USENIX Association

Proceedings of the 5th Annual Linux Showcase & Conference

Oakland, California, USA
November 5–10, 2001

A Secure Linux Platform

Nigel Edwards, Joubert Berger, Tse Huong Choo
Hewlett-Packard
{nigel_edwards, joubert_berger, tse-huong_choo}@hp.com

Abstract

This paper describes "HP Secure OS Software for Linux" (HP-LX) – a version of Linux that incorporates modifications into the kernel to improve security. A common attack strategy is to exploit a bug in a service causing it to execute code that downloads additional executables, and overwrites existing system executables and web pages. If the attack is in the form of a "worm", it will then probe the network looking for new targets.

This paper argues that incorporating additional features into the underlying operating system best resists such attacks. HP-LX has mechanisms that contain a process within a known part of the system and place severe limits on the damage that can be caused by attacks. These mechanisms restrict communication to constrain the ability to interfere with and probe the network or other processes. They protect the file system and can prevent even root from overwriting files. In addition HP-LX has extensive auditing mechanisms for detecting compromised processes.

1. Introduction

Linux is increasing in popularity as a platform for hosting web and other Internet services. With this increasing popularity we are starting to see an increase in the number of Linux specific attacks, for example so far in 2001 we have seen three significant "worms": Adore, Ramen and Lion. In addition "defacing" of web pages continues to occur with increasing frequency.

Figure 1 is generated from data obtained from Attrition [Attrition]. It shows that during the 12 months from May 2000 to April 2001, the monthly number of defaced Linux sites increased by more than 5 times. Not all defaced web sites are reported to Attrition, so the numbers on the vertical axis are a fraction of the total number of sites defaced.

So what can you do to make your Linux server more secure? There are three main strategies.

1. Keep up to date with patches
2. Run one or more security utilities
3. Strengthen the underlying kernel.

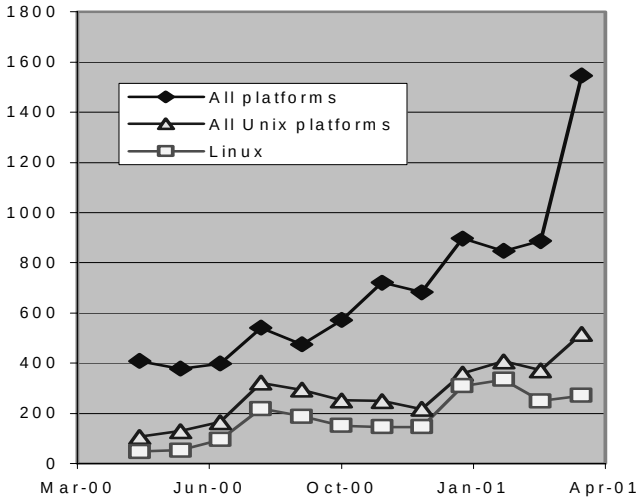

Figure 1: Web site defacement activity

It is very important to keep up to date with security patches. There is a lot of evidence to suggest that most sites are compromised after the flaw is identified and a patch is available [Lemos01]. However, a significant number of systems are compromised before the patch is available and this strategy does nothing to protect you against administration or installation errors: e.g. an application script in a web server that can be used to execute arbitrary shell commands.

You can also run one or more utilities that attempt to improve the security of various aspects of your system. For example, there are various scanners available that can scan your system for known vulnerabilities. Tripwire [Tripwire] will inform you if a change has been made to sensitive files so that you can change them back to their original form. Bastille [Bastille] will "lockdown" your system using the existing operating system security mechanisms. Its

features include securing the configurations of various popular daemons and services; changing the permissions on various files and executables to make them more secure; increasing logging from the default level.

All of this will improve your security.

However, there are limits to what can be achieved by layering security utilities on top of the existing Linux kernel. Tools that try to address known vulnerabilities are not completely reliable: a recent study [Forrestal01] reporting on vulnerability scanners showed that several of the currently available tools detected less than 60% of known vulnerabilities and the best detected less than 90%. In addition these tools cannot protect you against unknown vulnerabilities and can do little against system specific errors such as a faulty service available through a web server.

No security utility that is installed on top of the operating system can restrict the internal communication that takes place between processes on the system. This means a compromised process can interfere with and attack other processes. You cannot limit the communication ability of a particular process (e.g. the web server) to communicate with other hosts in your network without applying that restriction to all processes on the system. The access control mechanisms in Linux are discretionary – this means that any process that owns a file or directory can change the permission of that resource. Further any process that has write access to a file or directory can overwrite the entire contents. Most logging is performed by applications (e.g. writing to syslog). Linux does not provide facilities for logging the system calls executed by a process. As we shall explain, this limits the ability to detect an intrusion into the system.

We take a different approach. We accept that services will be compromised by as yet unknown vulnerabilities and mis-configuration. Our philosophy is to strengthen the underlying operating system to incorporate features that contain a process within a known part of the system, so even if it is compromised it can cause limited damage. The following are the main security features of HP-LX.

