Monday, February 4. 2008

Less known Solaris features: RBAC and Privileges - Part 3: Privileges

Privileges
We´ve talked a lot about RBAC, roles, role profiles. But what are Privileges? Privileges are rights to do an operation in the kernel. This rights are enforced by the kernel. Whenever you do something within the kernel the access is controlled by the privileges.At the moment, the the rights to do something with the kernel are seperated into 70 classes: contract_event contract_observer cpc_cpu dtrace_kernel dtrace_proc dtrace_user file_chown file_chown_self file_dac_execute file_dac_read file_dac_search file_dac_write file_downgrade_sl file_flag_set file_link_any file_owner file_setid file_upgrade_sl graphics_access graphics_map ipc_dac_read ipc_dac_write ipc_owner net_bindmlp net_icmpaccess net_mac_aware net_privaddr net_rawaccess proc_audit proc_chroot proc_clock_highres proc_exec proc_fork proc_info proc_lock_memory proc_owner proc_priocntl proc_session proc_setid proc_taskid proc_zone sys_acct sys_admin sys_audit sys_config sys_devices sys_ip_config sys_ipc_config sys_linkdir sys_mount sys_net_config sys_nfs sys_res_config sys_resource sys_smb sys_suser_compat sys_time sys_trans_label win_colormap win_config win_dac_read win_dac_write win_devices win_dga win_downgrade_sl win_fontpath win_mac_read win_mac_write win_selection win_upgrade_slEvery UNIX-System does this task hidden behind this privileges. There are many different privileges in the kernel. This privileges are not Solaris specific. It´s the way to control the access to this privileges.

Conventional Unix
On conventional unix systems you have a root user, he has all privileges. And you have a normal user, who has only a limited set of privileges. Sometimes you need the rights of an admin to do some tasks. You don´t even need to admin the system. You can only traceroute or ping a system, because both tools are setuid tools$ ls -l /usr/sbin/traceroute
-r-sr-xr-x 1 root bin 42324 Nov 21 00:09 /usr/sbin/traceroute
$ ls -l /usr/sbin/ping
-r-sr-xr-x 1 root bin 51396 Nov 18 19:31 /usr/sbin/pingsetuid is nothing else than a violation of the security policy. You need a special privilege to ping: The privilege to use access ICMP. On conventional system this right is reserved to the root user. Thus the ping program has to be executed with the rights of root. The problem: At the time of the execution of the programm, the programm has all rights of the user. Not only to access ICMP, the programm is capable to do everything on the system, as deleting files in /etc. This may not a problem with ping or traceroute but think about larger programs. An exploit in a setuid program can lead to the escalation of the users privileges. Setuid root and you are toast.

Let´s have a look at the privileges of an ordinary user. There is a tool to get the privileges of any given process in the system, it´s called ppriv. $$ is a shortcut for the actual process id (in this case the process id of the shell):bash-3.2$ ppriv -v $$
646: bash
flags =
E: file_link_any,proc_exec,proc_fork,proc_info,proc_session
I: file_link_any,proc_exec,proc_fork,proc_info,proc_session
P: file_link_any,proc_exec,proc_fork,proc_info,proc_session
L: contract_event, (..) ,win_upgrade_slEvery process in the system has four sets of privileges that determine if a process is enabled to use a priviliege or not. The theory of privileges is quite complex. i would suggest to read the chapter "How Privileges Are Implemented" in the Security Services manual to learn, how each set controls or is controlled other privilege sets.

At this time, i want only to explain the meaning of the first letter:E: effective privileges setI: inheritable privileges setP: permitted privileges setL: limit privileges setYou can think about the privilege sets as keyrings. The effective privilege set are the keys the janitor has on it´s keyring. The permited privilege set are the keys the janitor is allowed to put on it´s keyring. The janitor can decide to remove some of the keys. Perhaps he thinks: I work only in room 232 today. I don´t need all the other keys. I leave them in my office. When he looses his keyring he lost only the control about this single room, not about the complete campus.

The inheritable privilege is not a really a keyring. The janitor thinks about his new assistant: "Good worker, but i won´t give him my key for the room with the expensive tools." The limited privilege set is the overarching order from the boss of janitor to his team leaders: "You are allowed to give your assistant the keys for normal rooms, but not for the rooms with all this blinking boxes from Sun".

