-prefork)
```

**Note:** To configure Event Notification, refer to “Configure Security Event Notification” on page 46. After configuring security event notification, read the reports and determine whether events require follow up. Follow up may include the procedures outlined in “Receiving a Security Event Rejection” on page 54.

### **Severity Level Notification**

You can set up AppArmor to send you event messages for things that are in the severity database and above the level that you select. These are numbered one through ten, ten being the most severe security incident. The `severity.db` file defines the severity level of potential security events. The severity levels are determined by the importance of different security events, such as certain resources accessed or services denied.

### **Configure Security Event Notification**

Security event notification is an AppArmor feature that informs you

when systemic AppArmor activity occurs. When you select a notification frequency (receiving daily notification, for example), you activate the notification. You are required to enter an email address, so you can be notified via email when AppArmor Security events occur.

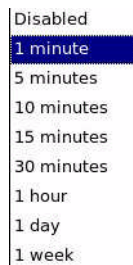
**Note:** You must set up a mail server on your SLES 9 server that can send outgoing mail using the smtp protocol (for example, sendmail, postfix, or qmail) in order for Event Notification to work.

1. In the Enable Security Event Notification section of the window, click the **Configure** button. The Enable Security Event Notification pop-up window displays.

The screenshot shows a window titled "Security Event Notification" with a close button in the top right corner. The window is divided into three sections: "Terse Notification", "Summary Notification", and "Verbose Notification". Each section has a "Frequency" dropdown, an "Email Address" text field, and a "Severity" spinner. In the "Terse Notification" section, "Frequency" is set to "15 minutes", "Email Address" is "Leona@immunix.com", and "Severity" is "7". There is a checked checkbox for "Include Unknown Severity Events". The "Summary Notification" section has "Frequency" set to "Disabled", "Email Address" is empty, and "Severity" is "0". It also has a checked checkbox for "Include Unknown Severity Events". The "Verbose Notification" section has "Frequency" set to "1 week", "Email Address" is "Leona@immunix.com", and "Severity" is "2".

2. In the Enable Security Notification window, you have the option to enable **Terse**, **Summary** or **Verbose** event notification, which are defined in "Severity Level Notification" on page 46. If you would like to be notified of AppArmor security events, select the type of notification you would like sent via email.
3. Enter the email addresses of those who should receive notification in the field provided. If notification is enabled, you must enter an email address.

4. For each notification type that you would like enabled, select the frequency of notification that you would like. The pull-down list displays:



The pull-down list options are detailed below:

- **1 minute:** A notification email is sent every one minute, informing you of AppArmor security events, if an event occurs.
  - **5 minutes:** A notification email is sent every five minutes, informing you of AppArmor security events, if an event occurs.
  - **10 minutes:** A notification email is sent every ten minutes, informing you of AppArmor security events, if an event occurs.
  - **15 minutes:** A notification email is sent every fifteen minutes, informing you of AppArmor security events, if an event occurs.
  - **30 minutes:** A notification email is sent every thirty minutes, informing you of AppArmor security events, if an event occurs.
  - **1 hour:** A notification email is sent every hour, informing you of AppArmor security events, if an event occurs.
  - **1 day:** A notification email is sent daily, informing you of AppArmor security events, if an event occurs.
  - **1 week:** A notification email is sent weekly, informing you of AppArmor security events, if an event occurs.
5. Select the **lowest severity** level for which a notification should be sent. Security events will be logged and the notifications will be sent at the time indicated by the interval when events are equal or greater than the selected severity level. If the interval is 1 day, the notification will be sent daily, if security events occur. Refer to *"Severity Level Notification"* on page 46 for more information about severity levels.
  6. Click the **OK** button.
  7. Click the **Done** button in the AppArmor Configuration window.



8. Click the **Close** button in the YaST Control Center window.

## Reports

AppArmor's Reporting feature adds flexibility by enhancing the way users can view system event data. The reporting tool enables users to do the following:

- export reports
- schedule periodic reports
- specify date ranges
- filter data
- view archived reports

Using reports, you can read important AppArmor security events reported in the log files without sifting through the cumbersome messages only useful to the logprof tool. You can narrow down the size of the report by filtering by date range or program name. You can also export an html or csv file.

1. To run reports, open the YaST GUI and click **SubDomain Security Module**. The AppArmor interface displays.
2. From AppArmor, click the View Security Events icon. The Reports window displays.
3. From the Reports window, select an option:
  - **Set Schedule:** In Set Schedule, you can schedule periodic reports to be run, automatically.
  - **View Archive:** In View Archive, you can specify the location of a cumulation of reports from one or more systems and view the results together in one report.

## Scheduling Reports

When a report is set up in AppArmor Scheduling Reports, it will periodically launch a report of AppArmor security events that have occurred on the system. You are able to configure a daily, weekly or monthly report to run for a specified period. You can set the report to display rejections for certain severity levels, or to filter by program name, profile name, severity level, and/or denied resources.

In the Scheduling Reports tab, you are given the option to edit a report, create a new report, delete a report, disable report, or close. These options are described in the following sections.

### Create New Report

When you click the Create New Report button, the Schedule New Report dialog box displays. From this box, you can specify and narrow-down report details. The following fields are available:

- **Report Name:** Here you can specify the name of the report. Use names that will easily discern one report from the next.
- **Run Monthly:** Select a beginning month to activate monthly filtering in reports. If you select **NA**, monthly filtering will not be performed.
- **Run Weekly:** Select a day of the week for which to schedule the weekly reports. If you select **NA**, weekly filtering will not be performed. If monthly reporting is selected, you can specify the day of the month that the report is run.
- **Time To Run:** Select the time in the Time to Run field to specify the time that you would like the reports to run. If you don't change the time, selected reports will run at midnight. If neither month nor day of week is selected, the report will run daily at the specified time.
- **Program Name:** When you enter a program name or pattern that matches the name of the binary executable of the program of interest, the report will display security events that have occurred for a specific program.
- **Profile Name:** When you enter the name of the profile, the report will display the security events that are generated for the specified profile. You can use this to see what is being confined by a specific profile.
- **Severity Level:** Select or enter the lowest severity level for security events that you would like to be included in the report. The selected severity level, and above, will be included in the reports.
- **Denied Resources:** A source to which the profile has denied access. This includes capabilities and files. you can see what resources are not allowed to be accessed by profiles.

### Edit Report

Scheduling Reports provides the ability to edit existing reports.

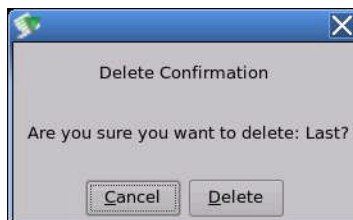
1. To edit a report, highlight the report in the List of AppArmor Scheduled Reports.
2. Click the Edit a Report button, the Edit a Report dialog box displays as shown in the image below:
3. Edit the report details, then click the **Save** button.

**Note:** Refer to “Create New Report” on page 50 if you would like definitions of the fields in this form.

### Delete Report

Delete a Report enables you to permanently remove a report from the List of AppArmor Scheduled Reports. If you just want to temporarily deactivate a report, refer to “Disable Report” on page 51 To delete a report, follow these instructions:

1. To remove a report from the list of reports, highlight the report and click the **Delete** button. The Delete Confirmation window displays.



2. If you are sure you want to continue, click the **Create** button.

### Disable Report

Disabling a report inactivates it in the List of AppArmor Scheduled Reports. The report remains in the list of reports, but does not run until it is activated.

1. To disable a report, highlight the report in the List of AppArmor Scheduled Reports.
2. Click the **Disable** button. The report can be re-enabled by clicking the Disable button again.

**Close**

Click the **Close** button to return to the AppArmor main menu.

**Creating Reports**

Creating Reports enables you to run an immediate report of either all AppArmor security events, or one that is customized. You can customize the report by filtering dates or names of the programs being accessed. The report can be exported to an html or Comma Separated Values (CSV) file format. The report, which will be displayed in the Report Viewer can be sorted by Date filter, Profile, Process ID number (PID), or AppArmor message (description of security event).

3. from the following parameters to report security event data:

- **View All Reports:** Displays all the security events.
- **Filter By Date Range:** Enables you to specify a date and time range to narrow down the security events you would like to view.

- **Filter By Program Name:** Enables you to narrow down events that only pertain to the program you specify.

- **Export Report:** Enables you to export a CSV (comma separated values) or html file. The CSV file separates pieces of data in the log entries with commas using a standard data format for importing into table-oriented applications. You can enter a pathname for your

exported report by typing in the full pathname in the field provided.



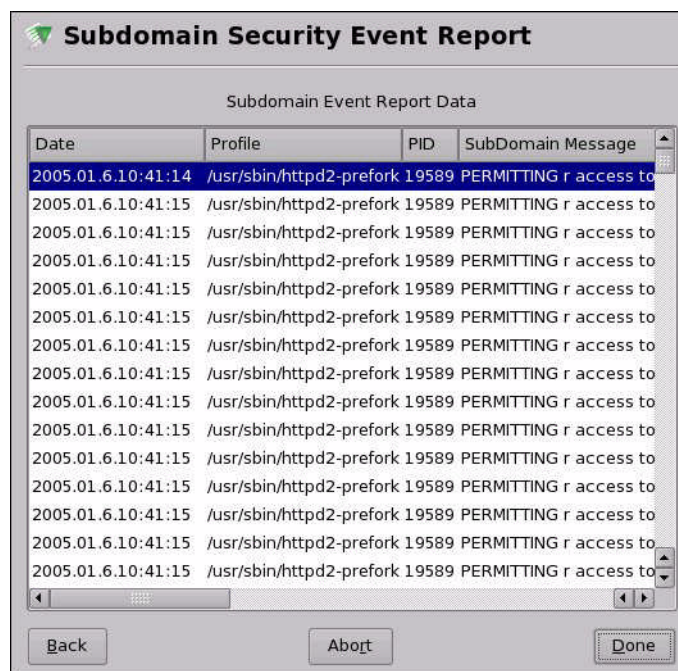
Export Report

Export File Location: /tmp/export.log

Select Export Format: ☒ CSV ☒ HTML

**Note:** View All Reports cannot be selected with Filter By Program Name or Filter By Date Range.

4. View the report data in the AppArmor Security Event Report window.



**Subdomain Security Event Report**

Subdomain Event Report Data

Date	Profile	PID	SubDomain Message
2005.01.6.10:41:14	/usr/sbin/httpd2-prefork	19589	PERMITTING r access to
2005.01.6.10:41:15	/usr/sbin/httpd2-prefork	19589	PERMITTING r access to
2005.01.6.10:41:15	/usr/sbin/httpd2-prefork	19589	PERMITTING r access to
2005.01.6.10:41:15	/usr/sbin/httpd2-prefork	19589	PERMITTING r access to
2005.01.6.10:41:15	/usr/sbin/httpd2-prefork	19589	PERMITTING r access to
2005.01.6.10:41:15	/usr/sbin/httpd2-prefork	19589	PERMITTING r access to
2005.01.6.10:41:15	/usr/sbin/httpd2-prefork	19589	PERMITTING r access to
2005.01.6.10:41:15	/usr/sbin/httpd2-prefork	19589	PERMITTING r access to
2005.01.6.10:41:15	/usr/sbin/httpd2-prefork	19589	PERMITTING r access to
2005.01.6.10:41:15	/usr/sbin/httpd2-prefork	19589	PERMITTING r access to
2005.01.6.10:41:15	/usr/sbin/httpd2-prefork	19589	PERMITTING r access to
2005.01.6.10:41:15	/usr/sbin/httpd2-prefork	19589	PERMITTING r access to
2005.01.6.10:41:15	/usr/sbin/httpd2-prefork	19589	PERMITTING r access to
2005.01.6.10:41:15	/usr/sbin/httpd2-prefork	19589	PERMITTING r access to
2005.01.6.10:41:15	/usr/sbin/httpd2-prefork	19589	PERMITTING r access to

Buttons: Back, Abort, Done

5. Click the **Done** button to close the window.

**Note:** You can also click the **Back** button if you want to re-run the report with new parameters or to export it after viewing it.

6. Click the **Close** button in YaST.

### Viewing Archived Reports

Viewing Archived Reports enables you to specify the location of a cumulation of reports from one or more systems, including the ability to

filter by date or names of programs being accessed and display them all together in one report.

## Reacting to Security Events

There are a few common maintenance issues that you should regularly inspect and deal with according to the rules that you have established. The following are some common maintenance issues that you might encounter:

- **Receiving a security event rejection** for an application via email or viewing in reports. See page 54.
- **Changing security** of applications with existing profiles. See page 55.

### *Receiving a Security Event Rejection*

The user intervention required when you receive a rejection is to examine the access violation and determine if that event indicated a threat or was part of normal application behavior. Application-specific knowledge is required to make the determination.

To look at security events, type the following in a terminal window as root:

```
tail -f /var/log/messages
```

In the list of log messages that displays, look for events that are rejections. If the rejection is not part of normal application behavior, then this access should be considered a possible intrusion attempt (that was prevented) and this notification should be passed to the person responsible for security within your organization.

### *Changing Application Security*

Users can manually edit a profile by using `vim` at the console. To change your profiles using the console to edit a profile, you would open the particular profile in `/etc/subdomain.d` using `vim`. Refer to “Edit an AppArmor Profile” on page 22.

## Maintaining Your Security Profiles

In a production environment, you should plan on maintaining profiles for all of the deployed applications. The security policies are an integral part of your deployment. You should plan on taking steps to backup/restore security policy files, plan for software changes, and allow any needed modification of security policies that your environment dictates. These items are covered in the following sections:

- **Backing up your security profiles**, see page 55.
- **Changing your security profiles**, see page 55.
- **Introducing new software into your environment**, see page 56.

### *Backing Up Your Security Profiles*

Because you take the time to make profiles, it makes sense to back them up. Backing up profiles might save you from having to re-profile all your programs after a disk crash. Also, if profiles are changed, you can easily restore previous settings by using the backed up files.

Backing up profiles can be done by copying the profile files to a specified directory.

1. You should first archive the files into one file. To do this, open a terminal window and type the following as root:

```
tar -cvf ~/profile_backup.tar /etc/subdomain.d
```

2. The simplest method to ensure that your security policy files are regularly backed up is to include the directory `/etc/subdomain.d` in your list of directories that you backup system archives.

**Note:** You can also use `scp` or a file management GUI to store the files on some kind of storage media, on the network, or on another PC.

### *Changing Your Security Profiles*

Maintenance of security profiles includes changing them if you decide that your system requires more or less security for its applications. To change your profiles using the console to edit a profile, you would open the particular profile in `/etc/subdomain.d` using `vim`. Refer to “Edit an AppArmor Profile” on page 22.

## ***Introducing New Software Into Your Environment***

When you add a new application version or patch to your system, you should always update the profile to fit your needs. You have several options that depend on your company's software deployment strategy. You can deploy your patches and upgrades into a test or production environment and the following explain how to go about it with each method.

If you intend to deploy a patch or upgrade in a *test* environment, perform the following:

- Run GenProf by typing `genprof` in a terminal while logged in as root. For detailed instructions, refer to "Genprof" on page 29.

If you intend to deploy a patch or upgrade directly into a *production* environment, the your best method for updating your profiles is to:

- Monitor the system frequently to determine if any new rejections should be added to the profile and update as needed using `logprof`. For detailed instructions, refer to "Logprof" on page 34.



## Chapter 6 Profiling Your Web Applications Using ChangeHat Apache

### What is ChangeHat?

A SubDomain profile represents security policy for an individual program instance, or process. It applies to an executable program, but if a portion of the program needs different access permissions than other portions, the program can "change hats" to use a different security context, distinctive from the access of the main program. This is known as a Hat or subprofile.

ChangeHat enables programs to change to or from a "hat" within a SubDomain profile. It enables you to define security at a finer grain level than the process.

This feature requires that each application be made "changehat aware" meaning that it is modified to make a request to the SubDomain module to switch security domains at arbitrary times during the application execution.

A profile can have an arbitrary number of subprofiles, but there are only 2 levels: a subprofile *cannot* have further sub-subprofiles. A subprofile is written as a separate profile, and named as the containing profile followed by the subprofile name, separated by a ^. Subprofiles must be stored in the same file as the parent profile.

**Note:** For more information see "man change\_hat" on your system.

### Apache ChangeHat

Immunix provides a `mod_change_hat` module for the Apache program. The `mod_change_hat` module works on your SLES 9 system to make the Apache web server become "ChangeHat-aware." It is installed if Apache is on your system. When Apache is ChangeHat-aware, with every URI request that it receives, it checks for the following customized SubDomain security profiles in the order given:

- URI-specific hat (for example, ^phpsysinfo-dev/templates/classic/images/bar\_left.gif)

- DEFAULT\_URI
- HANDLING\_UNTRUSTED\_INPUT

If you have the required Apache 2 on your system, the `mod_change_hat` module will be automatically installed with AppArmor as well as added to the apache configuration. Apache 1.3 is not supported.

**Note:** Should you install `mod_change_hat` without AppArmor, you need to make sure the Apache load module has a command in the config file that loads the `mod_change_hat` module by adding the following line to your Apache configuration file:

```
LoadModule change_hat_module modules/mod_change_hat.so
```

### ***Tools for Managing ChangeHat Aware Applications***

As with most of the SubDomain tools, you can use two methods for managing ChangeHat, the AppArmor Console program. Managing ChangeHat-aware applications has much more flexibility at the command-line, but it's also more complicated. They both allow you to manage the hats for your application and populate them with profile entries.

In the following steps, we walk you through a demo that will add Hats to an Apache profile using the AppArmor Console program. While running `Genprof` (refer to "Genprof" on page 29) the Immunix profiling utilities will prompt you to create new Hats for distinct URI requests. Choosing to create a new Hat will allow you to create individual profiles for each URI. This allows you to create very tight rules for each request.

If the URI that is processed does not represent significant processing, or otherwise doesn't represent a significant security risk, then you may safely select "Use Default Hat" to just process this URI in the default Hat, which is the default security profile.

In the demo, we create a new Hat for the URI `phpsysinfo-dev` and its subsequent accesses. Using the profiling utilities, we delegate what is added to this new hat. The resulting Hat becomes a tight-security container that encompasses all the processing on the server that occurs when the `phpsysinfo-dev` URI is passed to the Apache web server.

In this demo, we will generate a profile for the application **phpsysinfo** (refer to <http://phpsysinfo.sourceforge.net> for more information). The **phpsysinfo-dev** package is assumed to be installed under `/srv/www/htdocs/phpsysinfo-dev/` onto a clean (new) install of AppArmor.

1. Once **phpsysinfo-dev** is installed, you are ready to add Hats to the Apache profile. Type `genprof httpd2-prefork` and press enter.
2. The terminal window should now appear as follows:

```
Writing updated profile for /usr/sbin/httpd2-
prefork.
Setting /usr/sbin/httpd2-prefork to complain mode.

Please start the application to be profiled in
another window and
exercise its functionality now. Once completed,
select one of the
following options:

Profiling: /usr/sbin/httpd2-prefork

[(S)can for more SubDomain events] / (F)inish
```

3. The tool prompts you to run the application to be profiled in another terminal window. Perform as many of the application functions as possible so learning mode can log the files and directories the program requires access to in order to function properly. In this example, you should:

**Note:** Any program you are profiling, you would restart at this point.

- Restart Apache by typing `/etc/init.d/apache2 stop` and then `/etc/init.d/apache2 start` in a terminal window while logged in as root.
- Type **`http://localhost/phpsysinfo-dev/`** into an Internet browser window. The browser window should display network usage and system information.

**Note:** To ensure that this request is processed by the server and that you don't review cached data in your browser, you should refresh the

page. To do this - click the browser **Refresh** button to make sure that Apache processes the request for the `phpsysinfo-dev` URI.

4. Click the "S" key. `genprof` will run `logprof` against the system log from where it was marked when `genprof` was started, and reload the profile. AppArmor uses the Logprof ability to scan all the information learned in the previous step and begins to prompt you with profile questions.

**Note:** You can select "F" from the `genprof` menu to exit.

5. In our demo, Logprof first prompt us with **Add Requested Hat** or **Use Default Hat** because it noticed that a URI was accessed `phpsysinfo-dev`. Select **Add Requested Hat**.

```
Reading log entries from /var/log/messages.
Updating subdomain profiles in /etc/subdomain.d.

Profile:          /usr/sbin/httpd2-prefork
Default Hat:      DEFAULT_URI
Requested Hat:    /phpsysinfo-dev/

[(A)dd requested hat]/ (U)se Default/ (D)eny/
Abo(r)t/ (F)inish
```

6. Click the "A" key. Choosing Add Requested Hat in the previous step creates a new hat in the profile. Subsequent menu questions about the script's actions will be added to the newly created hat rather than to the default hat for this application.

In the next menu, AppArmor displays the external program that the script executed, which is `/bin/bash` in our example.

```
Profile: /usr/sbin/httpd2-prefork^/phpsysinfo-dev/
Execute: /bin/bash
[(I)nherit] / (U)nconstrained / (D)eny / Abo(r)t /
(F)inish
```

Your choices are to specify that the program should run

• **(I)nherit:** confined by the `phpsysinfo-dev` hat.

•**(U)nconfined**: unconfined or without any security profile.

**Note:** Selecting unconfined can make a significant security hole and should be done with caution. The **Profile** option is not available for `/bin/bash` because having a profile could disrupt operating system functionality.

7. Select **inherit** for the `/bin/bash` path. This will add `/bin/bash` (accessed by Apache), to the `phpsysinfo-dev` hat profile with the necessary permissions. In the menu, AppArmor displays the next external program executed by the script, as follows:

```
Profile: /usr/sbin/httpd2-prefork^/phpsysinfo-dev/
Execute: /usr/bin/who

(I)nherit / [(P)rofile] / (U)nconfined / (D)eny /
Abo(r)t / (F)inish)
```

Your choices are to specify that the program should run:

- **(I)nherit**: confined by the `phpsysinfo-dev` hat.
  - **(P)rofile**: confined by a separate profile. A new profile will be created for the program if one does not already exist.
  - **(U)nconfined**: unconfined or without any security profile.
8. Select **Profile** for the `/usr/bin/who` path. A new profile is created for the `who` program and AppArmor continues with similar questions.
  9. The remaining questions prompt you how to generate new profiles for hats, and about adding entries to existing profiles. The process of adding entries to profiles is covered in detail in the section “Genprof” on page 29.
  10. click the “F” key to save your changes and exit the wizard when all profiling questions are answered and you are returned to the following start menu:

```
(S)can for more SubDomain events] / (F)inish
```

### ***phpsysinfo Hat (Subprofile)***

The following is an example of what a `phpsysinfo-dev` Hat might

resemble.

**Note:** The profile, **^phpsysinfo-dev**, is only valid in the context of a process running under the parent profile **httpd2-prefork**.

```
^phpsysinfo {
#include <program-chunks/base-files>
/bin/df ix,
/bin/bash ix,
/dev/tty rw,
/etc/SuSE-release r,
/etc/fstab r,
/etc/hosts r,
/etc/mtab r,
/proc/** r,
/sbin/lspci ix,
/srv/www/htdocs/sysinfo/** r,
/sys/bus/pci/devices r,
/sys/devices/** r,
/usr/bin/who ix,
/usr/share/pci.ids r,
/var/log/apache2/{access,error}_log w,
/var/run/utmp r,
}
```

### ***Running Genprof While Using ChangeHat***

AppArmor provides ChangeHat options for adding the requested hat, using the default hat, denying access for the hat, aborting the process, or finishing.

Select one of the following options that will satisfy the request:

- **Inherit:** stay in the same security profile (parent's profile)
- **Profile:** requires that a separate profile exists for the executed program.
- **Unconfined:** program executed without a security profile.  
Note: unless absolutely necessary you don't want to run unconfined.
- **Deny:** Select Deny to prevent the program from accessing the

specified directory path entries.

- **Abort:** Aborts `logprof`, dumping all rule changes entered so far and leaving all profiles unmodified.
- **Finish:** Closes `logprof`, saving all rule changes entered so far and modifying all profiles.

## Apache Configuration for mod\_change\_hat

Apache has configuration files that customize the way Apache functions.

Apache is configured by placing directives in plain text configuration files. The main configuration file is usually `httpd.conf`. When you compile Apache, you can indicate the location of this file. Directives can be placed in any of these configuration files to alter the way Apache behaves. When you make changes to the main configuration files, you need to start or restart Apache so the changes will be recognized.

### ***vhosts Directives***

Vhosts directives control whether requests that contain trailing path-name information, following an actual filename (or non-existent file in an existing directory), will be accepted or rejected. For Apache documentation on Virtual Host directives, refer to

<http://httpd.apache.org/docs-2.0/mod/core.html#virtualhost>

The `change_hat` specific configuration keyword is `ImmDefaultHatName` and is used similarly to `ImmHatName`, for example `ImmDefaultHatName My_Funky_Default_Hat`.

The configuration option is actually based on what's called a *server directive*, which enables you to use the keyword outside of other options, thereby setting it for the default server. Virtual hosts are considered internally within apache to be separate "servers", so you can set a default hat name for the default server, as well as one for each virtual host, if desired.

When a request comes in to be handled, the following steps reflect the

sequence in which `mod_change_hat` attempts to apply hats.

1. A location/directory hat as specified by the `ImmHatName` keyword.
2. A hat named by the entire URI path.
3. A default server hat as specified by the `ImmDefaultHatName` keyword.
4. `DEFAULT_URI` (and if none of those exist, it'll go back to the "parent" apache hat).

### ***Location and Directory Directives***

Location and Directory directives specify Hat names in the program configuration file so the program calls the hat regarding its security. For Apache, you can find documentation about Location and Directory directive at <http://httpd.apache.org/docs-2.0/sections.html>.

The **Location Directive** example below specifies that for a given location, `mod_change_hat` should use a specific hat:

```
<Location /foo/>
    ImmHatName MY_HAT_NAME
</Location>
```

This will try to use "MY\_HAT\_NAME" for any URI that begins with `/foo/` (e.g. `/foo/`, `/foo/bar`, `/foo/cgi/path/blah_blah/blah`, etc.).

The **Directory Directive** works similar to the Location Directive, except it refers to a pathname in the filesystem, as seen in the following example:

```
<Directory "/srv/www/www.immunix.com/docs"> # Note
lack of trailing slash
    ImmHatName immunix.com
</Directory>
```

**Example:** The program **phpsysinfo** is used to illustrate a Location directive in the following example. The tarball can be downloaded from <http://phpsysinfo.sourceforge.com>.

1. After downloading, install it into `/srv/www/htdocs/sysinfo/`.



2. Create `/etc/apache2/conf.d/sysinfo.conf` and add the following text to it:

```
<Location "/sysinfo">
    ImmHatName sysinfo
</Location>
```

The following hat should then work for phpsysinfo:

```
^sysinfo {
#include <program-chunks/base-files>
/bin/df ix,
/bin/bash ix,
/dev/tty rw,
/etc/SuSE-release r,
/etc/fstab r,
/etc/hosts r,
/etc/mtab r,
/proc/** r,
/sbin/lspci ix,
/srv/www/htdocs/sysinfo/** r,
/sys/bus/pci/devices r,
/sys/devices/** r,
/usr/bin/who ix,
/usr/share/pci.ids r,
/var/log/apache2/{access,error}_log w,
/var/run/utmp r,
}
```

3. Reload subdomain profiles by typing `/etc/init.d/subdomain restart` into a terminal window as root.
4. Restart apache by typing `/etc/init.d/apache2 restart` into a terminal window while logged in as root.
5. Enter `http://hostname/sysinfo/` into a browser to receive the system information that phpsysinfo delivers.
6. Track down configuration errors by going to the `/var/log/syslog` or running `dmesg` and looking for any rejections in the output.

## Chapter 7 Immunix Support

### Updating AppArmor Online

SuSE LINUX Enterprise Server 9 provides a continuous stream of updates for SLES 9 through the YOU agent, and from time to time those updates include revisions to the Linux kernel. When you update your Linux kernel, you also need to re-compile the SubDomain kernel module to again match your new kernel. The Immunix Application Armor (AppArmor) includes features to do this automatically, which we describe here.

When AppArmor is installed, it includes RPM triggers so that when the kernel is updated, RPM events fire that cause the SubDomain kernel module to be re-compiled. In most cases, this re-compiling of the SubDomain kernel module should happen quickly and silently, and you may not even notice it.

However, this re-compilation can fail for a variety of reasons, including not having all of the required devtools packages installed, having a Linux kernel source tree that does not match your running kernel, and not having a Linux kernel source tree at all.

To defend against such failure possibilities, SubDomain is configurable with respect to what you would like to do on boot if SubDomain fails to load:

**Warn:** Logs a warning message and proceeds to boot. This provides maximum availability, in that your computer boots and runs normally, but also may cause security vulnerabilities, because your machine is now running without SubDomain protection. This is the default behavior.

**Build:** Attempt to build a module that is compatible with the running kernel. If successful, then SubDomain will be loaded and run normally. If the compile is not successful, an error message is logged as in the Warn case.

**Panic:** If the SubDomain module fails to load, then a failure message is logged and the machine drops to single-user mode. This compromises availability, but preserves security, in that the machine will not

be exposed to the network without SubDomain protection.

**Build-panic:** If the SubDomain module fails to load, then it attempts to compile it, as in the Build case above. If building the module fails, then panic.

To control which of these options is in effect, as root edit the file `/etc/immunix/subdomain.conf` and uncomment the suitable line.

## Using the Man Pages

There are man pages available for your use. At a command prompt, type **man subdomain** to find reference to the subdomain manpage.

Man pages are distributed in sections. The sections are numbered 1 through 8. Each section is specific to a category of documentation. section 1 is user commands, section 2 is system calls, section 3 is library functions, section 4 is device driver information, section 5 is configuration file formats, section 6 is games, section 7 is high-level concepts, and section 8 is administrator commands.

The section numbers are used to distinguish manual pages from each other; for example, `exit(2)` describes the exit system call, while `exit(3)` describes the exit C library function.

The Immunix-provided man pages are as follows:

- `unconfined.8`
- `autodep.1`
- `clean.1`
- `complain.1`
- `enforce.1`
- `format_subdomain_files.pl.1`
- `genprof.1`
- `logprof.1`
- `change_hat.2`
- `logprof.conf.5`

- subdomain.conf.5
- subdomain.d.5
- subdomain.vim.5
- subdomain.7
- subdomain\_parser.8

## Troubleshooting Solutions

SubDomain operation can generate various errors. Here is a list of possible errors and how to resolve them.

If you run `logprof` as a non-root user such as `bob` you will likely see this error:

```
bob@localhost:~> /usr/sbin/logprof
Can't find subdomain_parser.
```

**Note:** You should run `logprof` only as `root`.

Running `genprof` as a non-root user produces a similar result:

```
bob@localhost:~> /usr/sbin/genprof
/usr/sbin/genprof must be run as root.
```

Again, run `genprof` only as `root`.

You must also run the `subdomain start` and `subdomain stop` scripts as `root`. Running them as a non-root user produces this result:

```
bob@localhost:~> /etc/init.d/subdomain stop
/sbin/subdomain_parser: Sorry. You need root
priveleges to run this program.
Unloading SubDomain profiles..failed
```

Finally, manually editing SubDomain profiles can introduce syntax errors. If you attempt to start or restart SubDomain with syntax errors in your profiles, you will see error results like this:

```
localhost:~ # /etc/init.d/subdomain start
Loading SubDomain profiles
Subdomain parser error, line 2: Found unexpected
character: 'h'
Profile /etc/subdomain.d/usr.sbin.squid failed to
load
failed
```

## Getting Online Support

You can visit our website at [www.immunix.com](http://www.immunix.com) for information on our company and products.

## Using Mailing List Support

We have a user driven mailing list at **immunix-users@mail.wirex.com**. You can subscribe to this list at <http://mail.wirex.com/mailman/listinfo/immunix-users>

The announcement list is for announcements only the email for it is: [announce@mail.wirex.com](mailto:announce@mail.wirex.com). You can subscribe to this list at <http://mail.wirex.com/mailman/listinfo/immunix-announce>

## Chapter 8 Glossary

**Apache:** Apache is a freely available Unix-based Web server. It is currently the most commonly used web server on the Internet. More information about Apache can be found at the Apache website, <http://www.apache.org>.

**GUI:** Acronym for Graphical User Interface. This term refers to a software front-end meant to provide an attractive and easy to use interface between a computer user and application, its elements include such things as: windows, icons, buttons, cursors and scroll bars.

**RPM:** The RPM Package Manager (RPM) is an open packaging system available for anyone to use, and works on Red Hat Linux as well as other Linux and UNIX systems. It is capable of installing, uninstalling, verifying, querying, and updating computer software packages. See <http://www.rpm.org/> for more information.

**SSH (Secure Shell):** A service that allows you to access your server from a remote computer and issue text commands through a secure connection.

**URI:** Universal Resource Identifiers. the generic term for all types of names and addresses that refer to objects on the World Wide Web. A URL is one kind of URI.

**URL (Uniform Resource Locator):** An abbreviation of Uniform Resource Locator, the global address of documents and other resources on the World Wide Web.

The first part of the address indicates what protocol to use, and the second part specifies the IP address or the domain name where the resource is located.

For example, in <http://www.immunix.com/index.html> , **http** is the protocol to use.

**Attack Signatures:** Patterns in system or network activity that signal a possible virus or hacker attack. Intrusion detection systems might use attack signatures to distinguish between legitimate and potentially malicious activity.

By not relying on attack signatures, SubDomain provides "pro-active" instead of "reactive" defense from attacks. This is better because there is no window of vulnerability where the attack signature has to be defined for SubDomain as it does for products using Attack Signatures to secure their networks.

**Vulnerabilities:** An aspect of a system or network that leaves it open to attack. Characteristics of computer systems that allow an individual to keep it from correctly operating, or that will allow unauthorized users to take control of the system. Design, administrative, or implementation weakness or flaw in hardware, firmware, or software. If exploited, a vulnerability could lead to an unacceptable impact in the form of unauthorized access to information or disruption of critical processing.

**Mandatory Access Control:** A means of restricting access to objects that is based on fixed security attributes assigned to users and to files and other objects. The controls are mandatory in the sense that they cannot be modified by users or their programs.

**Application firewalling:** SubDomain contains applications and limits the actions they are permitted to take. It uses privilege confinement to prevent attackers from using malicious programs on the protected server or even using trusted applications in unintended ways.

**Profile Foundation Classes:** or profile building blocks needed for common application activities such as DNS lookup and user authentication.

**Streamlined Access Control:** SubDomain provides streamlined access control for network services by specifying which files each program is allowed to read, write, and execute. This ensures that each program does what it is supposed to do, **and nothing else**.

**Host Intrusion Prevention (HIP):** This technology works with the operating system kernel to block abnormal application behavior in the expectation that the abnormal behavior represents an unknown attack. It works by blocking the malicious packet on the host at the network level, before they could 'hurt' the application they're targeting.

