macOS Persistence Cheatsheet

v1.0 (07/04/2026)

DFIR artifacts, artifact scope, macOS version applicability, minimum privilege required to establish or modify persistence, false-positive risk, baseline dependence, and review guidance.

Scope	Source of Truth	Signal	Required Privilege	False Positive Risk
User · System · Mixed	Disk · State · Both	High · Medium · Noisy · Context	User · Admin · Root · Approval · Mixed	Low FP · Medium FP · High FP

Overview

Mechanism scan

31 visible · 31 total mechanisms

Quick index of the persistence mechanisms covered below. The search box and legend filters apply here too, so this view stays in sync with the tables.

Category

Core Launch & Login Persistence

6 / 6

Launch Agents [S1]
High · Mixed · Mixed · 10.4+
Trigger: User login / GUI session

Launch Daemons [S2]
High · Root · System · All
Trigger: Boot / pre-login

launchd overrides / disabled state [S3]
Medium · Mixed · Mixed · All
Trigger: Persistent state across reboot

BTM / SMAppService [S4]
Medium · Mixed · Mixed · 13+
Trigger: User login / helper startup

Login Items (legacy / classic) [S4a]
Medium · User · User · Legacy / pre-13
Trigger: User login

Embedded Login Helpers [S5]
Noisy · User · User · 10.6.6+
Trigger: Login / app-managed helper start

Category

Shell, Access, and Scheduled Execution

7 / 7

Shell init (zsh) [S7]
Medium · Mixed · Mixed · Current default
Trigger: Login shell / interactive shell / logout

Shell init (bash / sh) [S8]
Medium · Mixed · Mixed · Still present
Trigger: Login shell / interactive shell

SSH authorized_keys [S9]
High · Mixed · Mixed · All
Trigger: SSH connection

Cron [S10]
Medium · Mixed · Mixed · All
Trigger: Schedule / periodic

At / atrun [S11]
Context · Mixed · Mixed · All
Trigger: One-shot deferred execution

Periodic Jobs [S12]
Context · Root · System · All
Trigger: Daily / weekly / monthly maintenance

Emond [S13]
High · Root · System · ≤12; removed in 13
Trigger: Event-driven (startup, auth, policy)

Category

Extensions, Browser, Interpreter, and Handler Surfaces

8 / 8

System Extensions [S14]
Medium · Admin · System · 10.15+
Trigger: Boot / activation / app-managed load

KEXTs [S15]
High · Root · System · Legacy; deprecated 10.15+, tighter on 11+
Trigger: Boot / extension load

Chromium Extensions [S16]
Noisy · Mixed · Mixed · All
Trigger: Browser launch / profile load

Safari App Extensions [S17]
Medium · User · Mixed · Safari 10 / 10.12+
Trigger: Browser launch / extension activation

Safari Web Extensions [S17a]
Medium · User · Mixed · Safari 14+
Trigger: Browser launch / extension activation

Application / daemon plug-ins [S17b]
Context · Mixed · Mixed · All
Trigger: Host app / daemon launch or plug-in discovery

Python Startup Hooks [S18]
Context · Mixed · Mixed · All
Trigger: Python interpreter start

LaunchServices / URL Handlers [S19]
Context · User · User · All
Trigger: File open / URL scheme invocation

Category

Persistence-Adjacent & Legacy Mechanisms

8 / 8

Installer Packages [S20]
Context · Admin · Mixed · All
Trigger: Package installation

DYLD_* injection / LSEnvironment [S20a]
Medium · Mixed · Mixed · All; constrained on modern macOS
Trigger: Target process launch

Application / binary modification [S20b]
Medium · Mixed · Mixed · All
Trigger: Host app / binary execution

LC_LOAD_DYLIB Addition [S21]
Medium · Mixed · Mixed · All
Trigger: Trusted binary execution

Dylib Hijacking / Proxying [S21a]
Medium · Mixed · Mixed · All; constrained on hardened runtime
Trigger: Host app launch / dynamic load

Login Hooks [S22]
High · Mixed · Mixed · Deprecated since 10.11
Trigger: Login / logout

Startup Items [S23]
High · Root · System · Legacy; mainly pre-10.4 compatibility
Trigger: Boot