- Containment – the communication channels and files available to individual processes are rigorously controlled – we can prevent even root having write access to files
- Auditing of system calls to audit administration actions and to detect intrusion
- A secure administration model designed to avoid the need to share administrator passwords – root is not all powerful

What does this give us? Let's look at the strategy used by Ramen to compromise a host. Lestat documents this in detail in [Lestat01], here we give an overview abstracting from some of these details.

Ramen uses stack overflow vulnerabilities in the services rpc.statd, wu-ftpd and LPRng. Bugs in string processing code are exploited to overwrite the stack causing these processes to start executing code included at the end of the string. This code overwrites executables on your system, gains root access, downloads executables, and overwrites any index.html files (defacing your web pages). It then attempts to probe the network looking for other hosts to attack. When a host is found running a vulnerable version of rpc.statd, wu-ftpd or LPRng an attack on that host is launched, propagating the worm to that host.

The HP-LX security mechanisms are designed to prevent worm propagation, corruption of sensitive files on the system (web pages and executables) and to detect compromise of any service on the system. This means that result of an attack like Ramen should be limited in the worst case to unavailability of the attacked service until it is restarted. How does it do this?

HP-LX can restrict the communication channels available to a process and all its children. So general probing of the network is not possible. Typically we do not allow any outbound connections to be made from pure services (such as web servers), nor do we allow these services to send IP packets to any other hosts on the network other than to clients that already have established TCP connections. This prevents propagation of worms.

HP-LX can prevent write access to files including web pages to any user (including root). Access control is mandatory – the owner of the file cannot override it. This prevents web pages from being defaced and system executables from being overwritten.

HP-LX has extensive auditing of system calls. This can be used to build up profiles of processes, so that a change in behavior due to a compromise will be detected with high probability.

2. Review of security features

The primary purpose of HP-LX is to provide a secure environment in which to run services that are

accessible from the Internet. It is therefore designed to be robust against remote attacks and to provide an environment to secure services against remote attacks. It does not attempt to defend itself from attackers that have physical access to the machine.

In this section we give a technical overview of the following HP-LX security features.

- Containment
- File system integrity
- Lockdown of system configuration
- Audit
- The Secure administration model

These security features have been implemented by making a few small changes to the Linux 2.4 kernel to add security hooks. The bulk of the HP-LX kernel functionality is provided by Dynamically Loaded Kernel Modules (DLKMs) that are called by the security hooks.

2.1 Containment

In HP-LX, every process has a compartment attribute – this is an additional field in the process table. Every time a process attempts to access a file or attempts to communicate with another process or network resource, the kernel checks the compartment attribute against a table of rules to see if the access is allowed. Similarly each time a process receives a message from another process or from the network the kernel checks the compartment attribute against a table of rules to see if the access is allowed.

We say that two processes that have the same compartment attribute are "in" the same compartment.

All child processes inherit the attribute value of their parent. So all processes are started in the same compartment as their parents.

2.1.1 Containing communication access

To control communication, the kernel maintains a Communication Control Table or CCT that controls what communication can take place between processes in different compartments. It also controls what communication can take place with network interfaces from a given compartment. Most communication mechanisms are supported including IP, shared memory, semaphores, and message queues). Communication between processes in the same compartment is not restricted by the CCT.

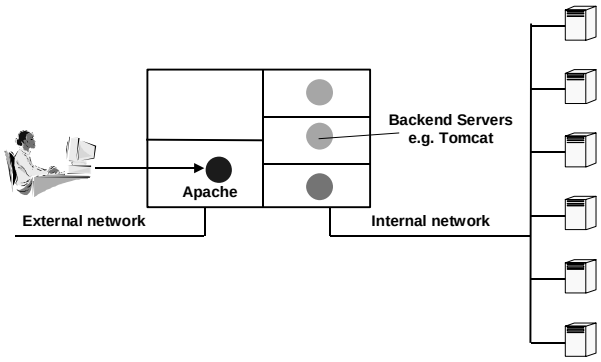

Figure 2: Typical HP-LX configuration

Figure 2 shows a typical HP-LX installation. It is often run as a dual-homed host providing a secure bridge between the external network (Internet) and internal network (Intranet). The HP-LX machine has a number of compartments in which various services are run.

In this example an Apache web server is run in a compartment called "Web". The following rules allow any process in the Web compartment to receive incoming tcp connections on ports 80 and 443 (i.e. http:// and https://)

```
HOST * -> COMPARTMENT web PORT 80
          METHOD tcp NETDEV lan_eth0

HOST * -> COMPARTMENT web PORT 443
          METHOD tcp NETDEV lan_eth0
```

The "`HOST *`" field indicates that the connection is permitted from any host. The "`NETDEV lan_eth0`" field indicates that the connection must be on the eth0 interface – that is the external network. The "`METHOD tcp`" field indicates that communication can only take place via TCP. Notice that there is no rule allowing an outbound connection. This means that the web server is unable to engage in any communication on the network except in direct response to an incoming TCP connection. So if the web server became compromised and was under the control of an attacker, it could not be used to probe the network or make connections to any other service on the network. This makes it harder to use HP-LX as a "trampoline" from which to attack other systems or to propagate worms through an HP-LX machine.

