pspy – unprivileged linux process snooping

pspy is a command line tool designed to snoop on processes without need for root permissions. It allows you to see commands run by other users, cron jobs, etc. as they execute. Great for enumeration of Linux systems in CTFs. Also great to demonstrate your colleagues why passing secrets as arguments on the command line is a bad idea.

The tool gathers it's info from procfs scans. Inotify watchers placed on selected parts of the file system trigger these scans to catch short-lived processes.

Getting started

Get the tool onto the Linux machine you want to inspect. First get the binaries. Download the released binaries here:

- 32 bit big, static version: pspy32 download
- 64 bit big, static version: pspy64 download
- 32 bit small version: pspy32s download
- 64 bit small version: pspy64s download

The statically compiled files should work on any Linux system but are quite huge (~4MB). If size is an issue, try the smaller versions which depend on libc and are compressed with UPX (<1MB). Alternatively, build the binaries yourself. Either use Go installed on your system or run the Docker-based build process which ran to create the release. For the latter, ensure Docker is installed, and then run make build-build-image to build a Docker image, followed by make build to build the binaries with it.

You can run pspy --help to learn about the flags and their meaning. The summary is as follows:

- -p: enables printing commands to stdout (enabled by default)
- -f: enables printing file system events to stdout (disabled by default)
- -r: list of directories to watch with Inotify. pspy will watch all subdirectories recursively (by default, watches /usr, /tmp, /etc, /home, /var, and /opt).
- -d: list of directories to watch with Inotify. pspy will watch these directories only, not the subdirectories (empty by default).
- -i: interval in milliseconds between procfs scans. pspy scans regularly for new processes regardless of Inotify events, just in case some events are not received.
- -c: print events in different colors. Red for new processes, green for new Inotify events.

The default settings should be fine for most applications. Watching files inside /usr is most important since many tools will access libraries inside it.

Some more complex examples:
[crayon-6a9d575c5c08c494436616/]