RC Scripts [S24]
High · Root · System · <=10.3
Trigger: Boot / init path

Category

Configuration Profiles & MDM

2 / 2

Configuration Profiles (.mobileconfig) [S25]
Medium · Mixed · Mixed · All
Trigger: Profile install / MDM push / managed pref load

High-impact MDM payloads [S25a]
High · Mixed · Mixed · All; payload set evolves per release
Trigger: Profile install / payload activation

01 Core Launch & Login Persistence

Mechanism	Required privilege	Source of Truth	Signal	Collection / Triage	Trigger	What to review
Launch Agents [S1] Mixed · 10.4+ Core current	Mixed	Both	High	Artifact / Path: `/System/Library/LaunchAgents`; `/Library/LaunchAgents`	User login / GUI session	Medium FP · Baseline: Critical · Use: Autostart — Parse Label, Program, ProgramArguments, RunAtLoad, KeepAlive, WatchPaths, QueueDirectories,

				~/Library/LaunchAgents **Resolves to** Program / ProgramArguments[0] Referenced MachServices / Sockets if present **Enumerate** launchctl print gui/<uid> (Copy) find ~/Library/LaunchAgents /Library/LaunchAgents -maxdepth 1 -name '*.plist' (Copy) **Inspect** launchctl print gui/<uid>/<label> (Copy) launchctl blame gui/<uid>/<label> (Copy) plutil -p <plist> (Copy)		StartCalendarInterval, PathState, Sockets, and MachServices. Use `launchctl blame` to identify the actual trigger that loaded a service: this distinguishes a job started by RunAtLoad from one started by WatchPaths, Sockets, or MachServices, which matters for hunting on-demand persistence. Prioritize targets outside Apple or vendor-controlled paths and jobs present on disk but absent or odd in live launchctl state. Scope and required privilege vary by path: user-scoped in ~/Library, system-scoped in /Library.
Launch Daemons [S2] System · All Core current	Root	Both	High	**Artifact / Path** /System/Library/LaunchDaemons /Library/LaunchDaemons **Resolves to** Program / ProgramArguments[0] Referenced service endpoints and loaded binary path **Enumerate** launchctl print system (Copy) find /Library/LaunchDaemons -maxdepth 1 -name '*.plist' (Copy) **Inspect** launchctl print system/<label> (Copy) launchctl blame system/<label> (Copy) plutil -p <plist> (Copy)	Boot / pre-login	Low FP · Baseline: Critical · Use: Autostart Very high-value persistence. Review launchd keys such as RunAtLoad, KeepAlive, StartCalendarInterval, and PathState, plus ownership, signer, Team ID, and whether the target binary lives in a user-writable or otherwise weak path. Use `launchctl blame` on the live service-target to surface the real trigger (RunAtLoad vs Sockets / MachServices / WatchPaths) instead of relying on plist intent alone.
launchd overrides / disabled state [S3] Mixed · All Storage varies	Mixed	State	Medium	**Artifact / Path** Persistent disabled / override state **Resolves to** Effective enabled / disabled state for a label Final live state shown by launchctl, not just plist intent **Enumerate** launchctl print-disabled system (Copy) launchctl print-disabled gui/<uid> (Copy) **Inspect** launchctl print system/<label> (Copy) launchctl print gui/<uid>/<label> (Copy)	Persistent state across reboot	Medium FP · Baseline: Helpful · Use: State control All versions; storage details vary. Do not trust plist review alone: disabled or override state can persist independently of the plist, and the exact storage location has changed over time.
BTM / SMAppService [S4] Mixed · 13+ Ventura+ / Background Task Management	Mixed	Both	Medium	**Artifact / Path** ~/Library/Application Support/com.apple.backgroundtaskmanagementagent/backgrounditems.btm (legacy / pre-Ventura) /private/var/db/com.apple.backgroundtaskmanagement/BackgroundItems-v*.btm (Ventura+, versioned BTM store) **Resolves to** Registered helper binary or app bundle in BTM Owning app bundle, Team ID, and registration record	User login / helper startup	Medium FP · Baseline: Critical · Use: Login helper macOS 13+ Background Task Management / SMAppService model. Review BTM state and on-disk records together, then correlate each entry with the owning app bundle, signer, Team ID, and the helper actually present on disk. Treated here as mixed scope because modern SMAppService can register login items as well as launch-item style services.

