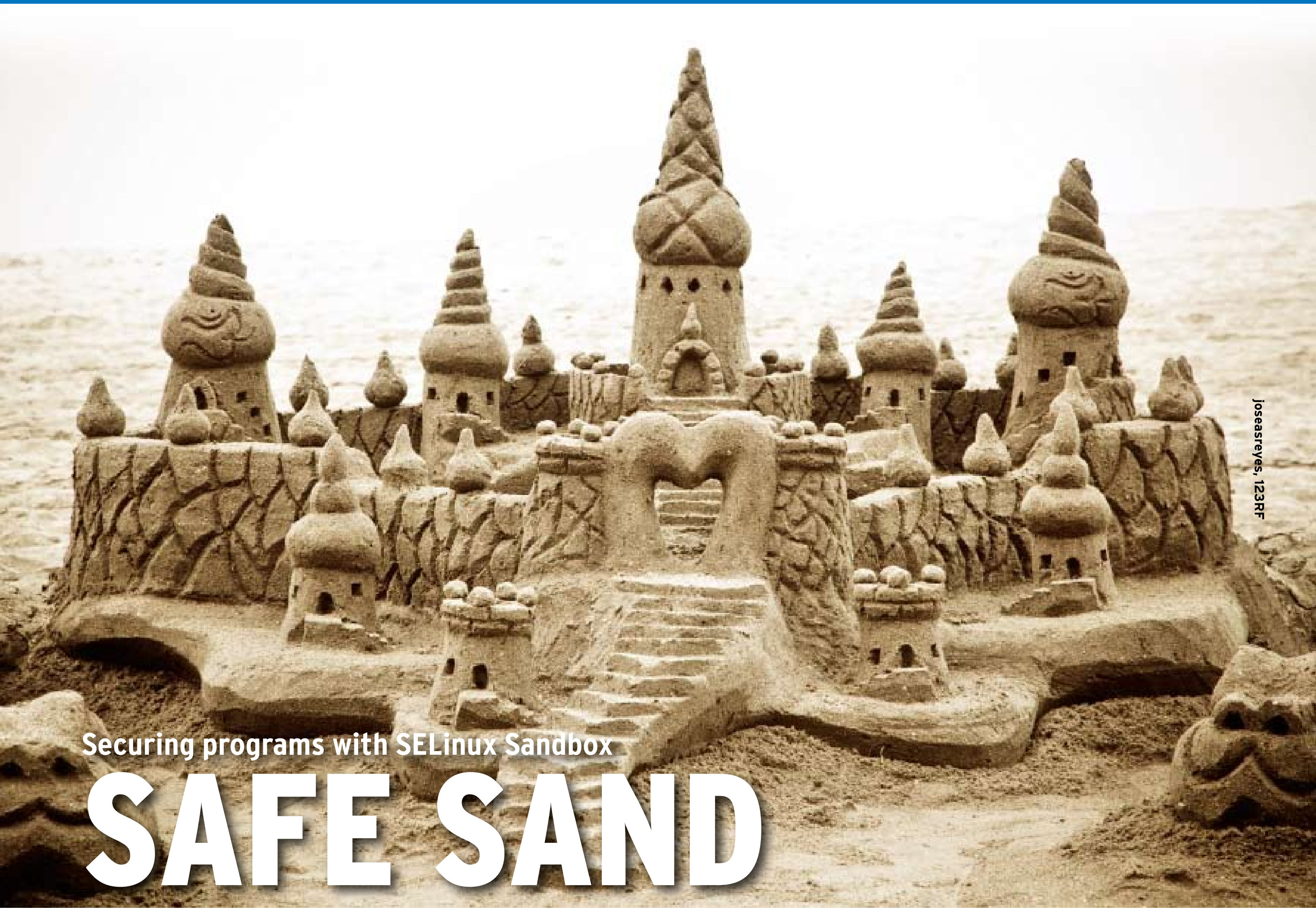


Securing programs with SELinux Sandbox

SAFE SAND

It is hard to keep your browser away from untrusted web scripts and applications. The SELinux Sandbox locks those untrusted apps into a safe place where they can't do mischief. **BY THORSTEN SCHERF**

Firefox is commonly regarded as very secure, but popularity is no protection against software bugs. In July 2009, the Mozilla Foundation disclosed several bugs in Firefox 3.5, including an error in the Just-in-time (JIT) JavaScript compiler. If a user surfed to a website with the exploit code using a vulnerable version of the browser, an attacker could run arbitrary code on the client system, which poses a huge risk for users who store credit card numbers, passwords, and other sensitive information in their browser.