At the moment the most interesting set is the E:. This is the effective set of privileges. This is the set of privilege effectively available to process. Compared to the full list of privileges mentioned above the set is much smaller. But this matches your experience when you use a unix system.

Some practical insighs to the system
You logged in as a normal user, and you have only a few privileges. It´s called the basic set.bash-3.2$ ppriv $$
815:	bash
flags =
	E: basic
	I: basic
	P: basic
	L: allOkay, this example looks different than the one shown before. Nevertheless is has the same meaning. With the switch -v you can expand the aliases.bash-3.2$ ppriv -v $$
815:	bash
flags =
	E: file_link_any,proc_exec,proc_fork,proc_info,proc_session
	I: file_link_any,proc_exec,proc_fork,proc_info,proc_session
	P: file_link_any,proc_exec,proc_fork,proc_info,proc_session
	L: contract_event, (..) ,win_upgrade_slLooks a little bit more familiar? Okay, now let´s login as root.$su root
Password:
ppriv $$
819:	sh
flags =
	E: all
	I: basic
	P: all
	L: all
This user has much more privileges. The effective set is much broader. The user has all privileges in the system.

How to give an user additional privileges
Now let´s assume, you have an user, that wants to use dtrace. You need three privileges to use Dtrace: dtrace_kernel,dtrace_proc,dtrace_user. root has this privileges. A normal user not. Giving root to a developer? God beware! This is a prelude to disaster. But no problem. Assign the matching privileges to the user, and the user is enabled to use dtrace.$ su root
Password:
usermod -K defaultpriv=basic,dtrace_kernel,dtrace_proc,dtrace_user jmoekamp
UX: usermod: jmoekamp is currently logged in, some changes may not take effect until next login.Exit to the login prompt and login as the user you´ve assigned the privilieges $ ppriv $$
829:	-sh
flags =
	E: basic,dtrace_kernel,dtrace_proc,dtrace_user
	I: basic,dtrace_kernel,dtrace_proc,dtrace_user
	P: basic,dtrace_kernel,dtrace_proc,dtrace_user
	L: all
Simple ...

RBAC and privileges combined
Well, but we can do better than that. We´ve learned there is a thing like RBAC. There is no reason that inhibits the assignment of privileges to a role. At first we create a role bughunt, for simplicity we use the Process Management role profile. After this we set the role password.# roleadd -m -d /export/home/bughunting -P "Process Management" bughunt
passwd bughunt
New Password:
Re-enter new Password:Now we assign the privileges ...
rolemod -K defaultpriv=basic,dtrace_kernel,dtrace_proc,dtrace_user bughunt... and the user to the role.# usermod -R bughunt jmoekamp
UX: usermod: jmoekamp is currently logged in, some changes may not take effect until next login.As you might have espected, the user itself doesn´t have the privileges to use dtrace. $ ppriv $$
883:	-sh
flags =
	E: basic

```
        I: basic
        P: basic
        L: allBut now assume the role bughunt$ su bughunt
Password:
$ ppriv $$
893:    pfsh
flags =
        E: basic,dtrace_kernel,dtrace_proc,dtrace_user
        I: basic,dtrace_kernel,dtrace_proc,dtrace_user
        P: basic,dtrace_kernel,dtrace_proc,dtrace_user
        L: allAnd DTrace is at your service.
```

Privilege-aware programming
The idea of managing the privileges is not limited to users and their shells. In any given system you find dozens of programs as daemons.

These daemons interact in several ways with the privileges. The best way is "Privilege-aware programming". Okay. Let´s assume, you code a daemon for your system. For example: You know, that you never will do an exec() call. So you can safely drop this privilege. The process modifies the permited privilege set. The process can remove a privilege but not add it. Even when someone is able to your code, the attacker can´t make an exec() call. The process doesn´t even have the privilege to do such a call. And the attacker can´t add the privilege again.

