Understanding Linux Anti-Forensics for Incident Responders

Taz Wake | Halkyn Consulting Ltd | Originally published 2025-10-27

When you work in Incident Response, "anti-forensics" techniques become pretty important. Even though we don't always operate in the same way as our forensics brethren, anti-forensics can undermine IR just as effectively. On a positive note, if you work your cases in a methodological manner, it is surprising how ineffective a lot of anti-forensics becomes.

Introduction

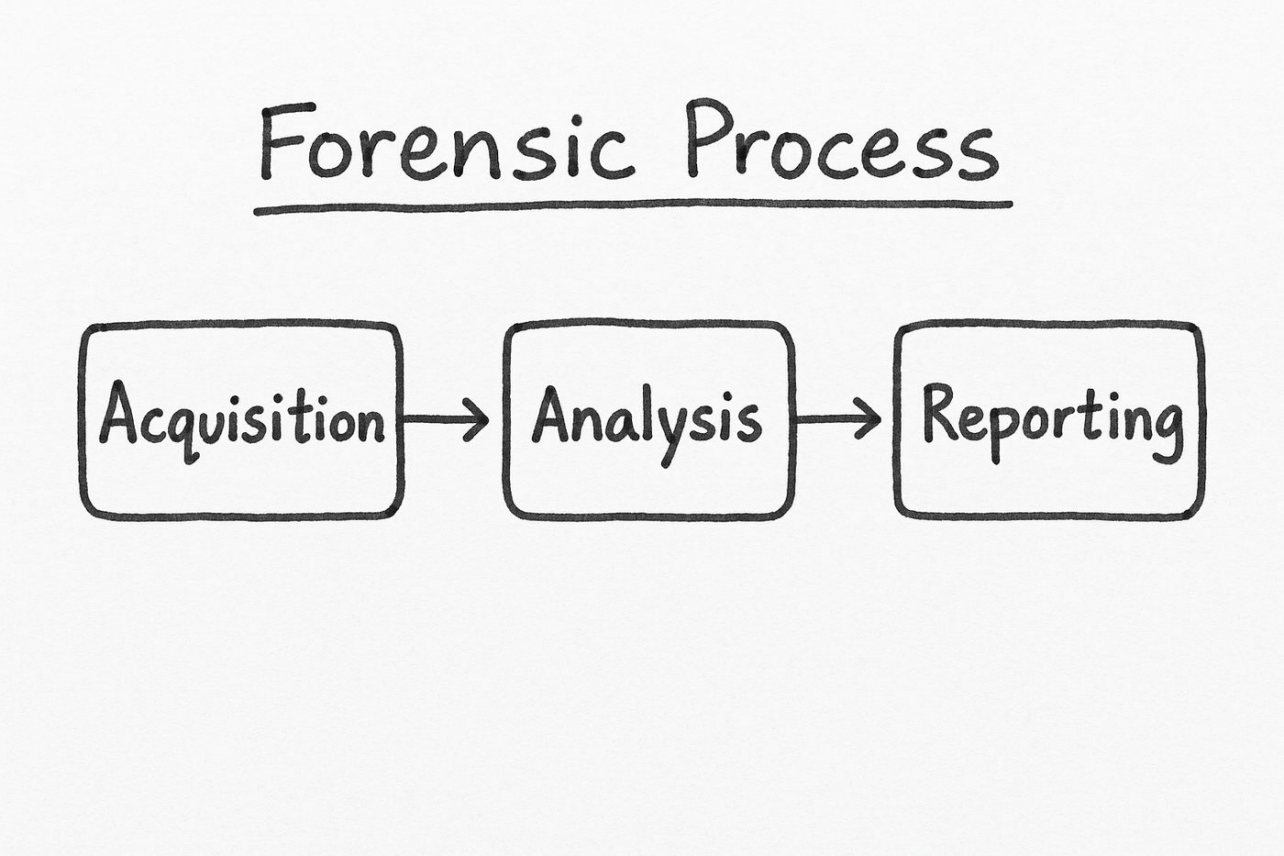
For our purposes, "anti-forensics" refers to the deliberate use of tools and methods to obstruct digital investigations. In Linux environments, these techniques are becoming more common among both opportunistic attackers and well-resourced adversaries. Their objective is straightforward: reduce visibility, corrupt evidence, and increase uncertainty during incident response.