The SELinux [1] mandatory access control system is designed to prevent an intruder who has gained a foothold on the system from escalating privileges. The original intent of SELinux was to protect system services, but some time ago, the SELinux developers team set their sights on protecting the web browser and other desktop applications. The goal is to provide an isolated *sandbox* to let the browser execute untrusted code in a more secure manner. But a browser-oriented SELinux ruleset is not as easy as it sounds. For example, most applications on a Gnome desktop want to have complete control of the user's home directory. Also, the application needs to communicate with the computer's messaging system (D-Bus), and the X server on which Firefox is running needs access to the *tmp* directory. One of SELinux's mainstays, the least privilege principle, is thus difficult to implement because of the complex relationships of the interdependent components.

The SELinux developer team at Red Hat decided to create a kind of virtual cage in which you can lock up graphical applications such as Firefox. The result-

Listing 1: Excerpt from sandbox.if

```
type $1_t, sandbox_x_domain;
domain_type($1_t)

type $1_file_t, sandbox_file_type;
files_type($1_file_t)

type $1_client_t, sandbox_x_domain;
domain_type($1_client_t)

can_exec($1_client_t, $1_file_t)
...
```

Listing 2: Opening Prohibited

```
# sesearch --allow -s sandbox_t -t etc_t -c file
Found 2 semantic av rules:
   allow sandbox_domain etc_t : file { read write getattr lock append } ;
   allow sandbox_domain file_type : file entrypoint ;
```

ing solution is both intelligent and elegant. Instead of allowing the application direct access to *tmp* and the user's home directory, the sandbox utility presents the application with different folders mapped to the required targets.

To lock an application away in a jail, you just enter the following:

```
sandbox -X evince
```

How It Works

The sandbox tool creates a virtual cage made up of two new directories: one in the user's home directory and the other in *tmp*. Each folder is assigned its own SELinux context. Different applications are placed in different jails, making it impossible for, say, the PDF viewer running in one sandbox to access the web browser's password file, which is running in a different sandbox. The *seunshare* module then bind-mounts the new directories. Each time the application launches, it sees new, and thus empty, home and *tmp* directories. *seunshare* then launches the user's shell, again using the sandbox's SELinux context, *sandbox_x_t:MCS*, and a random MCS (Multi-Category Security) label.

Client applications inside the jail are assigned the *sandbox_x_client_t* label. A highly granular policy exists for this domain to specify what access is permitted – the sandbox policy module (e.g., the Evince PDF viewer will need an X server and a Window manager). The X server can't access the host system. The application itself runs in full-screen mode in the sandbox's dedicated window manager, and this prevents access to the window manager's other functions. Access to the network is also initially impossible from inside the sandbox. The files have to be labeled *sandbox_x_file_t*; otherwise, the policy will prevent access to them (Figures 1 and 2).

To allow an application to access the network from within the sandbox, you can launch it with a different label, instead of the default *sandbox_x_client_t*. For example, to allow Firefox to visit the outside world, you could launch it with the label *sandbox_web_t*:

```
sandbox -X -t sandbox_web_t firefox
```

The *sandbox_net_t* label permits unrestricted network access (see Figure 3), but some caution is advisable. The idea of a sandbox is to support granular access, and if an application running with the *sandbox_net_t* label happens to be buggy, the application could transport sensitive user or system data out of the sandbox, despite the precautions. For this reason, each application should only access the network protocol that it needs to work properly.