				Enumerate `sfltool dumpbtm` Copy Inspect `plutil -p ~/Library/Application\ Support/com.apple.backgroundtaskmanagementagent/backgrounditems.btm` Copy `codesign -dv --verbose=4 <helper_or_owner_app>` Copy		

Login Items (legacy / classic)
[S4a]
User · Legacy / pre-13
Pre-BTM login items
	User	Both	Medium	Artifact / Path
				`Legacy login item records / shared file list state`
				`Resolved app or helper path launched at user login`
				Resolves to
				`App bundle or helper registered for login`
				`Owning app, signer, Team ID, and user-scoped registration`
				Enumerate
				`osascript -e 'tell application "System Events" to get the properties of every login item'` Copy
				`plutil -p <legacy_login_item_store_when_present>` Copy
				Inspect
				`codesign -dv --verbose=4 <login_item_or_owner_app>` Copy
				`ls -la <resolved_login_item_path>` Copy
					User login	Medium FP · Baseline: Critical · Use: Login helper
						Legacy / classic login items predate BTM / SMAppService. Backing files and exact storage vary by OS generation, so verify on the target OS and map each registration to the real app or helper on disk.

Embedded Login Helpers
[S5]
User · 10.6.6+
superseded by SMAppService on 13+
	User	Both	Noisy	Artifact / Path
				`<App>.app/Contents/Library/LoginItems`
				Resolves to
				`Embedded helper app / binary`
				`Owner app bundle, signer, Team ID, and expected install location`
				Enumerate
				`find /Applications ~/Applications -path '*/Contents/Library/LoginItems/*'` Copy
				Inspect
				`codesign -dv --verbose=4 <helper>` Copy
				`plutil -p <owner_app>/Contents/Info.plist` Copy
					Login / app-managed helper start	High FP · Baseline: Critical · Use: Helper launch
						Common in legitimate software too. Treat as modern app-bundled persistence and baseline Bundle ID, Team ID, signer, and owner app. Focus on signer mismatches, stale paths, and helpers embedded in unusual app locations.

02 Shell, Access, and Scheduled Execution

Mechanism	Required privilege	Source of Truth	Signal	Collection / Triage	Trigger	What to review

Shell init (zsh)
[S7]
Mixed · Current default
Default on current macOS
	Mixed	Disk	Medium	Artifact / Path
				`/etc/zshenv`
				`/etc/zprofile`
				`/etc/zshrc`
				`/etc/zlogin`
				`/etc/zlogout`
				`~/.zshenv`
				`~/.zprofile`
				`~/.zshrc`
				`~/.zlogin`
				`~/.zlogout`
				Resolves to
					Login shell / interactive shell / logout	High FP · Baseline: Helpful · Use: Shell exec
						zsh is the default shell on current macOS, but not exclusive to a single release boundary. Resolve ZDOTDIR before concluding nothing changed, or you can miss relocated per-user dotfiles and hidden sourced files.

Commands sourced by zsh startup files

ZDOTDIR-resolved files and downstream scripts / binaries

Enumerate

```
echo $ZDOTDIR
```
Copy

```
ls -la ~/.z* /etc/z*
```
Copy

Inspect

```
grep -nEv '^\s*(#|$)' ~/.zshenv ~/.zprofile
~/.zshrc ~/.zlogin ~/.zlogout /etc/zshenv
/etc/zprofile /etc/zshrc /etc/zlogin
/etc/zlogout
```
Copy

Shell init (bash / sh)

[S8]

Mixed · Still present

Common on older / migrated profiles

Mixed | Disk | Medium

Artifact / Path

```
/etc/profile
/etc/bashrc
~/.bash_profile
~/.bash_login
~/.bashrc
~/.bash_logout
~/.profile
```

Resolves to

Commands sourced by bash / sh startup files

Referenced scripts, PATH entries, and ENV / BASH_ENV targets

Enumerate

```
env | grep -E 'PATH|ENV|BASH_ENV'
```
Copy

```
ls -la ~/.bash* ~/.profile /etc/profile
/etc/bashrc
```
Copy

Inspect

```
grep -nEv '^\s*(#|$)' ~/.bash_profile
~/.bash_login ~/.bashrc ~/.bash_logout
~/.profile /etc/profile /etc/bashrc
```
Copy

Login shell / interactive shell

High FP · Baseline: Helpful · Use: Shell exec

bash remains present and is still common on older, migrated, admin, CI, and developer estates. Look for hidden sourcing, PATH abuse, curl or osascript launchers, and suspicious environment exports.