In this example there are three compartments for running Java servlet engines (Tomcat). These receive requests from the Apache web server and run Java servlets to access data from machines running on the internal network. Rules like the following are needed to allow the web server to talk to the servlet engines.

```
COMPARTMENT web -> COMPARTMENT tomcat1
                   PORT 8007 METHOD tcp
                   NETDEV lan_lo
```

This rule allows the web server to talk to the servlet engine in compartment Tomcat1 bound to port 8007 via “lo” or the local loopback. Notice that with the rules given so far the Tomcat1 compartment is not allowed to engage in any communication with the external network. The HP-LX host will ignore any packet arriving on eth0 (the Internet) for port 8007. In this respect the behaviour is similar to that of a firewall.

The Tomcat1compartment needs to be able to access the host test1 to retrieve data. This is accomplished by the following rule

```
COMPARTMENT tomcat1 -> HOST test1.foo.com
                       METHOD tcp PORT 8080
                       NETDEV lan_eth1
```

The interface eth1 is the internal network interface.

There are some similarities with the way communication is controlled in HP-LX and the Linux ipchains, netfilter and iptables facilities. The major differences are that we support communication mechanisms other than IP and that we have different rules that can be applied to different compartments. For example two servlet engines in different compartments may be able to communicate with different hosts on the internal network. The following rule allows the compartment Tomcat2 to access the host test2 (which is not accessible to compartment Tomcat1 by any of the above rules).

```
COMPARTMENT tomcat2 -> HOST test2.foo.com
                       METHOD tcp PORT 8080
                       NETDEV lan_eth1
```

2.1.2 Containing file access

In addition to the ‘r’, ‘w’, ‘x’ permission bits found in Linux, HP-LX also uses the compartment attribute to restrict what files a process can access. Each time a process opens a file its compartment attribute is checked against a set of rules in the File Control Table (FCT) to see if the access is allowed. The following are a set of sample rules for the Web compartment.

```
web /compt/web/apache/logs read,write,append
web /compt/web/dev         read,write
web /compt/web/tmp         read,write
web /compt/web             read
web /bin                   read
web /lib                   read
web /sbin                  read
web /usr                   read
web /                      none
```

Entries in the FCT consist of a compartment name, followed by a path and an access mode. The following access modes are supported: read, write, append and none. The most specific matching rule is enforced.

For example, if a process running in compartment Web attempted to access the following file: /compt/web/apache/htdocs/index.html
the following rule matches and specifies the access mode:

```
web /compt/web             read
```

If a process running in compartment Web attempts to access the file /etc/password, the following rule specificies the access mode:

```
web /                      none
```

This access mode is mandatory [TSEC] — it does not matter if the process is running as root or is the owner of the file, the kernel will still enforce the access mode specified in the FCT. So in this example, even if a process in the compartment Web is under the control of an attacker and becomes root, it cannot overwrite the index.html file and it can gain no access to the /etc/passwd file.

The rules in the FCT are mapped to inode numbers in the kernel. So any attempt to access the file by a different path (different mount path or a link) will still be subject to the control specified.

The FCT provides an extremely flexible mechanism for controlling access to files. It can restrict each compartment to a completely separate part of the file system, so that there are no shared files between compartments. However, it also allows files to be shared between compartments, with different compartments having different access modes. The following rule allows the compartment WebAdmin read and write access to the web server’s HTML files.

```
webAdmin /compt/web/apache/htdocs read,write
```

This means a user running processes in the WebAdmin compartment can update the web server’s pages.

HP-LX also supports the Linux chroot facility and is enhanced to make it more secure. The chroot facility provides a subset of the functionality of the FCT: it is used to contain processes within completely separate parts of the file system

We create a separate directory for each application to be run in a chroot environment. The files required by the application are then copied into the chroot

directory. Processes in one or more compartments can share the same chroot environment.

We also have the ability to "seal" a compartment. The kernel will not allow any process in a sealed compartment to run with the user identity root. In addition, it will not allow any process running in a sealed compartment to exec an SUID program. So strategies that might be used to escape chroot, such as running mknod (which requires you to be root), will not be available.

Chroot gives coarse grain access control to files. It is Mandatory Access Control [TSEC] because a process contained within a chroot environment cannot access files outside the chroot environment no matter who owns them

Our experience is that chroot on its own is not sufficient to provide containment. It is difficult and is sometimes not possible to chroot all applications. For example, it is not possible to chroot the backup process. Also it is convenient to have mandatory access control be more fine grain than is possible with chroot. For example, ensuring that the web pages visible to a web server are immutable. In addition it is sometimes more convenient to share access to files than to copy them into a chroot environment. We introduced the FCT to address all these issues.

If you have already engineered a chroot environment on your system for an application, you can use it on HP-LX to contain that application. You can also use the FCT to enhance the security of that chroot environment by doing things like granting read only access to the web pages.

