Understanding nohup

Taz Wake | Halkyn Consulting Ltd | Originally published 2024-01-15

What is nohup?

In Linux, nohup is a command used to allow processes to become persistent. Short for "*no hang up*," the command's primary function is to enable a process to continue running in the background even after the user has logged out from the system or killed the running session.

Because of the way nohup works, it is important that all security professionals, especially incident responders, have a basic understanding and know what to look for.

Core Functionality of nohup

Basically, nohup is a command-line utility that alters the default behaviour of the system. Normally, a process terminates if the user running the process exits the environment, or if the session the process is running in is exited. Orphan processes are more common in Linux than we might see in Windows, but this is really only when the parent process exits rather than the entire session being exited.

In use, nohup effectively detaches the process from the terminal, allowing it to run independently of the user session. This detachment is crucial for long-running processes or tasks that need to survive beyond the user's active engagement with the system.

How does it do that?

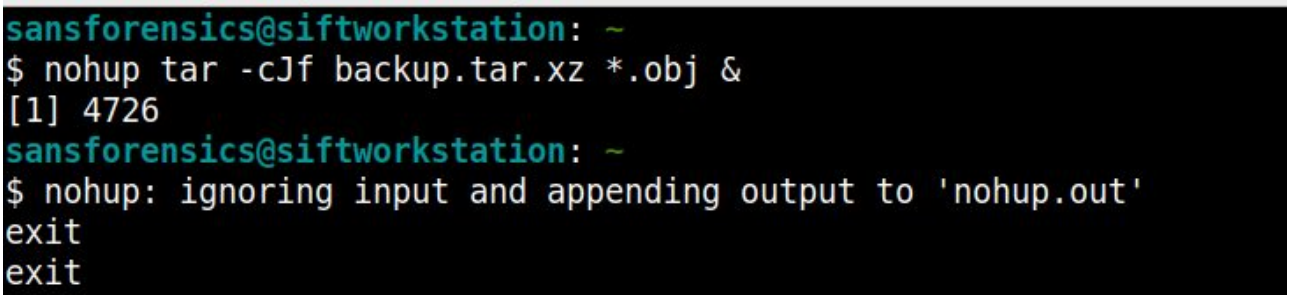
Pretty simply really.

Under the hood, nohup does its thing by intercepting the "SIGHUP" (signal hang up), which is sent to processes when their controlling terminal is closed. By ignoring this signal, nohup ensures that the process doesn’t terminate, instead it continues running in the background.

Using nohup

Practically, nohup is widely used in system administration and operations where processes need to be kept alive for extended periods. This is especially relevant for tasks such as data backups, system updates, and batch jobs, which may take a considerable amount of time to complete and are not dependent on an active terminal session.

The command line arguments are pretty simple – it is just **nohup [command]** – as shown below:



Example of the nohup command - this is running tar to create an xz compressed tar archive of all files with a *.obj extension in the current folder. After running the command the user exits the session but the command continues in the background.

Incident response and forensic investigations

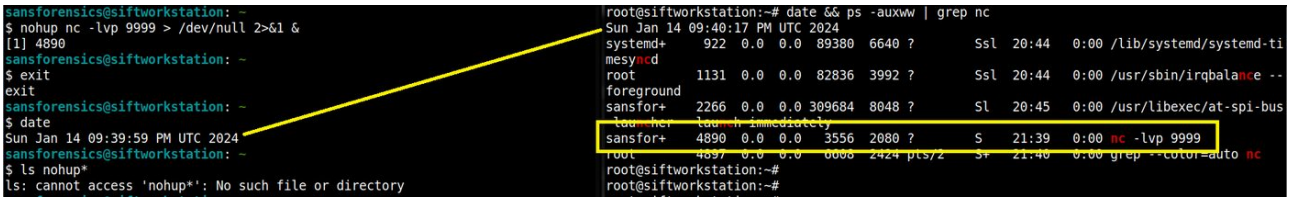
To maintain its operations, nohup typically redirects the standard output (stdout) and standard error (stderr) of the process to a file, commonly nohup.out, if not otherwise specified by the user. This file, usually created in the directory where the command is executed, serves as a log for the process, capturing its output and error messages post-disconnection.

