Access Control

Jackson Argo

Rackspace MO After-hours

April 27, 2016

What is Access Control?

How Is Access Determined?

Who Determines Access?

Forms of Access Control
- SELinux Booleans
- iptables
- File Access Control Lists
- Apache Access Control
- Linux Groups

Access Control in the Kernel
- Namespaces
- Control Groups
- Containers

What is Access Control?

- Access control is answering the question: *Who can access what?*
- Our every day life relies on access control in many ways:
  - We choose which thoughts that we want to share.
  - We carry an id so that we can buy alcohol.
  - The 2nd amendment guarantees us access to firearms.

Access Control in Linux

Unsurprisingly, Linux relies on access control as well.

- The kernel protects memory space so that programs can't inhibit one another.
- Regular users cannot arbitrarily overwrite the root filesystem.
- Apache blocks access to xmlrpc.php so people can run Wordpress sites.

How Is Access Determined?

Before we can implement access control, we need to have a reliable way to determine whether access can be permitted. Valid access control requires the following:

- There must be a non-biased mechanism used to test for access.
- The mechanism must be sure that the user cannot be spoofed or impersonated.
- The user must be able to activate this mechanism.
- The user must be sure that the mechanism cannot be spoofed or impersonated.
- The result must be enforced by the operating system.

Trust

Like all of security, access control relies on trust. If you use a mechanism for access control, then you inherently trust that the mechanism will work properly.

Who Determines Access?

Someone has to determine access control and security of different objects, whether it is an automated program or a user, i.e. *Who has access to control access control?*

There are two main approaches:

- **Discretionary** - Any user may be involved in the definition on the policy function and/or assignment of security attributes. This places the burden of security on the user. *E.x. file permissions, Facebook posts, 5^{th} amendment.*
- **Mandatory** - Policy functions and security attributes are tightly controlled by the system administrator. This places the burden of security on the administrator. *E.x. SELinux, firewalls, sudoers.*

Discretionary Access Control Gotchas

- I want to re-emphasize that **discretionary access control places the burden of security on the user.**
- This means that our customers need to have a good understanding of what needs to be protected and what doesn't.
- Our customers *are* our customers because they do not necessarily have a good understanding of this.
- It follows that **WE** have to be extra vigilant in checking that things like file permissions are sane and will not compromise the server.
- We should also educate our customers when they are doing something wrong.

Forms of Access Control

In Linux, the mechanism for access control typically checks the user against predetermined set of criteria.

These criteria look like:

- Simple Yes/No Rules
- Access Control Lists (ACL's)
- Role Based Access Control (RBAC)

Forms of Access Control

Here are some common ways we see access control in daily life that are not particularly common in Linux:

- Attribute Based Access Control (ABAC) - This is a more general form of RBAC. Instead of checking for only a list of roles, you can check any key attribute for any value.
- Time Based Access Control (TBAC) - Access is only granted during specific times, e.x. department store hours of operation.
- History Based Access Control (HBAC) - Access is granted based on a history of activities. fail2ban uses this type of access control.

Simple Yes/No Rules

- Yes/No/Pass/Fail rules are the simplest way to determine access, and all forms of access control can be generalized to these rules.
- Examples:
  - `[ $(id -u) = 0 ] || exit 1`
  - SELinux booleans

Example: SELinux Booleans

```
# curl -sI localhost | awk /HTTP/
HTTP/1.1 403 Forbidden
# tail -n1 /var/log/httpd/error_log
... (13)Permission denied...
# ls -lZ /var/www/html/index.html
-rwxr-xr-x. root root system_u:object_r:nfs_t:s0
    /var/www/html/index.html
# awk '/var\/www/ { print $1, $2, $3 }' /etc/mtab
127.0.0.1:/srv/nfs/www/html /var/www/html nfs4
# getsebool -a | awk '/httpd/ && /nfs/'
httpd_use_nfs --> off
# setsebool -P httpd_use_nfs on
# getsebool -a | awk '/httpd_use_nfs/'
httpd_use_nfs --> on
# curl -sI localhost | awk /HTTP/
HTTP/1.1 200 OK
```

Access Control Lists

- ACL's can be thought of as a chain or flow chart of standardized yes/no rules. The agent will be checked against all the rules until the chain is terminated by a final yes or no.
- Examples:
  - iptables rules
  - File ACLS
  - Apache access control

iptables

First, we define our iptables chains:

```
*filter
:INPUT ACCEPT [0:0] # Default input chain
:FORWARD ACCEPT [0:0] # Default forward chain
:OUTPUT ACCEPT [0:0] # Default output chain
:IN_PRIVATE - [0:0] # Input from cloud network
:IN_PUBLIC - [0:0] # Input from the public interface
:IN_SERVICENET - [0:0] # Input from servicenet
```