SSH authorized_keys

[S9]

Mixed · All

Access persistence

Mixed | Disk | High

Artifact / Path

```
~/.ssh/authorized_keys
~/.ssh/config
/etc/ssh/sshd_config
```

Resolves to

Authorized key entries actually accepted by sshd

AuthorizedKeysFile path(s), command= restrictions, principals, and signer context

Enumerate

```
grep -i '^AuthorizedKeysFile'
/etc/ssh/sshd_config
```
Copy

```
ls -la ~/.ssh
```
Copy

Inspect

```
ssh-keygen -lf ~/.ssh/authorized_keys
```
Copy

```
grep -n '' ~/.ssh/authorized_keys
/etc/ssh/sshd_config
```
Copy

SSH connection

Medium FP · Baseline: Helpful · Use: Access persistence

Access persistence rather than local autostart. High signal on privileged or shared admin accounts. Also verify AuthorizedKeysFile in sshd_config so you do not miss a custom key path. Treated as mixed scope because key material is commonly user-scoped while AuthorizedKeysFile policy may be system-scoped.

Cron

[S10]

Mixed · All

Less native

Mixed | Disk | Medium

Artifact / Path

```
/etc/crontab
/usr/lib/cron/tabs/
```

Resolves to

Command field and invoked script path

Schedule / periodic

Medium FP · Baseline: Helpful · Use: Scheduled exec

More common on older or mixed estates than on launchd-first modern deployments. macOS-specific quirk: per-user crontabs live in `/usr/lib/cron/tabs/`, not `/var/cron/tabs` as on most

Per-user or system crontab entry actually executed

Enumerate

```
crontab -l
```

Copy

```
sudo ls -la /usr/lib/cron/tabs/ /etc/crontab
```

Copy

```
for u in $(dscl . list /Users | grep -v '^_');
do sudo crontab -u "$u" -l 2>/dev/null && echo "
[$u]"; done
```

Copy

Inspect

```
sudo grep -nEv '^\s*(#|$)' /etc/crontab
/usr/lib/cron/tabs/*
```

Copy

BSD/Linux references. `crontab -l` only shows the current user's jobs, so iterate over local accounts (or list the spool directly with root) to avoid missing entries. Verify targets for user-writable paths, temp directories, or masquerading names.

At / atrun

[S11]

Mixed · All

Rare; atrun system daemon, per-user job queues

Mixed | Both | Context

Artifact / Path

```
at queue / spool
```

```
at.allow / at.deny
```

Resolves to

```
Queued command / script in the at job
```

```
atrun-driven one-shot execution target
```

Enumerate

```
atq
```

Inspect

```
at -c <job>
```

Copy

```
grep -n '' /etc/at.allow /etc/at.deny
```

Copy

One-shot deferred execution

Low FP · Baseline: Optional · Use: Scheduled exec

Largely theoretical on default macOS: `/usr/libexec/atrun` is disabled out of the box, so submitted at jobs do not actually fire until an attacker (or admin) enables it with root via

```
sudo launchctl load -w
/System/Library/LaunchDaemons/com.apple.atrun.plist
```

. Treat any host where atrun is loaded and at jobs are queued as a strong, version-independent signal. Prefer command-based review (`atq` , `at -c <job>`) over a single hardcoded spool path and focus on jobs that launch from temp, shared, hidden, or user-writable locations.