Several processes and programs in Solaris are already privilege aware. For example the kernel-level cryptographic framework daemon. Let´s look at the privileges of the daemon.

```
# ps -ef | grep "kcfd"
  daemon   125     1   0 14:24:19 ?           0:00 /usr/lib/crypto/kcfd
    root   734   728   0 15:54:08 pts/1       0:00 grep kcfd
# ppriv -v 125
125:    /usr/lib/crypto/kcfd
flags = PRIV_AWARE
        E: file_owner,proc_priocntl,sys_devices
        I: none
        P: file_owner,proc_priocntl,sys_devices
        L: noneThis daemon doesn´t have even the basic privileges of a regular user. It has the only the bare minimum of privileges to do it´s job.
```

Non-privilege aware processes

But the world isn´t perfect. Not every process is privilege aware. Thus you have to limit the privileges by other mechanisms. The service management framework comes to help. The following example is copied from Glen Brunettes Blueprint Limiting Service Privileges in the Solaris 10 Operating System

Let´s take the Apache Webserver as an example. The apache isn´t privilege aware. We start the daemon via the Service Management Framework.

```
# svcadm -v enable -s apache2
svc:/network/http:apache2 enabled.Okay, now we look at the processes of the Apache daemons.
# ps -ef | grep "apache"
webservd  1123  1122   0 19:11:54 ?           0:00 /usr/apache2/2.2/bin/httpd -k start
webservd  1125  1122   0 19:11:54 ?           0:00 /usr/apache2/2.2/bin/httpd -k start
    root  1122     1   1 19:11:50 ?           0:00 /usr/apache2/2.2/bin/httpd -k start
webservd  1128  1122   0 19:11:54 ?           0:00 /usr/apache2/2.2/bin/httpd -k start
webservd  1127  1122   0 19:11:54 ?           0:00 /usr/apache2/2.2/bin/httpd -k start
webservd  1126  1122   0 19:11:54 ?           0:00 /usr/apache2/2.2/bin/httpd -k start
webservd  1124  1122   0 19:11:54 ?           0:00 /usr/apache2/2.2/bin/httpd -k startSix daemons running as webservd, and one running as root.
# ppriv  1122
1122:   /usr/apache2/2.2/bin/httpd -k start
flags =
        E: all
        I: basic
        P: all
        L: allAs expected for a root process, this process has the complete set of privileges of a root user. Okay, now one of it´s child. # ppriv 1124
1124:   /usr/apache2/2.2/bin/httpd -k start
```

flags =
E: basic
I: basic
P: basic
L: allMuch better ... only basic privileges.

Okay, There is a reason for this configuration. On Unix systems, you have two groups of ports. Privileged ones from 1-1023 and unprivileged ones from 1024 up. You can only bind to a privileged port with the privilege to do it. A normal user doesn´t have this privilege, but root has it. And thus there has to be one process running as root. Do you remeber the list of privileges for the apache process running at root. The process has all privileges but needs only one of them, that isn´t part of the basic privilege set.

How to get rid of the root apache
Well, but it hasn´t to be this way. With Solaris you can give any user or process the privilege to use a privileged port. You don´t need the root process anymore.

Now, let´s configure it this way. At first we have to deactivate the runing apache.svcadm -v disable -s apache2
svc:/network/http:apache2 disabled.I won´t explain the Service Management Framework here, but you can set certain properties in SMF to control the startup of a service.# svccfg -s apache2
svc:/network/http:apache2> setprop start/user = astring: webservd
svc:/network/http:apache2> setprop start/group = astring: webservd
svc:/network/http:apache2> setprop start/privileges = astring: basic,!proc_session,!proc_info,!file_link_any,net_privaddr
svc:/network/http:apache2> setprop start/limit_privileges = astring: :default
svc:/network/http:apache2> setprop start/use_profile = boolean: false
svc:/network/http:apache2> setprop start/supp_groups = astring: :default
svc:/network/http:apache2> setprop start/working_directory = astring: :default
svc:/network/http:apache2> setprop start/project = astring: :default
svc:/network/http:apache2> setprop start/resource_pool = astring: :default
svc:/network/http:apache2> endLine 2 to 4 are the most interesting ones. Without any changes, the Apache daemon starts as root and forks away processes with the webservd user. But we want to get rid of the root user for this configurtion. Thus we start the daemon directly with the webservd user. Same for the group id.