One advantage that chroot has over the FCT is that it creates the illusion of a separate file system. For example, if we have two compartments Web and Tomcat, we create chroot environments under /compt/web and /compt/tomcat for each. We can then put the binaries needed for the Web compartment in /compt/web/bin and the binaries needed for the Tomcat compartment in /compt/tomcat/bin. A process running in a chroot environment under /compt/web will see the binaries in /compt/web/bin as if they were in /bin on a nomal system. So running the command "ls" from a shell in the chroot environment in the compartment Web runs the binary /compt/web/bin/ls. Similarly a process running in a chroot environment under /compt/tomcat will see the binaries in /compt/tomcat/bin exactly as if they were in /bin. So running the command ls from a shell in this chroot environment runs the binary /compt/tomcat/bin/ls.

In contrast to chroot, the FCT does not hide the parts of the file system that are not accessible to a compartment: the absolute paths are still visible.

We often use chroot and the FCT to completely isolate compartments into separate, non-overlapping parts of the file system. We use chroot to provide the illusion of a separate files system and then use the FCT to provide fine grain access control within the "separate" file system. This has the major benefit of allowing us to run multiple instances of the same service on the same machine: for example multiple web servers. Each web server is run in its own compartment in its own part of the file system. The CCT does not allow any communications between web server compartments. This completely isolates the web servers – they cannot interfere with each other via the file system or direct communication.

All the file system protection described above is implemented without any changes to the Linux file system – security attributes associated with a file are stored independently from the file. The reason that this is important is that it allows the files to be mounted by another system with no conflict of security attributes. This is to allow us to take advantage of shared-disk high availability technology. Further, this approach allows standard backup and restore utilities to be used.

2.2 File system integrity

HP-LX includes Tripwire [Tripwire]. This utility takes a cryptographic hash of all sensitive files. It informs administrators if they have been changed so that they can change them back again.

Tripwire provides additional protection against administration errors. If an administrator with the privilege to update a sensitive file updates that file, Tripwire will send an email message to them notifying them of this change. The administrator can then review the change to decide whether or not it should be reversed.

2.3 System configuration lockdown

System configuration lockdown in an operating system refers to creation and persistence of a secure or hardened configuration that eliminates non-essential services and secures the services left enabled.

Newly installed operating systems are often vulnerable to attack because the operating system's installation program generally installs all available packages and its initialisation program enables all

available services, daemons, and networking features. This allows an attacker to explore multiple avenues of attack. Every process that does not perform an essential function on the system thus constitutes a security risk. Moreover, file permissions on a newly installed operating system are generally set for high accessibility. As a result, the configuration of the system and its applications is vulnerable to unauthorized modification.

In HP-LX we take a minimalist approach. The only service installed on the system by default is the Secure Shell Daemon that is used for remote administration. In addition we have a lockdown script that performs many functions that are similar to those in Bastille [Bastille][1] — functions that are standard practice for Linux system configuration security:

- Disables console logins accept for root and administrator
- Defines limits on resources (CPU, disk space) for specific users and groups
- Unsets SUID/SGID bits from server executables
- Causes logs to be rotated weekly and compressed
- Sets password expiration to ninety days
- Restricts permissions on automated system tasks (cron jobs)
- Restricts permissions on a various executable programs

This approach ensures that HP-LX starts up with the most secure configuration by default. All extraneous applications and services must be explicitly installed and enabled, allowing administrators to achieve a well-considered balance between security and usability.

2.4 Audit

The architecture of the audit subsystem is shown in Figure 3. It is implemented as a DLKM that collects audit data by hooking the system call entry points and collecting the data that is passed into the kernel. The data that is collected by the kernel is pooled in a kernel buffer area and then later spooled to disk via a user space daemon.

[1] We had to implement our own lockdown feature, as Bastille was not ready. We hope to use Bastille in a future release.

Linux capabilities are used to protect the audit subsystem[2]. The audit subsystem checks for AUDIT_CONTROL whenever the subsystem is configured or managed (e.g. started, stopped, etc). The AUDIT_WRITE capability is needed by applications that want to add their own "application audit records". Additionally, the audit daemon runs with minimal capabilities. As it finishes with the capabilities that are needed to configure it, it removes them from the permitted set. It is finally left running with only 2 capabilities.

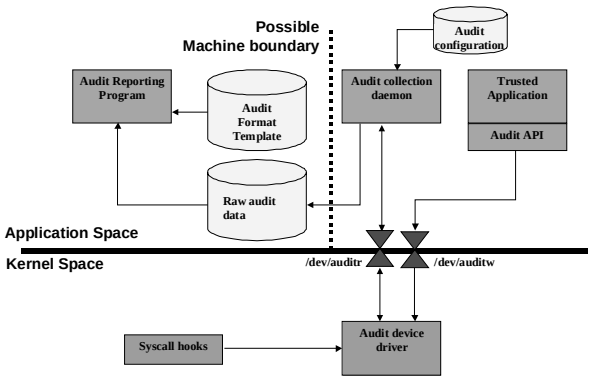

Figure 3: Audit subsytem architecture

The audit daemon's task is to move the audit data from the kernel buffer to storage (e.g. disk). It is notified by the kernel when to start writing the audit data. While writing this data, the kernel continues to collect audit data in a second audit buffer.