Periodic Jobs

[S12]

System · All

System scheduler

Root | Disk | Context

Artifact / Path

```
/etc/periodic/daily
```

```
/etc/periodic/weekly
```

```
/etc/periodic/monthly
```

```
/etc/defaults/periodic.conf
```

```
/etc/periodic.conf (override; absent by default)
```

Resolves to

```
Script executed from periodic dirs
```

```
Any custom path introduced through periodic.conf
```

Enumerate

```
ls -la /etc/periodic/daily /etc/periodic/weekly
/etc/periodic/monthly
```

Copy

Inspect

```
grep -R '.' /etc/periodic
/etc/defaults/periodic.conf /etc/periodic.conf
2>/dev/null
```

Copy

Daily / weekly / monthly maintenance

Low FP · Baseline: Helpful · Use: Scheduled exec

Low-frequency but worth keeping for exhaustive modern macOS triage. Note that `/etc/defaults/periodic.conf` is the shipped default; `/etc/periodic.conf` does not exist by default and is only read if created as a local override, so its mere presence is itself a signal worth investigating. Review inherited script paths, shellouts, and permissions rather than assuming stock maintenance content.

Emond

[S13]

System · ≤12; removed in 13

Removed in Ventura

Root | Disk | High

Artifact / Path

```
/etc/emond.d/rules/
```

```
/private/var/db/emondClients
```

```
/System/Library/LaunchDaemons/com.apple.emond.plist
```

Resolves to

```
Action command or script referenced by the rule
```

```
Condition-to-action chain that leads to code execution
```

Enumerate

```
ls -la /etc/emond.d/rules
/private/var/db/emondClients
```

Copy

Inspect

Event-driven (startup, auth, policy)

Low FP · Baseline: Optional · Use: Event-driven

Apple removed emond entirely in macOS 13 Ventura: the `/sbin/emond` binary and `/System/Library/LaunchDaemons/com.apple.emond.plist` no longer ship. On any modern fleet (13+), the presence of `/etc/emond.d/rules/*` , `/private/var/db/emondClients` , or any emond-related plist is itself a strong signal because the mechanism does not exist natively. On ≤12 hosts, treat unexplained rules, clients, or odd actions as immediate priority.

				`plutil -p /System/Library/LaunchDaemons/com.apple.emond.plist` Copy `grep -R '.' /etc/emond.d/rules` Copy		

03 Extensions, Browser, Interpreter, and Handler Surfaces

Mechanism	Required privilege	Source of Truth	Signal	Collection / Triage	Trigger	What to review

System Extensions
[S14]
System · 10.15+
Current ext model
	Admin
	Approval
		State	Medium	**Artifact / Path**
				`Contents/Library/SystemExtensions`
				`Owning app bundle`
				`Current activation state`
				Resolves to
				`Activated system extension and owning app`
				`Bundle ID, Team ID, and activation state shown by systemextensionsctl`
				Enumerate
				`systemextensionsctl list` Copy
				Inspect
				`codesign -dv --verbose=4 <owning_app>` Copy
				`systemextensionsctl list | grep -E 'enabled|active'` Copy
					Boot / activation / app-managed load	Medium FP · Baseline: Critical · Use: Extension / plugin
						macOS 10.15+ current model. Use state-first enumeration. Baseline Team ID, Bundle ID, and activation status, then verify the owning app, signer, and approval context.

KEXTs
[S15]
System
Legacy; deprecated 10.15+, tighter on 11+
Superseded by System Extensions
	Root
	Approval
		Both	High	**Artifact / Path**
				`/Library/Extensions`
				Resolves to
				`Loaded kext bundle ID and path`
				`Actual loaded extension plus the on-disk bundle`
				Enumerate
				`kmutil showloaded` Copy
				`kextstat`
				Inspect
				`codesign -dv --verbose=4 <kext>` Copy
				`kmutil inspect -b <bundle_id>` Copy
					Boot / extension load	Low FP · Baseline: Critical · Use: Extension / plugin
						Legacy mechanism. Kernel extensions are deprecated from macOS 10.15 onward in favor of system extensions, and their use is more restricted on 11+. Presence on a modern fleet is high-signal and often implies exception or reduced-security context.

