

# Eradication and Recovery: Getting the Attacker Out and Staying Out

Mon, Sep 14, 10:00 AM EDT · <https://scottslab.io/posts/eradication-and-recovery>



## TL;DR

Eradication removes every trace of the compromise. That means finding the actual entry vector, not just the malware sitting on top of it, and checking it against the real categories of persistence attackers use. Rotate every credential the compromised account or host could touch, not just the obvious one. Recovery rebuilds or reimages. The cardinal rule: don't restore from a pre-attack image without patching the original hole, or you've recreated the door they walked through. Rebuild from a clean image is the safer default whenever persistence is confirmed or scope is uncertain. Reconstitution is the moment to add controls. When you retire media, NIST SP 800-88 Rev. 2 (2025) gives you three levels: clearing, purging, destroying, and it drops degaussing as a general answer since it does nothing to an SSD.

Containment stops the spread; eradication gets the attacker out for good. That means removing all traces of the compromise: not just the obvious malware, but the persistence they left behind, the accounts they created or hijacked, and the vulnerability they came in through. Secure the accounts, patch the holes, and scrub the artifacts. Miss one backdoor and you haven't eradicated anything; you've just given them a quieter way back.

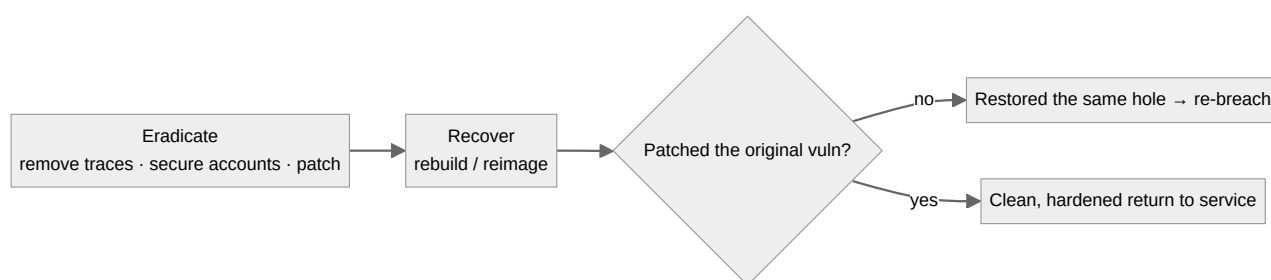
## Symptom versus cause

The most common way eradication fails is treating the malware as the incident instead of the evidence of one. Deleting a payload doesn't tell you how it got there, and if you don't know how it got there, you don't know it can't get back. Trace to the actual entry vector: the phished credential, the unpatched CVE on an internet-facing service, the exposed RDP port, the vulnerable third-party library. Fix that, not just the thing it dropped. This is where the pen-testing and detection posts in this series connect: the backdoors an attacker leaves behind are the same categories a real assessment would test for, and MITRE's ATT&CK framework catalogs them under its Persistence tactic (TA0003): scheduled tasks, new or modified accounts, registry run keys and other autostart hooks, web shells, and simply reusing valid credentials rather than planting anything detectable at all. Walk that list deliberately rather than trusting a single antivirus sweep to have caught everything; a persistence mechanism that doesn't look like malware won't get flagged as malware.

Credential rotation belongs in eradication for the same reason. It's not enough to reset the one password you know was compromised. Rotate everything that account or host could have touched: API keys, service-account passwords, SSH keys, active session tokens, and any shared secret that lived on the box. Attacker persistence increasingly lives in stolen credentials rather than planted code, precisely because credentials survive a malware scan.

## Rebuild, don't just clean, when you can't be certain

Recovery is rebuilding what was compromised, and it hides the single most common way organizations get re-breached during recovery. You rebuild or reimage the affected systems, and you must not restore from a pre-attack image without first patching the vulnerability that was originally exploited. Think about why: that clean-looking backup from last week predates the attack, but it also predates the patch, so restoring it faithfully recreates the exact weakness the attacker used. "We restored from backup" feels like recovery and is actually re-arming the trap. Restore, then patch. Or patch the image before you restore it.



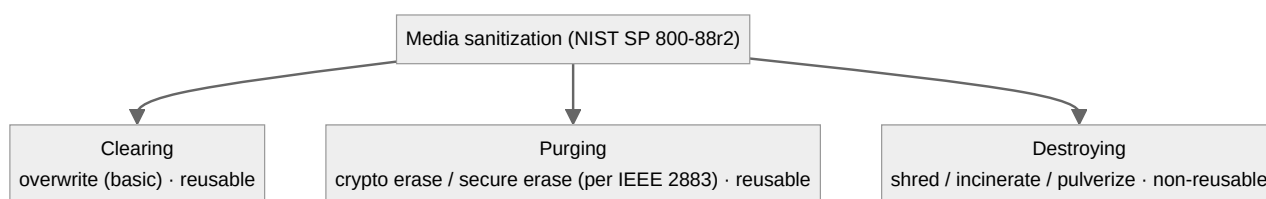
Rebuild, don't just clean, when you can't be certain

There's a real trade-off underneath "rebuild or reimage," and it's worth naming rather than pretending it's settled: cleaning a live system in place is faster and less disruptive, and for a narrow, well-understood, low-severity incident it's often the proportionate choice. But once persistence is

confirmed, or the scope of what the attacker touched isn't fully known, rebuild from a known-good image is the safer default. You can never be completely certain you found every artifact on a system where someone else had privileged access, and "probably clean" isn't the standard you want on anything that matters. Match the response to how confident you actually are, not to how much rebuilding would cost you this week.

Bringing a system back doesn't mean walking away from it. NIST is explicit that recovery should come with heightened logging and network monitoring, not a return to baseline watchfulness, for a plain reason: a resource that's been successfully attacked once is disproportionately likely to be attacked again, either by the same actor testing whether the fix held or by a different one who found the same weakness. Treat the first days back in production as an extension of the incident, not the end of it. That's the real criteria for declaring the incident closed, and it belongs to the next post in this series, not this one.

Reconstitution is bringing systems back, and it's also the best opportunity you'll get to improve: you're already touching everything and leadership is finally paying attention. For anything beyond a single host, do it in phases rather than all at once. The early phase should raise the floor immediately with quick, high-value changes over days to weeks; slower infrastructure-level changes follow once the pressure is off. It's the moment to add the controls that would have helped: application whitelisting or blacklisting, mobile device management, data loss prevention, URL filtering, updated firewall rules, and certificate revocation if keys were exposed. An incident is a terrible thing to waste; the rebuild is where "we should really do X" finally becomes "we did X."



Rebuild, don't just clean, when you can't be certain

Finally, when hardware or media from the incident gets retired, sanitize it to the right level. NIST SP 800-88 defines three. Clearing overwrites the data, which is fine for routine reuse. Purging is a stronger method, cryptographic erase or a firmware secure-erase, for higher-sensitivity data that stays in the organization; the 2025 Revision 2 of 800-88 explicitly drops degaussing as a general-purpose purge technique, because it does nothing to non-magnetic storage like an SSD, and defers technique selection to the IEEE 2883 standard instead. Destroying makes the media non-reusable, through shredding, incinerating, or pulverizing, for when the data is sensitive enough that no overwrite is trusted. Match the method to the data's sensitivity, and remember the boring failure mode this prevents: the "wiped" drive that gets resold with recoverable data still on it.