We might not be in the year of *Linux on the Desktop*, but it really does underpin servers, containers, and cloud workloads. I've spoken to a lot of people who say "we don't have Linux in our environment", but they *always* do. The fact that you are reading this over the internet indicates there is at least one Linux device in use between you and this article.

Because of this, analysts must understand how adversaries subvert forensic workflows and what we might be able to do to find and detect the attacks.

Understanding Anti-Forensics

Fundamentally, *Incident Response* relies on *Digital Forensic* techniques. While it is also a simplification, it is true to say that digital forensics depends on integrity and context. Our job is to reconstruct what happened, when, and how.



Anti-forensic techniques target those dependencies, typically by disrupting three stages:

- **Acquisition** – destroying or altering artefacts before they can be captured.
- **Analysis** – introducing falsified or misleading data to distort conclusions.
- **Reporting** – removing context or metadata that links actions to a threat actor.

The intent isn't to achieve perfect invisibility. It's to delay investigators, introduce doubt, and degrade confidence in their findings. If a threat actor can make our job harder, it is a (mini) victory for them.

Common Anti-Forensic Techniques on Linux

Metadata Manipulation

Most filesystems (including ones used by Linux distros) store file metadata within inodes rather than in the file itself. The inode contains the important, "catalogue," data about a file that allows the operating system to understand and interact with the file. Examples of what gets stored include the file size, file ownership, location on disk, and, importantly for us, the file timestamps.

Most filesystems (today, anyway) support four timestamps: file creation, file access, file content modification, and file metadata change. This is the data we use to understand the sequence of events on a filesystem.

Attackers exploit this, generally to rewrite timestamps, a tactic known as *timestomping*. Tools like **touch**, **debugfs**, or even custom scripts can easily falsify times.

Possibly the easiest and most basic attack is the **touch** command:

This will change the access and content modification times of to be 2020-01-01 10:13:12. For an attacker, this has the possible benefit of making an investigator think that the log is "out of scope" for their investigation.

From a responder's perspective, however, this is often quite easy to spot if we simply use tools like .

```
taz@stingray:~$ touch -t 202001011013.12 victim_file


taz@stingray:~$ ls -l victim_file
-rw-r--r-- 1 taz taz 2048557 Jan  1  2020 victim_file


taz@stingray:~$ stat victim_file
  File: victim_file
  Size: 2048557         Blocks: 4008       IO Block: 4096   regular file
Device: 8,48    Inode: 2324        Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/     taz)   Gid: ( 1000/     taz)
Access: 2020-01-01 10:13:12.000000000 +0000
Modify: 2020-01-01 10:13:12.000000000 +0000
Change: 2025-10-25 19:50:23.142550841 +0100
 Birth: 2025-10-25 19:49:33.353929186 +0100
```

Data Hiding

Hiding data under dotfiles remains a simple but effective tactic, especially in paths like **/dev/shm** or **/var/tmp**. The /dev/shm folder, intended to be a "sharable memory" location, allowing processes to share data and store objects like mutexes, is extensively abused by threat actors. Staging data in here is effective because it isn't captured in disk images, and our investigations often overlook it. **Fun fact**: Most auditd configurations even explicitly exclude this folder...

More sophisticated approaches include:

- **Steganography**: embedding data within images or audio using **steghide** or **outguess**.
- **Extended attributes**: storing data within filesystem metadata via **setfattr** or **attr**. This is similar to ADS in Windows.

Today, steganography is **not commonly** found in attacks. It is more something we see in CTF or Red-Team type engagements. However, I want to caveat this, because it is hard to detect, so if a lot of attackers are using it *well*, we might just never know...