Chromium Extensions
[S16]
Mixed · All
Chromium only
	Mixed	Both	Noisy	**Artifact / Path**
				`~/Library/Application Support/Google/Chrome/*/Extensions`
				`~/Library/Application Support/Google/Chrome/*/Preferences`
				`/Library/Managed Preferences/com.google.Chrome.plist`
				Resolves to
				`Extension ID, manifest, and effective policy source`
				`Force-installed or managed extension config tied to a real profile`
				Enumerate
				`find ~/Library/Application\ Support/Google/Chrome -path '*/Extensions/*'` Copy
				`plutil -p /Library/Managed\ Preferences/com.google.Chrome.plist 2>/dev/null | grep -E 'ExtensionInstallForcelist|ExtensionSettings'` Copy
				Inspect
				`grep -n 'extensions' ~/Library/Application\ Support/Google/Chrome/*/Preferences` Copy
				`codesign -dv --verbose=4 <extension_host_app_if_app_mode>` Copy
					Browser launch / profile load	High FP · Baseline: Critical · Use: Extension / plugin
						All versions with Chromium-based browsers; noisy on managed fleets. Review extension IDs, policy-forced installs, managed profiles, update URLs, and any extension granted unusual permissions or loaded from a nonstandard path. Treated as mixed scope because this row covers both per-profile installs and managed / policy-backed installs.

Safari App Extensions

[S17]

Mixed · Safari 10 / 10.12+

Safari 10+ app extension model

User · Approval

Both

Medium

Artifact / Path

- `Owning app bundle / appex`
- `Safari extension state / preferences`

Resolves to

- `appex / owner app bundle actually providing the extension`
- `Stored extension state tied to the installed owner app`

Enumerate

```
pluginkit -mAv | grep -i safari
```
Copy

```
find /Applications ~/Applications -name '*.appex'
```
Copy

Inspect

```
codesign -dv --verbose=4 <appex_or_owner_app>
```
Copy

```
find ~/Library/Containers/com.apple.Safari/Data/Library/Preferences -name "*.plist" -exec plutil -p {} \; 2>/dev/null
```
Copy

Browser launch / extension activation

Medium FP · Baseline: Helpful · Use: Extension / plugin

Safari App Extensions are the older app/appex-backed model, introduced with Safari 10 (macOS 10.12 Sierra; also available as a Safari 10 update on 10.11.6 El Capitan). Review the owner app, signer, Team ID, and whether stored state matches the app actually present on disk. Treated as mixed scope because extension state is user-specific while the owning app may live in either user or global application space.

Safari Web Extensions

[S17a]

Mixed · Safari 14+

Current Safari web extension model

User · Approval

Both

Medium

Artifact / Path

- `Owning app bundle / web extension payload`
- `Safari extension state / preferences`

Resolves to

- `Safari web extension actually providing the browser code`
- `Stored enablement / permissions tied to the installed owner app`

Enumerate

```
pluginkit -mAv | grep -i safari
```
Copy

```
find /Applications ~/Applications -name '*.appex'
```
Copy

Inspect

```
codesign -dv --verbose=4 <appex_or_owner_app>
```
Copy

```
find ~/Library/Containers/com.apple.Safari/Data/Library/Preferences -name "*.plist" -exec plutil -p {} \; 2>/dev/null
```
Copy

Browser launch / extension activation

Medium FP · Baseline: Helpful · Use: Extension / plugin

Safari Web Extensions are the current model on Safari 14+. Review the owner app, signer, Team ID, extension permissions, and whether the stored state matches the installed app and expected browser profile. Treated as mixed scope because extension state is user-specific while the owning app may live in either user or global application space.

Application / daemon plug-ins

[S17b]

Mixed · All

Stealthy host-loaded plug-ins

Mixed

Both

Context

Artifact / Path

- `App / daemon-specific plug-in directories`
- `/Library/Security/SecurityAgentPlugins/ (authorization plug-ins)`
- `/Library/DirectoryServices/PlugIns/ (Open Directory)`
- `~/Library/iTunes/iTunes Plug-ins (historical; iTunes removed in 10.15)`
- `~/Library/QuickLook / /Library/QuickLook`
- `/Library/Spotlight`

Resolves to

- `Actual plug-in bundle loaded by the host app or daemon`
- `Owning host, signer, and execution context inherited from that host`

Enumerate

```
pluginkit -mAv
```
Copy

```
qlmanage -m plugins 2>/dev/null
```
Copy

```
mdimport -L 2>/dev/null
```
Copy