Now it gets interesting. Without this line, the kernel would deny Apache to bind to port 80. webservd is a regular user without the privilege to use a privileged port. The property start/privileges sets the privileges to start the service. At first, we give the service basic privileges. Then we add the privilege to use a privileged port. The service would start up now.

But wait, we can do more. A webserver shouldn´t do any hardlinks. And it doesn´t send signals outside it´s session. And it doesn´t look at processes other than those to which it can send signals. We don´t need this privileges. proc_session, proc_info and file_link_any are part of the basic privilege set. We remove them, by adding a ! in front of the privilege.

Okay, we have notify the SMF of the configuration changes: # svcadm -v refresh apache2
Action refresh set for svc:/network/http:apache2.

Until now, the apache daemon used the root privileges. Thus the ownership of files and directories were unproblematic. The daemon was able to read and write in any directory of file in the system. As we drop teis privilege by using a regular user, we have to modify the ownership of some files and move some files.# chown webservd:webservd /var/apache2/2.2/logs/access_log
chown webservd:webservd /var/apache2/2.2/logs/error_log
mkdir -p -m 755 /var/apache2/run

We need some configuration changes, too. We have to move the LockFile and the PidFile. There wasn´t one of the two configuration directives in my config file, thus i´ve simply appended them to the end of the file.# echo "LockFile /var/apache2/2.2/logs/accept.lock" >> /etc/apache2/2.2/httpd.conf
echo "PidFile /var/apache2/2.2/run/httpd.pid" >> /etc/apache2/2.2/httpd.confOkay, everything is in place. Let´s give it a try.
svcadm -v enable -s apache2
svc:/network/http:apache2 enabled.Now we check for the running httpd processes:# ps -ef | grep "apache2" | grep -v "grep"
webservd 2239 2235 0 19:29:54 ? 0:00 /usr/apache2/2.2/bin/httpd -k start
webservd 2241 2235 0 19:29:54 ? 0:00 /usr/apache2/2.2/bin/httpd -k start
webservd 2235 1 1 19:29:53 ? 0:00 /usr/apache2/2.2/bin/httpd -k start

webservd 2238 2235 0 19:29:54 ? 0:00 /usr/apache2/2.2/bin/httpd -k start
webservd 2240 2235 0 19:29:54 ? 0:00 /usr/apache2/2.2/bin/httpd -k start
webservd 2242 2235 0 19:29:54 ? 0:00 /usr/apache2/2.2/bin/httpd -k start
webservd 2236 2235 0 19:29:54 ? 0:00 /usr/apache2/2.2/bin/httpd -k startYou notice the difference ? There is no httpd running as root. All processes run with the userid webservd. Mission accomplished.

Let´s check the privileges of the processes. At first the one, who ran as root before. # ppriv 2235
2235: /usr/apache2/2.2/bin/httpd -k start
flags =
E: basic,!file_link_any,net_privaddr,!proc_info,!proc_session
I: basic,!file_link_any,net_privaddr,!proc_info,!proc_session
P: basic,!file_link_any,net_privaddr,!proc_info,!proc_session
L: allOnly the least privileges to do the job, no root privileges.

And even the other processes are more secure now:# ppriv 2238
2238: /usr/apache2/2.2/bin/httpd -k start
flags =
E: basic,!file_link_any,net_privaddr,!proc_info,!proc_session
I: basic,!file_link_any,net_privaddr,!proc_info,!proc_session
P: basic,!file_link_any,net_privaddr,!proc_info,!proc_session
L: allBefore we changed the configuration of the webserver, it has the basic privileges of a regular user. Now we limited even this set.

`Posted by Joerg Moellenkamp in English, Solaris at 13:06`

Hi Jörg,
this is a real nice feature.
Except for Apaches with SSL certificates. I have to change the read permissions for the key files to the user apache runs at.
Can you describe a method to start Apache as a non root user with maybe file_dac_read and proc_setid? And a second user which is used as User in the configuration?
Does it make sense? Is the start user so mighty that we can even use root?
Or should we run two Apache where one ist started as root and handles SSL requests and send them to the second one?
Many questions, I know...
Thanks for your braindumps
Lars
Anonymous on Nov 4 2009, 10:50