Really, for this type of activity, you need to go threat hunting. Good things to check are hidden files and folders (anything preceded by a character, for example) and anything with unusual file attributes.

Examples of ways you could search for this are:

Obfuscation and Encryption

To evade analysis, scripts and binaries are often obfuscated or packed. Common examples include:

- Base64 or gzip-encoded shell scripts
- ELF binaries packed with **UPX**
- Scripts compiled via **shc**

Full-disk or container encryption adds another layer. Adversaries may deploy **eCryptfs**, **EncFS**, or **VeraCrypt** volumes that only exist when mounted and decrypted in memory.

This is a lot harder to find and quite often drops into individual file analysis. One approach is to run **Yara** or use the **file** command and pipe to grep for the string UPX. For example:

Be aware that it can be very manual and time-consuming, unless you have good EDR-type tools deployed.

Log and Artefact Clearing

This is super-common. In most attacks, we see threat actors wiping some, if not all, of the system logs. Attacks on , , Systemd journals frequently happen. You should probably assume they are going to happen every time. Some attackers delete logs entirely; others surgically edit entries to appear legitimate. The good news is that surgical editing is slightly rarer than just wiping the log.

Selective tampering is harder to detect but more dangerous; it creates plausible-looking histories while removing incriminating context.

The frequency of this is a strong case to make sure all important log data is forwarded off the device in real-time. Do not overlook history files, though.

Disk and Memory Manipulation

Secure deletion tools (**shred**, **dd**) are frequently abused to wipe files or partitions. Others manipulate inode references directly, making recovery impossible even with carving utilities. If an attacker *shreds* the data, it is gone forever. You do not really have a feasible way to recreate it, so backups are essential. You will probably be able to detect the use of the tools, though, and often that is just enough.

Memory manipulation is also common. Libraries such as **libprocesshider** hide malicious processes from **/proc**, while kernel-level rootkits intercept calls to conceal sockets, files, and running tasks. Some even monitor for forensic tools and terminate them at runtime.

To deal with this, one approach is to capture RAM and analyse it externally to the device - but that is slightly more challenging in Linux than we might like. Another approach, and a way to threat hunt, is to compare the output of commands like ps with what is actually in /proc, or what is using threads on the kernel.

How you hunt here is entirely driven by what you are looking for, but this is an example of how you could try to compare the output of the ps command with what is actually running in /proc:

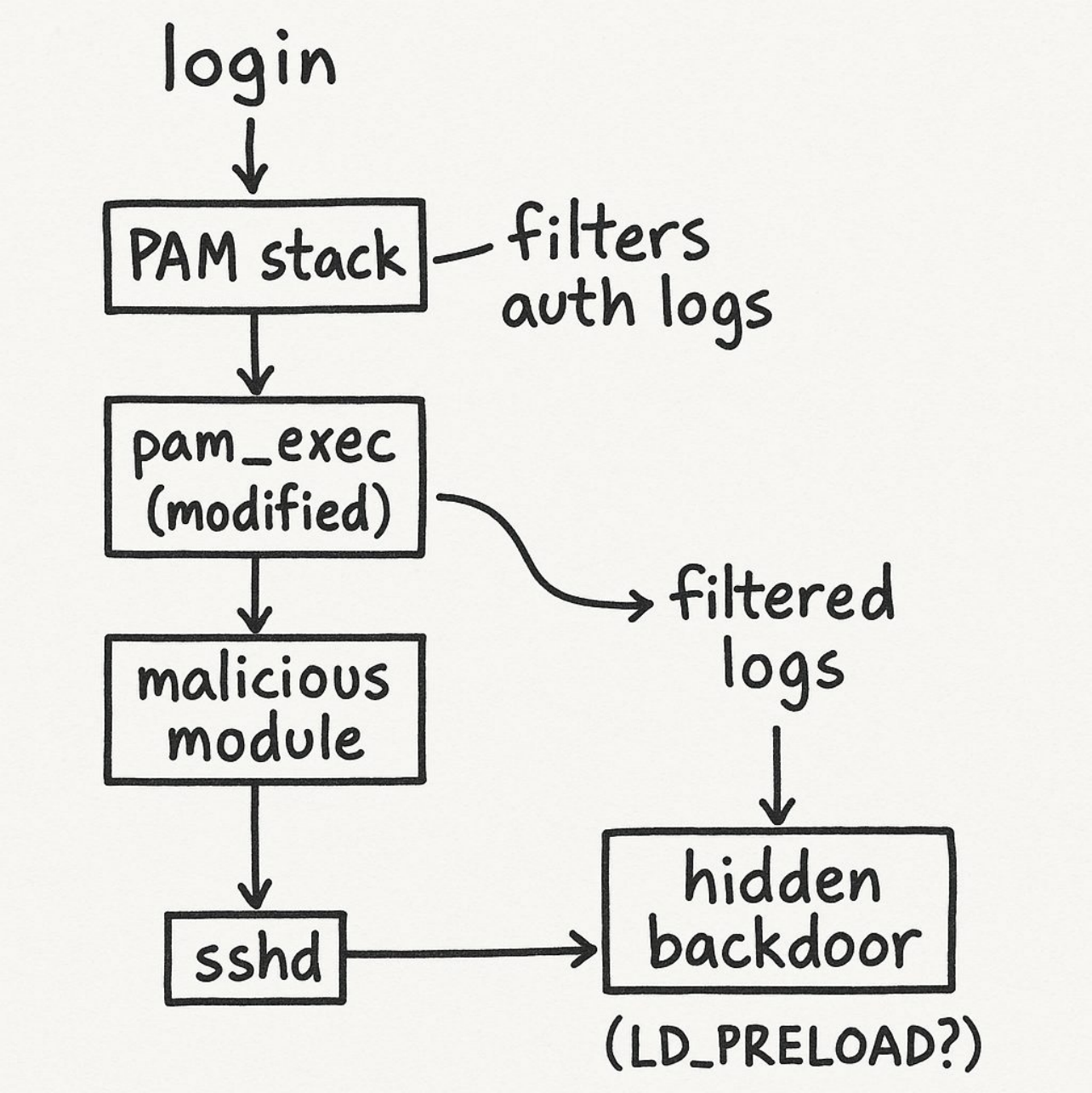
```
```

But be aware, this will not help you find things hiding from entries in /proc or if a rootkit is manipulating all the output.

Case Study: Very Common PAM Backdoors

In a recent case, attackers used a modified **PAM** (*Pluggable Authentication Module*) to maintain access to a compromised Linux host. Beyond persistence, the implant implemented anti-forensic measures, which include filtering authentication logs, overwriting timestamps, and hiding its process via .