Inspect

```
codesign -dv --verbose=4 <plugin_bundle>
```
Copy

Host app / daemon launch or plug-in discovery

Medium FP · Baseline: Helpful · Use: Extension / plugin

Covers non-browser plug-in persistence such as app-specific plug-ins and daemon plug-ins. Two especially sensitive subclasses deserve dedicated attention: **authorization plug-ins** in `/Library/Security/SecurityAgentPlugins/` (loaded by `authorizationhost` / `SecurityAgent`, can intercept credential prompts and run very early in the auth chain) and **Open Directory plug-ins** in `/Library/DirectoryServices/PlugIns/` (loaded by `opendirectoryd`, root-context, very high impact). For all rows, focus on writable plug-in directories, unexpected host / signer pairings, and bundles that inherit sensitive host privileges or sandbox exceptions. Treated as mixed scope because plug-ins may live in user-writable or system-wide host directories.

				`find ~/Library /Library -type d \( -name '*Plug-Ins*' -o -name 'QuickLook' -o -name 'Spotlight' \) 2>/dev/null` Copy		

Python Startup Hooks
[S18]
Mixed · All
Dev-heavy
	Mixed	Disk	Context	**Artifact / Path**
				`.pth`
				`sitecustomize.py`
				`usercustomize.py`
				Resolves to
				`Imported module or code run on interpreter start`
				`.pth executable lines, sitecustomize, or usercustomize targets`
				Enumerate
				`python3 -m site` Copy
				`find ~/Library /usr/local /opt/homebrew \( -name '*.pth' -o -name 'sitecustomize.py' -o -name 'usercustomize.py' \) 2>/dev/null` Copy
				Inspect
				`grep -n '' <.pth_or_customize_file>` Copy
					Python interpreter start	High FP · Baseline: Helpful · Use: Interpreter hook
						Most relevant on dev, automation, and build hosts. Hunt for executable import lines in .pth files, unexpected site-package directories, and hooks that silently reach out to the network or launch other interpreters. Treated as mixed scope because this row covers both user-specific and system-level Python locations.

LaunchServices / URL Handlers
[S19]
User · All
Indirect / stateful
	User	Both	Context	**Artifact / Path**
				`LSHandlers`
				`Custom scheme / file-handler associations`
				`~/Library/Preferences/com.apple.LaunchServices/com.apple.launchservices.secure.plist`
				Resolves to
				`Handler Bundle ID and actual registered app`
				`Custom scheme or file association that causes the execution path`
				Enumerate
				`/System/Library/Frameworks/CoreServices.framework/Frameworks/LaunchServices.framework/Support/lsregister -dump` Copy
				Inspect
				`plutil -p ~/Library/Preferences/com.apple.LaunchServices/com.apple.launchservices.secure.plist` Copy
					File open / URL scheme invocation	High FP · Baseline: Helpful · Use: Handler hijack
						All versions; preference and registered-state storage varies. Lower-priority and indirect, but useful when persistence appears to hinge on a document opener or custom URL scheme workflow.

04 Persistence-Adjacent & Legacy Mechanisms

These stay in the sheet because they matter in DFIR, but they are not primary autostart mechanisms in the same sense as launchd, BTM, login helpers, or shell startup.

Mechanism	Required privilege	Source of Truth	Signal	Collection / Triage	Trigger	What to review

Installer Packages
[S20]
Mixed · All
Dropper-adjacent
	Admin	Disk	Context	**Artifact / Path**
				`preinstall`
				`postinstall`
				Resolves to
				`preinstall / postinstall script target`
				`Dropped payload or persistence artifact created by the package`
				Enumerate
				`pkgutil --pkgs` Copy
				Inspect
				`pkgutil --expand <pkg> <dir>` Copy
					Package installation	High FP · Baseline: Helpful · Use: Installer / dropper
						Usually a deployment or dropper surface rather than final persistence. Review because it often lays down LaunchDaemons, helpers, login items, or extensions. Scope is treated as mixed because packages are a deployment surface that can lay down either user- or system-scoped persistence.