Examples

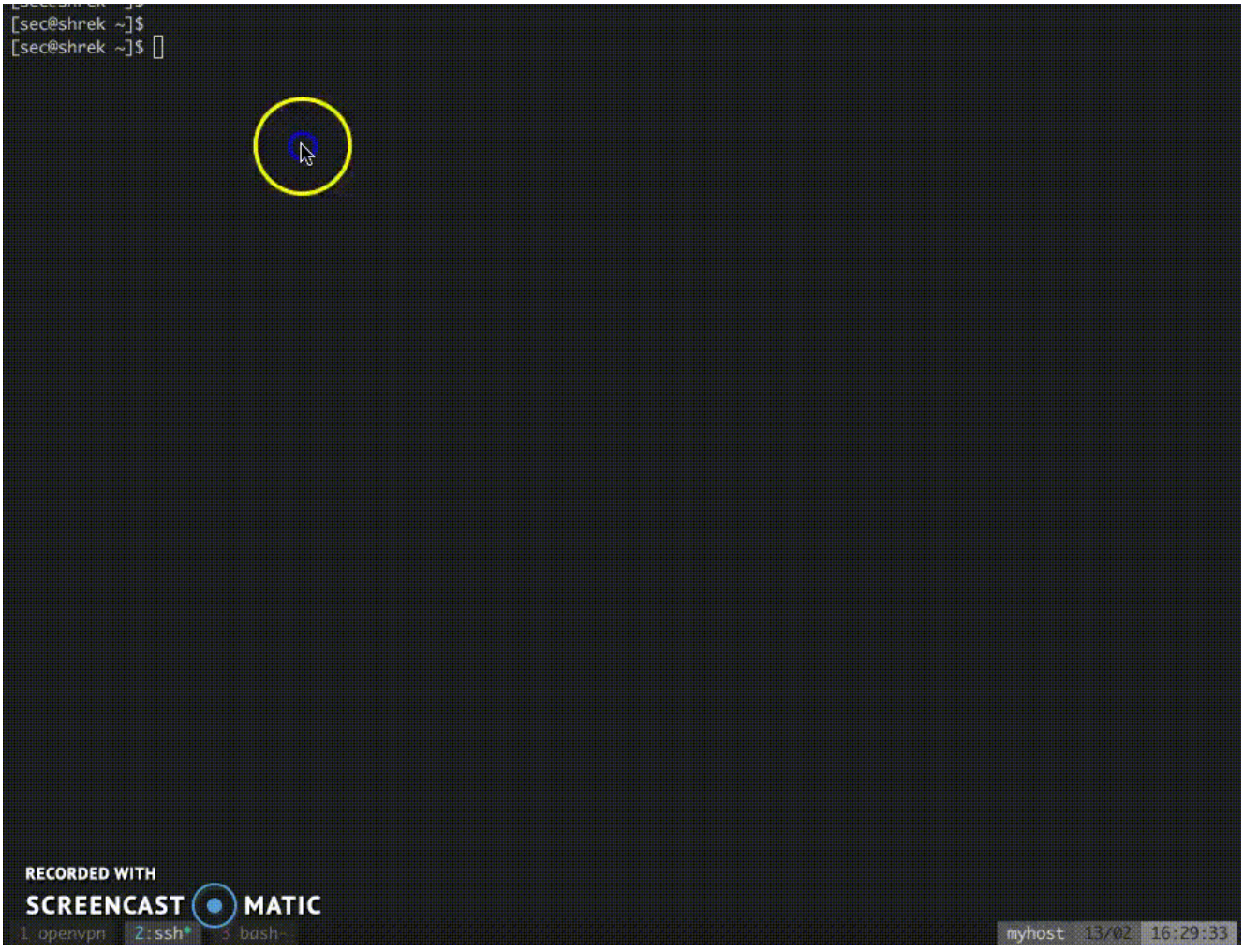
Cron job watching

To see the tool in action, just clone the repo and run make example (Docker needed). It is known passing passwords as command line arguments is not safe, and the example can be used to demonstrate it. The command starts a Debian container in which a secret cron job, run by root, changes a user password every minute. pspy run in foreground, as user myuser, and scans for processes. You should see output similar to this:
[crayon-6a9d575c5c092752003658/]
First, pspy prints all currently running processes, each with PID, UID and the command line. When pspy detects a new

process, it adds a line to this log. In this example, you find a process with PID 23 which seems to change the password of myuser. This is the result of a Python script used in roots private crontab /var/spool/cron/crontabs/root, which executes this shell command (check crontab and script). Note that myuser can neither see the crontab nor the Python script. With pspy, it can see the commands nevertheless.

CTF example from Hack The Box

Below is an example from the machine Shrek from Hack The Box. In this CTF challenge, the task is to exploit a hidden cron job that's changing ownership of all files in a folder. The vulnerability is the insecure use of a wildcard together with chmod (detailsfor the interested reader). It requires substantial guesswork to find and exploit it. With pspy though, the cron job is easy to find and analyse:



How it works

Tools exist to list all processes executed on Linux systems, including those that have finished. For instance there is forkstat. It receives notifications from the kernel on process-related events such as fork and exec.

These tools require root privileges, but that should not give you a false sense of security. Nothing stops you from snooping on the processes running on a Linux system. A lot of information is visible in procfs as long as a process is running. The only problem is you have to catch short-lived processes in the very short time span in which they are alive. Scanning the /procdirectory for new PIDs in an infinite loop does the trick but consumes a lot of CPU.

A stealthier way is to use the following trick. Process tend to access files such as libraries in /usr, temporary files in /tmp, log files in /var, … Using the inotify API, you can get notifications whenever these files are created, modified, deleted, accessed, etc. Linux does not require priviledged users for this API since it is needed for many innocent applications (such as text editors showing you an up-to-date file explorer). Thus, while non-root users cannot monitor processes directly, they can monitor the effects of processes on the file system.

We can use the file system events as a trigger to scan /proc, hoping that we can do it fast enough to catch the processes. This is what pspy does. There is no guarantee you won't miss one, but chances seem to be good in my experiments. In general, the longer the processes run, the bigger the chance of catching them is.