iptables

Next, we define the rules for each chain:

```
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -i eth0 -j IN_PUBLIC
-A INPUT -i eth1 -j IN_SERVICENET
-A INPUT -i eth2 -j IN_PRIVATE
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
-A IN_PRIVATE -j ACCEPT
-A IN_PUBLIC -p tcp -m tcp --dport 22 -j ACCEPT
-A IN_SERVICENET -p tcp -m tcp --dport 22 -j ACCEPT
-A IN_SERVICENET -p tcp -m tcp --dport 80 -j ACCEPT
```

iptables

Now we can check our firewall settings. I ran these commands from bastion:

```
$ head -n0 < /dev/tcp/104.239.175.131/22
$ head -n0 < /dev/tcp/104.239.175.131/80
-bash: connect: No route to host
-bash: /dev/tcp/104.239.175.131/80: No route to host
$ head -n0 < /dev/tcp/10.209.66.22/22
$ head -n0 < /dev/tcp/10.209.66.22/80
$ head -n0 < /dev/tcp/10.209.66.22/111
-bash: connect: No route to host
-bash: /dev/tcp/10.209.66.22/111: No route to host
```

File Access Control Lists

```
# cd /var/www/html
# curl -s https://wordpress.org/latest.tar.gz | tar xz
# curl -sI localhost/wordpress/ | awk /HTTP/
HTTP/1.1 200 OK
# ls -l wordpress/index.php
-rw-r--r--. 1 nobody nfsnobody 418 Sep 24 2013
    wordpress/index.php
# chown -R jpeaches:jpeaches wordpress/
# find wordpress/ -type d -exec chmod 700 {} \+
# find wordpress/ -type f -exec chmod 600 {} \+
# curl -sI localhost/wordpress/ | awk /HTTP/
HTTP/1.1 403 Forbidden
# setfacl -R -m u:apache:rX wordpress/
# setfacl -R -m default:u:apache:rX wordpress/
# curl -sI localhost/wordpress/ | awk /HTTP/
HTTP/1.1 200 OK
```

File Access Control Lists

```
# sudo -u wpftp touch wordpress/index.php
touch: cannot touch 'wordpress/index.php': Permission denied
# setfacl -R -m u:wpftp:rwX wordpress/
# setfacl -R -m default:u:wpftp:rwX wordpress/
# sudo -u wpftp touch wordpress/index.php
# ls -l wordpress/index.php
-rw-rw----+ 1 jpeaches jpeaches 418 Apr 18 04:06
    wordpress/index.php
# sudo -u wpftp touch wordpress/newfile
# getfacl wordpress/newfile | awk "/^user:(apache|wpftp)/"
user:apache:r-x #effective:r--
user:wpftp:rwx #effective:rw-
# sudo -u jpeaches touch wordpress/newfile
touch: cannot touch 'wordpress/newfile': Permission denied
# setfacl -R -m u:jpeaches:rwX wordpress/
# setfacl -R -m default:u:jpeaches:rwX wordpress/
# sudo -u jpeaches touch wordpress/newfile
```

File Access Control Lists

```
# getfacl wordpress | awk '!/^(#|$)/'
user::rwx
user:apache:r-x
user:jpeaches:rwx
user:wpftp:rwx
group::---
mask::rwx
other::---
default:user::rwx
default:user:apache:r-x
default:user:jpeaches:rwx
default:user:wpftp:rwx
default:group::---
default:mask::rwx
default:other::---
```

Apache Access Control

```
AuthType Basic
AuthName "Restricted Access"
AuthUserFile "/etc/httpd/passwd/passwords"
<RequireAll>
  Require user jpeaches
  <RequireAny>
    Require ip ::1 127.0.0.1
    Require host localhost
  </RequireAny>
</RequireAll>
```

Apache Access Control

```
# mkdir /etc/httpd/passwd/
# chown apache:apache /etc/httpd/passwd/
# chmod 700 /etc/httpd/passwd/
# awk -F: -v OFS=: '/^jpeaches/ { print $1, $2 }' /etc/shadow >
    /etc/httpd/passwd/passwords
# htpasswd -b /etc/httpd/passwd/passwords wpftp ILovePeaches
# chown apache:apache /etc/httpd/passwd/passwords
# chmod 600 /etc/httpd/passwd/passwords
```

Apache Access Control

```
# curl -sI localhost/wordpress/ -u jpeaches:ILovePeaches | awk
    /HTTP/
HTTP/1.1 200 OK
# curl -sI 127.0.0.1/wordpress/ -u jpeaches:ILovePeaches | awk
    /HTTP/
HTTP/1.1 200 OK
# curl -sg6 -I [::1]/wordpress/ -u jpeaches:ILovePeaches | awk
    /HTTP/
HTTP/1.1 200 OK
# curl -sI 104.239.175.131/wordpress/ -u jpeaches:ILovePeaches |
    awk /HTTP/
HTTP/1.1 403 Forbidden
```

Apache Access Control

```
<Directory "/www/mydocs">
    <RequireAll>
        <RequireAny>
            Require user superadmin
            <RequireAll>
                Require group admins
                Require ldap-group cn=Administrators,o=Airius
                <RequireAny>
                    Require group sales
                    Require ldap-attribute dept="sales"
                </RequireAny>
            </RequireAll>
        </RequireAny>
        <RequireNone>
            Require group temps
            Require ldap-group cn=Temporary Employees,o=Airius
        </RequireNone>
    </RequireAll>
</Directory>
```