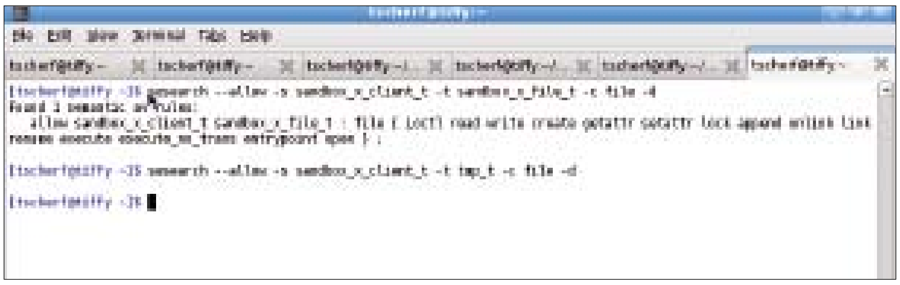
The SELinux developers provide a number of macros to help you do this, and these macros are really useful for developing custom policies. To create a ruleset for arbitrary applications, you can use *sandbox_x_domain_template*. For example, if you want an application to use *smtp* from within the sandbox, you can call the macro inside your own policy, as follows:

```
sandbox_x_domain_template⤦
    (sandbox_mail)
```

A glance at the macro definition source code (Listing 1) quickly reveals what happens when you call the macro. On accessing the macro, you receive a basic framework for a new SELinux domain in which the application will run. If you then add your own rules, you can call *semodule* to add the newly compiled policy module to the system.

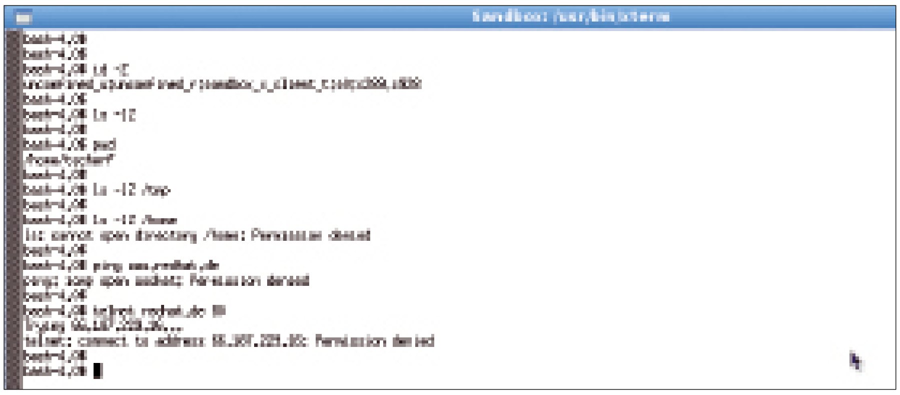
Scripts

Additionally, a sandbox is handy if you want to run new or unknown scripts and non-GUI applications safely [2]. This technique is particularly useful if you download a script and aren't sure whether the source is trustworthy. Launching the unknown application in a sandbox applies the predefined ruleset, which specifies the kind of access the script is permitted. The syntax is similar to that for graphical applications, but you can leave out the *-X* option.



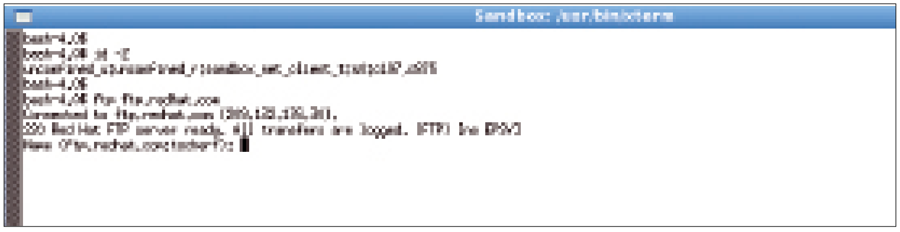
```
[tscherf@tiffy ~]$ sesearch --allow -s sandbox_x_client_t -t sandbox_x_file_t -c file -d
Found 1 semantic av rules:
   allow sandbox_x_client_t sandbox_x_file_t : file { ioctl read write create getattr setattr lock append unlink link
rename execute execute_no_trans entrypoint open } ;

[tscherf@tiffy ~]$ sesearch --allow -s sandbox_x_client_t -t tmp_t -c file -d

[tscherf@tiffy ~]$
```