To protect the audit subsystem, including the integrity of the audit data we use compartments. The audit daemon is run in its own compartment and so is not accessible to services running in other compartments. In addition the audit configuration files and data are not accessible to services running in other compartments.

A filtering mechanism is used to help reduce the volume of data that is generated. This mechanism allows you to specify criteria to determine whether a particular event should be recorded. For instance, this filtering can be done based upon specific system calls or user IDs. The audit daemon loads the filtering information from its configuration file. The filtering

[2] Linux capabilities confer the ability to execute privileged system calls. The root user normally has all capabilities. By selectively removing capabilities from the set of active capabilities of a running process, the set of allowable operations is reduced.

information is an XML document. Here are two examples.

```
<syscall id="execve"/>
<syscall id="open">
   <uid action="equal">4</uid>
   <guid action="equal">20</guid>
</syscall>
```

Each <syscall> element may contain a number of child elements that restrict the occurrences of the system call that will be audited. The action attribute can take a value of “equal or “notequal”.

The first example instructs the audit system to audit all occurrences of the “execve” call. The second instructs the audit system to audit all occurrences of the “open” system call by user ID 4 and the group ID 20.

Two major issues for audit data are: how best to display the data to make it easy to understand; how to format it so other programs can process it. To address these issues, templates are used to specify the format of the data: XML, plain text, etc. Administrators are able to format their own templates giving them the freedom to view the data as they want.

A template specifies how each audit record will be printed by the auddump utility. Here is an example fragment of a template that we use to print the audit data as XML.

```
<event>\
  <header>\
    .
    .
    <eventid> &at_evtid;</eventid>\
    <date> &at_time;</date>\
    <pid> &at_pid;</pid>\
     .
     .
  </header>\
  <data>\
     .
     .
  </data>\
</event>\
\
```

The auddump utility applies this template to each audit record. The symbols beginning with “&” are replaced with the corresponding elements in the audit record. This generates output like the following.

```
<event>
 <header>
   .
   .
   <eventid>195</eventid>
   <date>Thu Jul 26 15:19:59 2001 BST</date>
   <pid>1</pid>
   .
   .
 </header>
 <data>
   .
   .
 </data>
</event>
```

HP-LX also has an audit API, to allow trusted applications to write their own audit data.

Auditing can generate a lot of data, so it is usual to limit the amount of disk space available to the audit system (e.g. have the records stored on a separate partition) and specify the behavior of the system when there is insufficient disk space to store more audit records. On HP-LX the audit daemon either writes to the system logging subsystem or it halts the system.

One important feature of the HP-LX auditing system is that by auditing the system calls rather than relying on application logs (e.g. web server logs) we do not rely on an application being honest about its behavior. This is important because a compromised process will almost certainly stop writing log data. In the case of some attacks the last application log entry may give some indication of the nature of the compromise, as is demonstrated in [Lestat01]. In other cases no applications level logs will be generated or they will be erased to hide the attack. It is much harder to avoid kernel level auditing. All processes need to make system calls and these system calls will be audited by the HP-LX audit subsystem. Preventing HP-LX auditing requires compromising the kernel. This is a difficult task, if the only attack points of the system are specific remote services (e.g. a web server). Thus HP-LX auditing will continue to record the behavior of a compromised process, even if it fails to write an application log. This can be used by an automated intrusion detection system to detect an attack in progress.

2.5 The secure administration model

In addition to the compartment attribute the HP-LX process table also contains a privilege bit field for each process. Currently we use only two bits: one is designated “hplx_admin” and the other is designated “set_comp”. These are the administration privileges for HP-LX.

The hplx_admin privilege allows processes to create new compartments, reconfigure existing compartments and change the rules in the CCT and FCT. The kernel will check that this bit is set before executing the appropriate system call.

The set_comp privilege allows a process to change its compartment. Again, the kernel checks that this bit is set before executing the appropriate system call.

In the kernel startup code (pre user space operation) these bits are cleared for all tasks (including kernel threads). They are set within the init kernel thread prior to it invoking the init user space program. Init is the first user space process and all other user space processes are descended from it.

A child inherits the bit values of its parent. By default init clears the hplx_admin bit and set_comp bit for all its child processes. The exceptions are if the entry in /etc/inittab explicitly tells init that the hplx_admin bit or set_comp bit are to be left set. Note that the file /etc/inittab is protected by the HP-LX file protection mechanisms.

It is important to note that we provide no method to set hplx_admin or set_comp after they have been cleared from user space. It is only possible to clear each bit or query its value from user space. Therefore a user space process can voluntarily give up one or both these privileges (never to get them back), but it can never gain a privilege it did not have when created.

In the default configuration init spawns two processes with both hplx_admin and set_comp set: getty for terminal 1 (console login) and an instance of the Secure Shell Daemon [SSH].

The console login is only usable by those that have physical access to the system. Any successful login to terminal 1 will have both the hplx_admin and set_comp bits set, irrespective of user identity. This is a convenience feature for those who have physical access to the hardware – we do not attempt to protect the machine from those that have physical access.