Role Based Access Control

Users are assigned roles, and objects have particular roles that are allowed to access them. These roles are often hierarchical, e.x. full admin $>$ network admin $>$ firewall admin.
Examples:

- Linux groups
- Rackspace Cloud RBAC
- Rackspace internal roles

Linux Groups

When used in conjunction with sudo, Linux groups are a nice user abstraction that can be extended beyond simple file level access control.
We can use /etc/sudoers to create a *firewall-admin* group that can:

- Add and remove entries to /etc/hosts.
- Check what process are listening on open ports.
- Manage network interfaces and routes.
- View/update the firewall.

Linux Groups

Here is our sudoers file:

```
Cmnd_Alias FIREWALLADM = /usr/sbin/ip, /usr/sbin/iptables,
    /usr/bin/netstat
Defaults   always_set_home
Defaults   env_reset
Defaults   secure_path = /sbin:/bin:/usr/sbin:/usr/bin
root  ALL=(ALL)  ALL
%firewall-admin ALL = (root) NOPASSWD: FIREWALLADM
```

Linux Groups

Now we put the *firewall-admin* group to use:

```
# groupadd
# gpasswd -a jpeaches firewall-admin
# su jpeaches
$ iptables -L
iptables v1.4.21: can't initialize iptables table 'filter':
    Permission denied (you must be root)
Perhaps iptables or your kernel needs to be upgraded.
$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target    prot opt source               destination
ACCEPT    all  --  anywhere             anywhere             ctstate
    RELATED,ESTABLISHED
$ sudo echo hello
[sudo] password for jpeaches:
Sorry, user jpeaches is not allowed to execute '/bin/echo hello'
    as root on jack8684-acbrownbag.
```

Access Control in the Kernel

In each form of access control we've discussed so far, the kernel is the decider and enforcer. The kernel ultimately decides who has has access to what, and it follows that we should be able to completely isolate a process's access to objects through the kernel alone. There are two kernel features that give us this power:

- **Namespaces** - Isolates a process's virtual resources.
- **Control Groups** - Restricts access to hardware resources.

Namespaces

- Think of namespaces as an abstraction of *chroots*. Instead of just files, we can restrict network interfaces, pid's, users, and more.
- A namespace wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource.
- Changes to the global resource are visible to other processes that are members of the namespace, but are invisible to other processes.
- One use of namespaces is to implement containers.

Types of Namespaces

There are 6 types of namespaces:

- **Mount** - Essentially a much stronger form of the classic chroot.
- **Network** - The namespace gets it's own virtual interface, and cannot see any other interface. Processes can attach to any ports on that interface without interfering with the portmapping outside the namespace.
- **PID** - The namespace gets it's own process list. The first process in the namespace will have pid 1.

Types of Namespaces

- **User** - Similarly, the namespace gets it's own user list. A user inside the namespace can have the same uid as another user outside the namespace.
- **UTS** - Let's you set hostname and domain name for a namespace.
- **IPC (Interprocess Communication Mechanisms)** - Isolates message queues, semaphore sets, and shared memory segments.

Control Groups

- Think of control groups as an abstraction on *nice*. Instead of only limiting CPU priority, we put limits on pretty much every way a process will interact with the hardware.
- There is a **LOT** of control we as system administrators have over cgroups, and I could dedicate an entire brown bag on how these work and how we can use them.
- Fedora has some pretty thorough documentation on this in their Resource Management Guide.

Types of Control Groups

- **blkio** — this subsystem sets limits on input/output access to and from block devices such as physical drives (disk, solid state, USB, etc.).
- **cpu** — this subsystem uses the scheduler to provide cgroup tasks access to the CPU.
- **cpuacct** — this subsystem generates automatic reports on CPU resources used by tasks in a cgroup.
- **cpuset** — this subsystem assigns individual CPU's (on a multi-core system) and memory nodes to tasks in a cgroup.
- **devices** — this subsystem allows or denies access to devices by tasks in a cgroup.

Types of Control Groups

- **freezer** — this subsystem suspends or resumes tasks in a cgroup.
- **memory** — this subsystem sets limits on memory use by tasks in a cgroup, and generates automatic reports on memory resources used by those tasks.
- **net_cls** — this subsystem tags network packets with a class identifier (classid) that allows the Linux traffic controller (tc) to identify packets originating from a particular cgroup task.
- **net_prio** — this subsystem provides a way to dynamically set the priority of network traffic per network interface.
- **ns** — the namespace subsystem.

Containers

- We have all of these cool access control tools like namespaces, control groups, and SELinux policies, but how to we leverage them together?
- Linux Containers burrito all of these features to give us very strong process isolation (https://linuxcontainers.org/).
- This is the backbone of our favorite tool, Docker.

END

Questions? Comments? Concerns? Food?

Contribute!