Figure 1: The sandbox policy permits access to sandbox_x_file_t types of files, but not to tmp_t file types.



```
bash-4.0$
bash-4.0$
bash-4.0$ id -Z
unconfined_u:unconfined_r:sandbox_x_client_t:s0:c[illegible]
bash-4.0$
bash-4.0$ ls -lZ
bash-4.0$
bash-4.0$ pwd
/home/tscherf
bash-4.0$
bash-4.0$ ls -lZ /tmp
bash-4.0$
bash-4.0$ ls -lZ /home
ls: cannot open directory /home: Permission denied
bash-4.0$
bash-4.0$ ping www.redhat.de
ping: icmp open socket: Permission denied
bash-4.0$
bash-4.0$ telnet redhat.de 80
Trying 66.187.229.[illegible]...
telnet: connect to address 66.187.229.[illegible]: Permission denied
bash-4.0$
bash-4.0$
```

Figure 2: The sandbox only permits access to specific resources.



```
bash-4.0$
bash-4.0$ id -Z
unconfined_u:unconfined_r:sandbox_net_client_t:s0:c[illegible]
bash-4.0$
bash-4.0$ ftp ftp.redhat.com
Connected to ftp.redhat.com (209.132.183.[illegible]).
220 Red Hat FTP server ready. All transfers are logged. (FTP) [no EPSV]
Name (ftp.redhat.com:tscherf):
```

Figure 3: Launching an application with the sandbox_net_t label gives the application full network access.

```
# ls -lZ /etc/passwd
-rw-r--r--. root root ↩
   system_u:object_r:etc_t:s0 ↩
    /etc/passwd
# sandbox cat /etc/passwd|cut -d: -f1
/bin/cat: /etc/passwd: ↩
   Permission denied
```

Listing 2 would normally output a list of existing user accounts, but the policy does not define an open permission for *etc_t* types of files, which effectively prevents applications running in the sandbox from opening them (Listing 2)

This rule might appear confusing at first glance because it does specify the read and write permissions. Of course, you need to pass the file descriptors into the sandbox for this kind of access. If you change the script as shown in the following code snippet, you will have the kind of access restrictions you need:

```
cat /etc/passwd|sandbox cut -d: -f1
```

Again, it makes sense to check the policy to discover the details of permitted access (Listing 3).

If you want an application to open a file autonomously from within the sandbox, the file must be of the *sandbox_file_t* type; otherwise, this access is not permitted. Network access is generally prohibited for scripts in the *sandbox_t* domain (Listing 4); to change this, you need a ruleset for the *sandbox_t* domain.

Because development of the sandbox functions has only just begun, they are only available in the Fedora 12 beta version right now [3]. Some functions, such as copy and paste between the sandbox and the host system or the ability to store files inside the sandbox have not been implemented as of this writing, although they are likely to be available in the near future.

Kiosk System

If you like what you have heard thus far and are interested in restricting access for all the applications running on a system, you might like to check out the guest/xguest account on an SELinux system. This account lets you confine a complete user account inside a terminal (guest) or desktop session (xguest) and only permit access defined in a ruleset (guest/xguest policy module).

Whereas *Sandbox* mainly uses the SELinux Type Enforcement (TE) implementation for restricting access to resources, a guest/xguest account also relies on Role Based Access Control (RBAC). After the merge between the targeted and strict policies is accomplished, a single ruleset is available.