Additionally, system logs and process monitoring tools can reveal the invocation of nohup and the associated process details. It is important that every Linux system has auditd installed **and** well configured, although that is outside the scope of this article. In RHEL family distros, auditd is installed by default but it is rarely well configured and currently, it isn't installed on Debian-family distros by default.

Easy attacker countermeasure

Sadly, it is entirely possible to send the output of nohup to /dev/null, which effectively removes the main artifact we look for in an investigation.

This is an example of running nc to create a listener, pointing the output to /dev/null and then exiting the session. As you can see, the process is still running but no nohup.out file exists.



Example output

This means, in most practical incident response scenarios it can be very challenging to see if something was running with nohup.

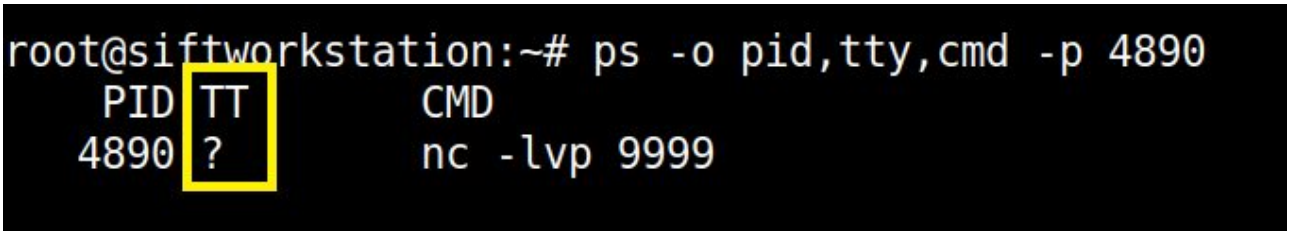
Forensic notes

- The running process is now a child process of the systemd process, which can make it difficult to spot in pstree output. However, this does depend on how obvious the process name is. Here you need to use basic process analysis methods to identify the

potential malicious process but there is very little evidence that nohup had been used (as there are lots of reasons why systemd can become the parent process in Linux)

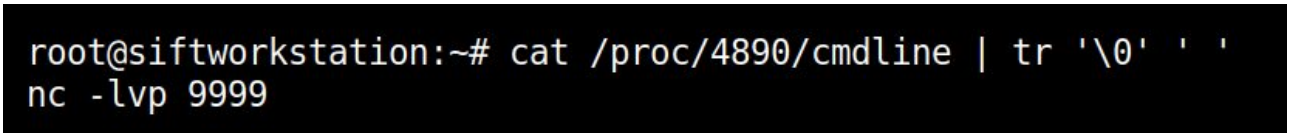
The nc command is now a child process of systemd, shown in pstree output.

- The tty session for the process no longer exists and is shown as a ? in ps output. This isn't a great thing to hunt for, however, as lots of processes are detached from a terminal.



ps output in Linux showing a question mark for the terminal session.

- The command line for the process does not (normally) maintain any reference to nohup. This is because nohup is a separate process that has exited. Really, it is unlikely there will be any reference to nohup in /proc at all.

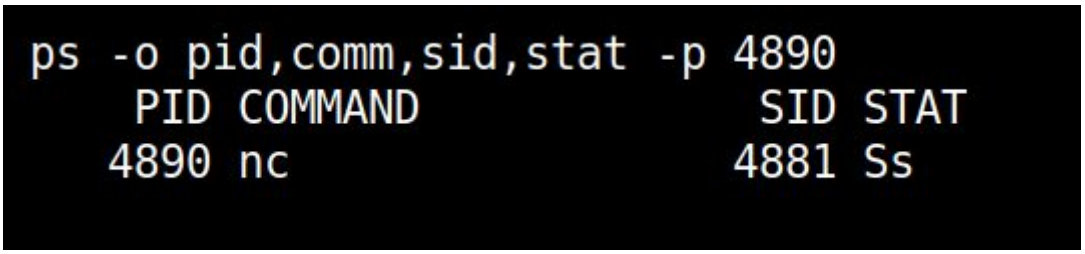


The command line file does not refer to the nohup command.