The Secure Shell (SSH) is used for remote administration. After a remote user has been successfully authenticated by SSH, a PAM (Pluggable Authentication Modules) module checks whether or not that user is an authorized administrator, if they are not authorized the hplx_admin and set_comp bits are cleared by the PAM module. Using PAM in this way avoids modifying SSH.

By default we do not permit remote login as root using SSH. We require users login under their real identity so we can audit their administration activity. Note that users need to su to root to perform administrative operations such as mount, as they would on an ordinary Linux system.

The mechanisms described in this section work independently of the "root" privilege and Linux capabilities. HP-LX still requires the root privilege to perform administrative operations only available to root on Linux. The set_comp and hplx_admin privileges are additional privileges introduced and used only for managing the security configuration of HP-LX. We choose not to use Linux capabilities because we wanted a well-defined management model for controlling who gets the privilege as described above. This means we can audit which users are performing administrative actions on the system, because we will have their real user identities. In addition, it enables us to easily control which processes administrative users can use to enter the system. Finally, it was not obvious which of the existing capabilities in Linux we could use without changing the semantics associated with that capability. The three privileges of "hplx_admin", "set_comp" and "root" constitute a split privilege model, but the granularity is much coarser than is found in many implementations. For example there are currently 29 capabilities in Linux and in some systems there are over 70 privileges [Sun]. By choosing a coarse granularity we have chosen to trade flexibility for simplicity.

3. The typical default configuration

Figure 4 shows the typical default compartment configuration for HP-LX with a web server (Apache) and Tomcat engine installed.

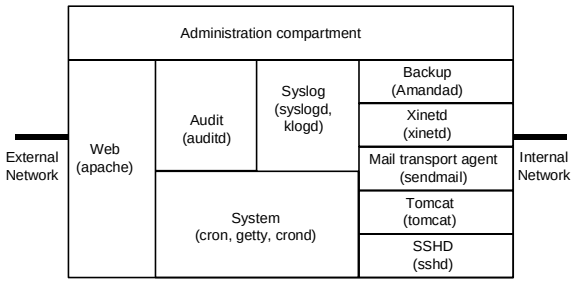

Figure 4: Typical compartment configuration

The administration compartment is a unique compartment on HP-LX. This is not subject to the HP-LX communication and file access controls. Processes running in this compartment are not able to distinguish HP-LX from an ordinary Linux system:

they can access any network, any file (provided permission bits allow it) and can send messages to any process in any other compartment.

The primary use of the administration compartment is for system diagnosis and administration. It is not recommended that network services be run in this compartment – any service run in this compartment will not derive benefit from the security mechanisms in HP-LX.

Recall that HP-LX is often installed as a dual-homed host providing a secure application-level gateway between the Internet and Intranet. In this configuration, with the exception of the administration compartment, the only compartment that has access to the external network or Internet is the Web compartment.

The following compartments have access to the internal network or Intranet

- Backup – so that backups can be made and sent to remote backup servers
- Xinetd – runs xinetd which is used to start the backup daemon (amandad)
- Mail transport agent – for sending outbound email (certain services need this, for example, Tripwire needs to be able to email reports of file integrity problems to an administrator) – HP-LX will not accept incoming email in its default configuration
- Tomcat – runs the java servlet engine Tomcat to connect to and retrieve data from hosts on the Intranet.
- SSHD – runs the Secure Shell Daemon to allow remote logins for secure administration.

The Audit, Syslog and System compartments are used to run various services (see diagram) that do not require direct network connections.

4. The tension between security and usability

It is a well-accepted and good security principle that you should never have "unnecessary" software on a host [Pipkin97]. The idea is that an attacker could abuse this software. In the past this has led to "hardened" versions of operating systems being shipped to customers with much less software being installed than on the default "non-hardened" system.

Unfortunately predicting what is "unnecessary" is very hard. Different customers require different application sets. Support activities such as diagnosing a fault often requires additional software not used in normal operation of the host (e.g. debuggers). So what software is necessary will change during the lifecycle of the system.

In HP-LX we take a different approach. When the system is initially installed it has only a few minimal services running as described above. However, after initial installation and configuration, any software available in the Red Hat 7.1 Linux distribution can be installed and run on HP-LX. We rely on the HP-LX security mechanisms to prevent the software from being used to execute an attack.

The administration compartment is used for running software such as ping, traceroute and gdb. In this compartment ping and traceroute behave as they would on any normal Linux installation. The debugger, gdb can be attached to any process in any other compartment. If you tried to run the debugger in a compartment other than the administration compartment, it would be allowed only to attach to processes in its own compartment.

5. Related work

HP-LX draws heavily on our own experience with HP Virtualvault [Virtualvault] – a secure web server appliance. This system is based on an evolution of the Bell and La Padula model [Bell76] and its refinement in [TSEC]. The basic concept in [TSEC] is that a system is divided into a number of hierachical classification levels (unclassified, secret, top secret, etc) and non-hierarchical categories (project1, project2). Every entity (processes and files) in the system is assigned a sensitivity label consisting of its classification and categories (an entity can only have a single classification, but can be in multiple categories). The classifications and categories in a system form a security lattice. [TSEC] specifies the information flows allowed within the lattice: information can flow up the lattice, but not down or across it. In Virtualvault we used the security lattice to separate the machine into 4 separate regions as shown in Figure 5.