If you log on to a system with an xguest account, the user shell is launched in the protected SELinux xguest domain. The user is assigned to the SELinux xguest role, which only allows access to specific domains:

```
# id -Z
xguest_u:xguest_r:xguest_t:s0
```

The xguest policy module again controls access to the individual resources. All of the user's applications run in the xguest domain. The applications are not permitted unrestricted access to the network, the only exception being the Firefox web browser, which is allowed unrestricted access via HTTP. The policy also prevents the execution of files in the user's *tmp* or home directories. If a user were to download a malicious program using Firefox, the xguest policy would still stop the program and thus prevent further

Listing 3: sesearch Displays the Policy

```
$ sesearch --allow -s sandbox_t -c file -p open -d
Found 1 semantic av rules:
   allow sandbox_t sandbox_file_t : file { ioctl read write create getattr
   setattr lock append
   unlink link rename execute execute_no_trans open } ;
```

Listing 4: Ping Prohibited

```
type=AVC msg=audit(1256541838.863:183): avc:  denied  { create } for  pid=5474
comm="ping" scontext=unconfined_u:unconfined_r:sandbox_t:s0:c313,c341
tcontext=unconfined_u:unconfined_r:sandbox_t:s0:c313,c341 tclass=rawip_socket
```

Listing 5: Excerpt from audit.log

```
type=AVC msg=audit(1256575191.000:861): avc:  denied  { name_connect }
for pid=15663 comm="telnet" dest=25 scontext=xguest_u:xguest_r:xguest_t:s0
tcontext=system_u:object_r:smtp_port_t:s0 tclass=tcp_socket
```

Listing 6: xguest Policy for SMTP

```
# mkdir ~/policy
# cat > xguest_smtp.te <<eof

policy_module(xguest_smtp,1.0.0)

require {
        type xguest_t;
}

#============= xguest_t ==============
corenet_tcp_connect_smtp_port(xguest_t)

# make -f /usr/share/selinux/devel/Makefile
Compiling targeted xguest_smtp module
/usr/bin/checkmodule:  loading policy configuration from
tmp/xguest_smtp.tmp
/usr/bin/checkmodule:  policy configuration loaded
/usr/bin/checkmodule:  writing binary representation
(version 10) to
tmp/xguest_smtp.mod
Creating targeted xguest_smtp.pp policy package
rm tmp/xguest_smtp.mod.fc tmp/xguest_smtp.mod

# semodule -i xguest_smtp.pp
# semodule -l |grep xguest_smtp
xguest_smtpt  1.0.0
```

damage. But if you really want to grant an xguest user permission to run arbitrary files, you can use a Boolean:

```
setsebool -P
   allow_xguest_exec_content=1
```

To allow access to other network services from within the xguest domain, you again need a separate policy module. Say you want to let an xguest user send mail; in this case, the domain needs access to the SMTP port. Without a rule allowing this, access to SMTP would be prohibited, as you can see from the *audit.log* excerpt in Listing 5. A separate module will grant SMTP access (Listing 6).

After adding the new policy module to the system, SMTP-based access should now work. If you want to extend the policy, check *audit.log* for more deny messages and then add instructions to the policy module to change the access types. One remaining question is how to add a user account to the xguest domain. The simplest case would be to use the *usermod* tool for existing accounts or *useradd* for new accounts (Listing 7). When a user logs in to a desktop, the whole user session runs in the protected SELinux xguest domain:

```
$ id -Z
xguest_u:xguest_r:xguest_t:s0
```

Listing 7: New Accounts for xguest

```
# usermod -Z xguest_u foobar
# semanage login -l

Login Name                    SELinux User              MLS/MCS Range

__default__                   unconfined_u              s0-s0:c0.c1023
foobar                        xguest_u                  s0
root                          unconfined_u              s0-s0:c0.c1023
system_u                      system_u                  s0-s0:c0.c1023
xguest                        xguest_u                  s0
```

Conclusions

The new SELinux Sandbox technology makes it easy to place an application in the protective hands of SELinux. The SELinux rulesets were designed to protect individual system services, and Mandatory Access Control protection is now extended to normal users. Thanks to xguest, new types of applications are on the SELinux radar screen. ■

INFO

[1] SELinux: *http://selinux.sourceforge.net/*

[2] Introducing the SELinux sandbox: *http://danwalsh.livejournal.com/28545.html*

[3] SELinux with Fedora: *http://fedoraproject.org/wiki/SELinux*