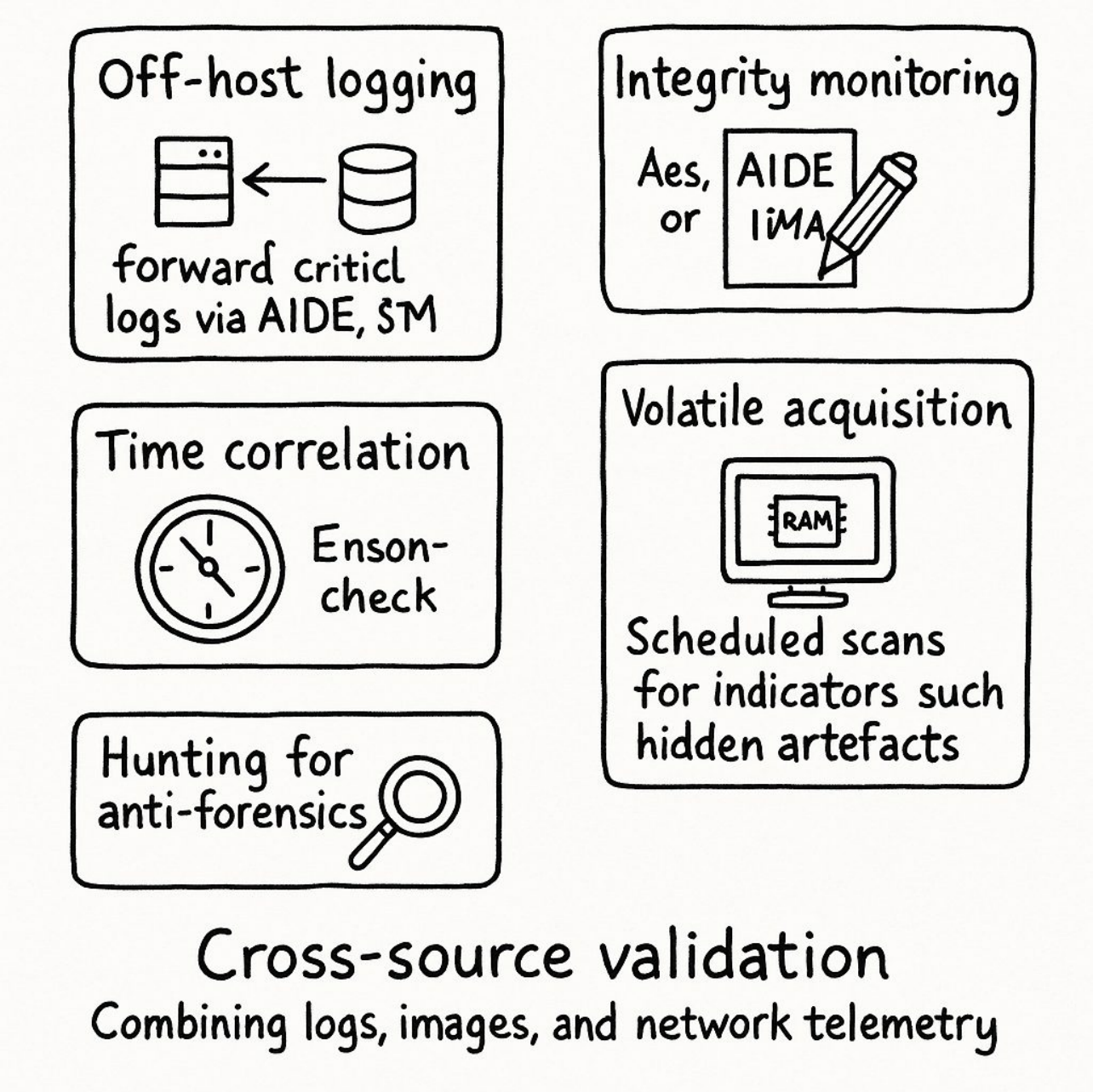
Detection only occurred when the responders performed a hash comparison against a clean baseline image, revealing subtle differences in the PAM binaries.



This should remind us that anti-forensic techniques are often embedded within persistence mechanisms rather than deployed separately.

Detection and Countermeasures

Remember, complete prevention is unrealistic, but you can reduce attacker effectiveness. Some examples of how you can do this are:



1. **Off-host logging** – forward critical logs to append-only storage or a SIEM.
2. **Integrity monitoring** – baseline binaries and configs with AIDE, Tripwire, or IMA.
3. **Time correlation** – reconstruct events using independent clocks (system, NTP, Zeek, or proxy logs).
4. **Volatile acquisition** – capture memory before shutdown; volatile artefacts often reveal decrypted volumes or hidden processes.
5. **Hunting for anti-forensics** – schedule scans for indicators such as hidden mounts, non-standard extended attributes, or modified preload libraries.

Above all, assume the local system cannot be trusted in isolation. Cross-source validation, i.e. combining logs, images, and network telemetry, is the only reliable path to truth.

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