The web pages and any immutable data is stored with sensitivity label S. It is at the bottom of the lattice, so it can be read, but not written by processes in other parts of the lattice. The web server is assigned the label SO, as is the external network interface. The internal network interface and the processes used to access the internal network are assigned the label SI. The label SIO is used to store data that cannot be read by any other processes in other parts of the lattice (although they can write it). This is used to

store audit data. Since there is no relationship between SI and SO, no information can flow between them. This is a problem, because we need to get data from the web server to the internal network and back again in a controlled, secure fashion. The information flows permitted by the lattice are too restrictive to build a useful system.

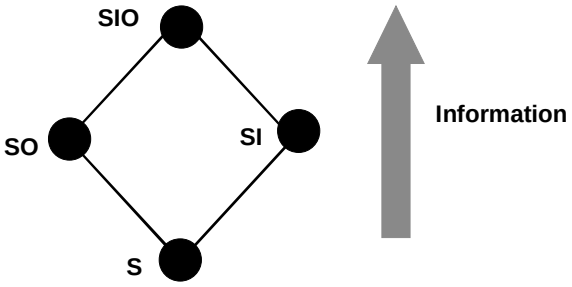

Figure 5: The security lattice in Virtualvault

To address this problem a number of privileges are introduced that a process can possess to allow it to override the information flows permitted by the lattice security model. To relay information across the lattice, privileged processes are used. These privileged processes are called "Trusted Processes" – this is an appropriate term as the security of the whole system depends on them not being compromised and abusing their privileges. Writing a Trusted Process is a difficult and time-consuming task – you need a deep understanding of the model in [TSEC] and the use of privileges.

In HP-LX we carried forward the fundamental idea of separating the machine into a number of regions – we call these compartments. However, we do not have any implicit information flows. Instead we require administrators to configure the communication patterns explicitly using metaphors that should already be familiar to them (similar to the idea of configuring a packet filter). Instead of assigning files into compartments only processes are assigned to compartment and we require administrators to make explicit what files should be available to particular compartments and what the access mode should be.

Domain and Type Enforcement is a more recent model [O'Brien91, Badger96]. The basic idea in type enforcement is that each active entity (e.g. process) is assigned a domain and each passive entity (e.g. file is assigned a type). A Domain Definition Table (DDT) specifies the interactions allowed between a given domain and type. A Domain Interaction Table (DIT) specifies the interactions allowed between entities in different domains. Badger et al. reported that this approach has a number of problems [Badger96]: the configuration of the tables becomes very complex; the tabular structure does not map well to the hierarchical structure of a file system; designing the type enforcement policies is complicated and it is hard to reuse them. Fundamentally these problems concern difficulties in mapping the model to the abstractions used by the underlying operating system. [Badger96] proposed overcoming these problems by introducing a high-level policy language that can be used to specify the information held in the DDT and DIT. Our approach differs in that our model is built around the underlying abstractions used by the operating system (files, directories and ports). So our model fits well with the underlying operating system abstractions and the metaphors used to configure it are familiar to system administrators.

PitBull LX [Argus] uses the concept of domains to label system resources. A dominance and intersect relationship is defined on domain sets. Whenever a process attempts to access a file the process's domain set must dominate the file's domain set for the access to be allowed. Similarly a process's domain set must intersect the network's domain set for a network access to succeed. Thus this model creates implicit information flows based on abstract domain set relationships that are not part of the underlying operating system. Our philosophy is that it is simpler to make the information flows explicit using the abstractions of the underlying operating system.

Linux Intrusion Detection System or LIDS [LIDS, Chaubal01] has several features to improve the security of Linux: controlling kill signals, protection of sensitive parts of the kernel and file system protection. The file system protection mechanisms have some similarities with those in HP-LX. LIDS can specify access controls that apply to everybody including root: so even root can be prevented from modifying a file. LIDS can specify access controls that apply to particular executables – so an executable can be restricted to a predefined set of files and directories. The key differences between HP-LX and LIDS are that LIDS does not have a notion of compartments and does not provide control of network access. Compartments provide us with a convenient way of grouping processes together, so that they are contained together and subject to the same controls.

Immunix [Immunix] has two main features that seek to contain applications and services in a similar way to HP-LX. Immunix provides protection against two common classes of bugs that are exploited by

attackers to compromise and take over applications and services. The HP-LX containment mechanisms attempt to protect against arbitrary vulnerabilities (not just specific classes). In addition the HP-LX containment mechanisms provide some protection against application mis-configuration. Immunix also provides a file system protection mechanism called SubDomain. This is very similar to the HP-LX FCT mechanism: it allows the administrator to list the files a program may access, and the operations the program may perform on those files. The major difference between the HP-LX FCT and SubDomain is that in HP-LX we specify the file access per compartment, rather than per process. Using compartments in this way makes it easy to apply controls to groups of processes. HP-LX also provides extensive control on inter-process communication and network use. Immunix includes no specific features to control communication.