Forensicator solutions

Possibly the best way to validate if a process is running with nohup is to see if it responds to the SIGHUP signal from the operating system. The whole purpose of nohup is to prevent a process from responding, so this is likely to be the best thing to check. As always, YMMV based on the specific incident you are dealing with.

One way to determine this is to check if the process is running as "session leader." This term is used to describe a process that started a session, typically a login shell or a process that initiates others, often grouping related processes. Being a session leader has implications for signal distribution and terminal control. You can find this by viewing the stat value with ps:

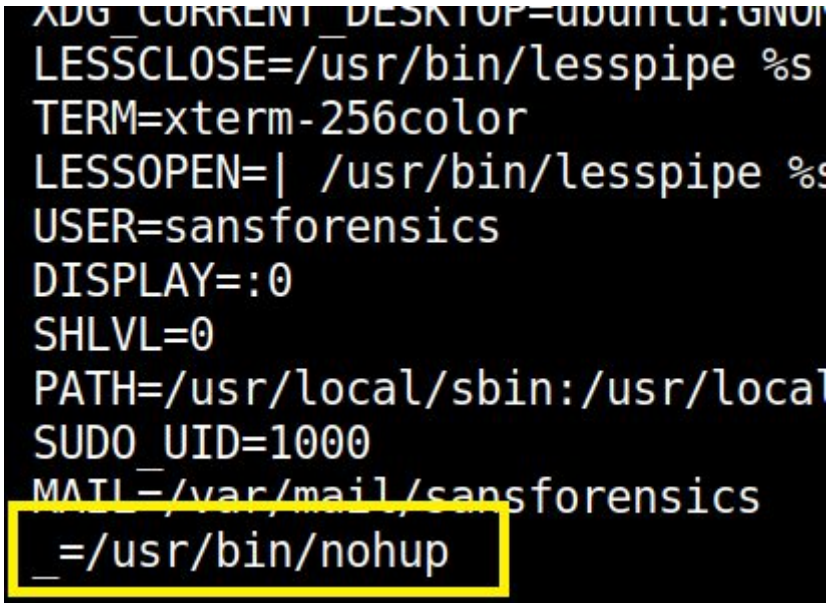


Example of finding a nohup'd process with the Session Leader status

It is worth noting that this is inconsistent and may depend on the process characteristics.

Other places to check, with varying degrees of value, include:

- Check the process environment with a command similar to the one below. This can be a very effective way of proving nohup use.



Evidence nohup was used, appended to the end of the process environment settings in /proc

- Check the process status and see if the SigIgn value identifies SIGHUP is being ignored. This is reasonably complex, as the values here are binary showing whether or not a particular signal is ignored, with each signal having its own position in the

string. The good news is that SIGHUP is the rightmost bit, so may be a bit easier to check. In reading the output, a 1 means that the signal at that point is ignored by the process.

```
root@siftworkstation:~# cat /proc/4890/status | grep SigIgn
SigIgn: 0000000000001001
```

Checking the SigIgn value of a running process. In this example it shows the rightmost bit is a 1, so SIGHUP is being ignored by this process.

- Attach a debugger and review the file. In gdb, using **call (void)signal(1, SIG_IGN)** or **call (void)signal(1, 1)**, might indicate if the process is ignoring SIGHUP.

***Note**: This has inconsistent results and gdb might misinterpret the signal handling. In use, gdb will terminate the process if it receives a SIGHUP signal and the application may show this output rather than the process's handling.*

- Check the history file. Hopefully, the nohup command will have been captured.
- Check the audit logs/journal. This is especially valuable if you have Sysmon for Linux running.

Solutions?

Really, the best approach is to have good, continuous, endpoint monitoring in place. Auditd, Sysmon for Linux and an EDR would be ideal.

If you don't have at least one of them, then it is going to be very hard work to prove nohup was used.

About Halkyn Consulting

Halkyn Consulting Ltd is a UK-based security and risk management consultancy providing 24/7 incident response, digital forensics, threat hunting, virtual security leadership (vCISO) and security compliance support to organisations worldwide. We deliver clear, practical, vendor-independent advice based on real-world threats — no hype, no upsell.

More free security resources are available at www.halkynconsulting.co.uk/security-resources.

Web: www.halkynconsulting.co.uk | **Email:** info@halkynconsulting.co.uk | **Tel:** +44 1244 940858