F

Fileless Malware: Techniques, Detection, and Response

Fileless malware minimizes conventional executable files by using memory, scripts, the registry, WMI, and trusted tools. Learn how to detect and respond to the behavior.

PUBLISHER
Gridinsoft Support

UPDATED
Oct 8, 2026

READ TIME
4 min

CONTENTS

In this guide

01 What fileless really means
02 Common techniques
03 Living off the land vs. malware
04 Warning signs
05 Detection strategy
06 Incident response
07 Prevention
08 Frequently asked questions

Quick answer: Fileless malware is a broad label for malicious activity that avoids or minimizes conventional executable files on disk. It may run code in memory, store scripts or payloads in the registry or WMI repository, and abuse legitimate interpreters such as PowerShell. “Fileless” rarely means that no file is involved anywhere in the attack chain; detection must focus on behavior, execution context, identity, and memory.

Kovter and Poweliks are examples discussed in connection with registry-based persistence and in-memory execution. Their behavior helps explain why removing a downloaded file may not address the mechanism that starts the code again.

01 What fileless really means

The term covers a spectrum. A phishing document may be a file but launch code directly in memory. A script can be stored in a registry value and executed by a trusted system component. An attacker with remote access may type commands manually without dropping a custom binary. Some payloads exist only in process memory until reboot.

Microsoft’s fileless threat taxonomy distinguishes in-memory, indirect file activity, and other storage and execution patterns. Precise descriptions are more useful than the label alone.

02 Common techniques

- **PowerShell and other interpreters:** download, decode, and execute commands using installed system tools.
- **WMI:** execute commands remotely or create event subscriptions that trigger persistence.
- **Registry storage:** keep encoded scripts, configuration, or payloads outside ordinary standalone files.
- **Process injection:** place code into an existing process and run it from memory.
- **Reflective loading:** load a library or executable image into memory without the normal on-disk installation path.

- **Macros and document scripts:** use an initial file to launch memory-resident or living-off-the-land stages.
- **Shared memory and temporary runtime locations:** stage code outside expected persistent application directories.

03 Living off the land vs. malware

PowerShell, WMI, command shells, scripting engines, and remote-management tools are legitimate. Their presence is not an IoC by itself. Maliciousness comes from who invoked them, the parent process, arguments, encoded content, destination, privilege, timing, and resulting actions.

Blocking every interpreter can break administration and applications. Prefer constrained use, signed scripts, application control, least privilege, and monitoring tuned to business baselines.

04 Warning signs

- an Office application or browser launches a shell or scripting interpreter;
- long encoded commands, hidden windows, download-and-execute patterns, or unusual parent-child processes;
- new WMI event consumers, filters, bindings, or suspicious repository content;
- registry values containing large encoded scripts or commands that run at startup;
- memory protection changes, injection, or executable regions without a matching trusted file;
- legitimate administration tools used by unexpected accounts, hosts, or time windows;
- network connections from processes that normally do not communicate externally.

05 Detection strategy

Collect process creation with command lines, script-block and interpreter logs where supported, WMI activity, registry changes, memory and injection telemetry, identity events, DNS, and network connections. Correlate a chain rather than alerting on one keyword. A document

spawning PowerShell that decodes content, changes a run key, and contacts a rare domain is stronger evidence than PowerShell alone.

Use behavior-based endpoint protection, attack-surface-reduction rules, and application control. Protect logs from local tampering and centralize them quickly because volatile evidence disappears after process exit or reboot.

06 Incident response

1. **Isolate the host.** Use managed endpoint isolation when possible while retaining power for memory collection.
2. **Capture volatile evidence.** Qualified responders should collect memory, processes, network state, sessions, and loaded modules before reboot.
3. **Preserve persistence stores.** Export relevant registry, WMI, scheduled task, service, script, and event-log evidence.
4. **Trace the chain backward.** Find the email, exploit, account, remote session, document, or web event that launched activity.
5. **Determine scope.** Hunt for the same identities, command patterns, WMI objects, registry data, destinations, and behavior across systems.
6. **Rebuild when needed.** If privileged access or unknown in-memory persistence makes integrity uncertain, use a trusted reimage and rotate affected credentials.

07 Prevention

Patch applications and operating systems, restrict macros and internet-origin scripts, and remove local administrator rights from routine users. Require phishing-resistant MFA for remote and privileged access. Use PowerShell security features, constrained modes where appropriate, script signing, transcription and logging, and narrow remoting permissions.

Audit WMI and registry persistence locations, protect endpoint controls from tampering, and segment administration. Test policies in audit mode before enforcement so defenders can distinguish approved automation from abuse.

08 Frequently asked questions

Can antivirus detect fileless malware?

Modern endpoint tools can inspect scripts, memory, behavior, and command chains, but no tool detects every variation. Logging and incident context remain important.

Does rebooting remove it?

It may remove purely volatile code, but registry, WMI, tasks, stolen credentials, or remote access can restore execution. Rebooting can also erase evidence.

Is all PowerShell activity suspicious?

No. PowerShell is a legitimate administration tool. Investigate unusual context, arguments, identity, parent processes, destinations, and outcomes.

NEED MORE HELP?

We are here when the guide is not enough.

Visit the Gridinsoft Support Center for current product guidance or submit a support request with the details of your case.

Open this guide online at support.gridinsoft.com