The focus of the Security Enhanced Linux [Loscocco01] project is to develop the underlying mechanisms to support a wide range of robust security models into the Linux kernel. The architecture is based on Type Enforcement, but also provides support for Multi-Level security [TSEC]. Both these models have been discussed above. The focus of our work has been to develop a new security model that improves on the usability of earlier security models.

6. Issues and further work

HP-LX should be seen as part of a wider security strategy. It provides a robust platform to protect you against compromises of your applications and services. However, it cannot protect you against security errors inside your applications. For example, suppose a user successfully authenticates to an application and then (due to an error in the application-level access control) is able to wrongly access data inside the application. HP-LX cannot protect you against this error. Thus your wider security strategy needs to consider the security model inherent in your application and any other elements that might be part of the interaction between the client and the service, such as the secure configuration of packet filters or firewalls.

Two issues that we have not addressed in the current version of HP-LX are NFS and distributed containment. Support for NFS will allow us to provide secure access to remote files. Distributed containment will allow us to preserve compartment attributes across multiple HP-LX machines. This requires using a secure networking technology such as IPSEC to authenticate and preserve the compartment attributes on a remote machine.

7. Summary and conclusion

HP-LX provides a secure environment to protect you against compromises of applications and services by bugs or mis-configuration. The key idea is containment – to contain a service to a known part of the machine and to contain its ability to communicate with other entities both on the machine and remote. This prevents processes probing the network (to propagate worms) and overwriting sensitive files (defacing web pages).

The mechanism we introduced to implement containment is based on compartments. Each process has a compartment attribute associated with it. Every time a process attempts to communicate or access a file the kernel examines the compartment attribute and checks the File Control Table or Communication Control Table to see if the action is allowed.

This security model was developed using our experience of deploying an existing security model developed for military use [TSEC]. An important objective was to make the new security model much easier to use. We do this by requiring administrators to explicitly identify files and communication endpoints using abstractions that are already available in the operating system. This is in contrast to requiring the administrator to work within the confines of an abstract security model that may not fit well with the abstractions of the underlying operating system.

Availability

Further details including availability of HP Secure OS software for Linux (HP-LX) are available at:
http://www.hp.com/security/products/linux/

Acknowledgements

The authors are grateful to the whole of the HP secure Linux team and in particular for comments from: Chris Dalton, Craig Rubin and Scott Ruff. We are also grateful for comments and feedback from Zach Brown of OSDL.

References

[Argus] “PitBull LX OS-Level Security for Solaris and Linux”, Argus Systems Group, May 2001, http://www.argus-systems.com/

[Attrition] http://www.attrition.org/mirror/attrition/os.html

[Badger96] A Domain and Type Enforcement Unix Prototype, L. Badger, D.F. Sterne, D.L. Sherman, K.M. Walker, The Usenix Association, Computing Systems, 9(1), 1996, pp46-83.

[Bastille] http://www.bastille-linux.org/

[Bell76] "Secure Computer System: Unified Exposition and Multics Interpretation", D.E. Bell, L.J. La Padula, Mitre Report, MTR-2997 Rev. 1, March 1976.

[Chaubal01] "LIDS and Mandatory Access Control (MAC) on Linux", Ameet Chaubal, UnixReview.com, http://unixreview.com/articles/2001/0106/0106m/0106m.htm

[Forrestal01] Vulnerability Assessment Scanners, Jeff Forrestal and Greg Shipley, Network Computing (http://www.networkcomputing.com/), January 8, 2001.

[Immunix] http://www.immunix.org

[Lemos01] "Software "fixes" routinely available but often ignored", Robert Lemos, ZDNet News, 24th January 2001, http://news.zdnet.co.uk/story/0,,s2083937,00.html

[Lestat01] "The Ramen Worm and its use of rpc.statd, wu-ftpd and LPRng Vulnerabilities in Red Hat Linux", Morgan Lestat, February 7, 2001, http://www.sans.org/infosecFAQ/malicious/ramen.htm

[LIDS] http://www.lids.org

[Loscocco01] Integrating Flexible Support for Security Policies into the Linux Operating System, P. Loscocco, S. Smalley, Jan 2, 2001. http://www.nsa.gov/selinux/

[O'Brien91] Developing Applications on LOCK, R. O'Brien and C. Rogers, in Proc. 14th National Computer Security Conference, pages 147--156, Washington, DC, October 1991.

[Pipkin97] "Halting the Hacker: A Practical Guide To Computer Security", Donald L. Pipkin, Prentice-Hall, 1997.

[SSH] http://www.openssh.org/

[Sun] "Trusted Solaris[tm] 8 Operating Environment – A Technical Overview", http://www.sun.com/

[TSEC] Department of Defense Trusted Computer System Evaluation Criteria, DoD 5200.28-STD, December, 1985.

[Tripwire] http://www.tripwire.org/

[Virtualvault] "HP Virtualvault concepts Guide", HP Part No. B5413-90051, Hewlett-Packard Co, 1999.