				`spctl --assess --type install <pkg>` Copy		

DYLD_* injection / LSEnvironment
[S20a]
Mixed · All; constrained on modern macOS
Dynamic library environment injection
	Mixed	Disk	Medium	**Artifact / Path**
				`LSEnvironment in app Info.plist`
				`EnvironmentVariables in launchd plists`
				`DYLD_INSERT_LIBRARIES`
				`DYLD_FRAMEWORK_PATH`
				Resolves to
				`Injected dylib path and targeted host process`
				`Plist or app bundle whose launch path causes the dylib to load`
				Enumerate
				`grep -R -n 'DYLD_|LSEnvironment|EnvironmentVariables' /Applications ~/Applications ~/Library/LaunchAgents /Library/LaunchAgents /Library/LaunchDaemons 2>/dev/null` Copy
				Inspect
				`plutil -p <plist_or_Info.plist>` Copy
				`codesign -d --entitlements :- <target_app_or_binary> 2>/dev/null` Copy
					Target process launch	Medium FP · Baseline: Helpful · Use: Execution hijack
						Separate from Mach-O load-command tampering. On modern macOS, the dynamic loader ignores DYLD_* environment variables for Apple platform binaries and for third-party apps built with the hardened runtime, unless the target carries one of two specific entitlements: `com.apple.security.cs.allow-dyld-environment-variables` or `com.apple.security.cs.disable-library-validation`. The fastest discriminating check is therefore `codesign -d --entitlements :- <app>` on the target host process: a hardened-runtime app exposing either entitlement is the realistic injection target (this is the class of weakness behind the Zoom dylib injection cases). Treated as mixed scope because the persistence artifact may be a user app / plist or a system-wide app / launch item.

Application / binary modification
[S20b]
Mixed · All
Host app subversion / infected binary
	Mixed	Disk	Medium	**Artifact / Path**
				`Modified app bundle executable or resources`
				`Trojanized or infected Mach-O host binary`
				Resolves to
				`Host app or binary that now re-executes malicious code`
				`Tampered code signature, loader chain, or embedded persistence logic`
				Enumerate
				`codesign --verify --deep --strict <app_or_binary>` Copy
				`spctl --assess --type execute <app_or_binary>` Copy
				Inspect
				`codesign -dv --verbose=4 <app_or_binary>` Copy
				`otool -l <app_or_binary>` Copy
				`shasum -a 256 <app_or_binary>` Copy
					Host app / binary execution	Medium FP · Baseline: Helpful · Use: Execution hijack
						Broader than added load commands alone. Use this row for trojanized apps, patched bundle contents, and infected binaries that silently restore or re-establish other persistence when the host is launched. Treated as mixed scope because the tampered host may live in user-writable or system-wide locations.

LC_LOAD_DYLIB Addition
[S21]
Mixed · All
Binary-level
	Mixed	Disk	Medium	**Artifact / Path**
				`Mach-O load commands`
				Resolves to
				`Injected / added dylib path`
				`Impacted Mach-O binary that loads the dylib at runtime`
				Enumerate
				`otool -L <binary>` Copy
				Inspect
				`otool -l <binary>` Copy
				`codesign -dv <binary>` Copy
					Trusted binary execution	Medium FP · Baseline: Helpful · Use: Execution hijack
						Persistence-adjacent binary tampering. Focus on unexpected dylibs from user-writable paths, hidden directories, or signers that do not fit the host baseline. Treated as mixed scope because the modified host binary may live in user-writable or system-wide locations.

Dylib Hijacking / Proxying

[S21a]

Mixed | All; constrained on hardened runtime

Vulnerable search paths and LC_REEXPORT_DYLIB proxies

Mixed | Disk | Medium

Artifact / Path

- App-local Frameworks / PlugIns / LoginItems dylib search paths
- Weak / optional dylib dependencies declared by a host Mach-O
- Proxy dylib carrying an LC_REEXPORT_DYLIB to a renamed copy of the original

Resolves to

- Malicious dylib loaded under an existing trusted host process
- Original library functionality preserved through the proxy re-export

Enumerate

```
otool -l <host_binary> | grep -E 'LC_LOAD_DYLIB|LC_LOAD_WEAK_DYLIB|LC_REEXPORT_DYLIB' -A 3
```

Copy

```
find <App>.app/Contents \( -name '*.dylib' -o -name '*.framework' \)
```

Copy

Inspect

```
otool -l <suspect_dylib> | grep -A 3 LC_REEXPORT_DYLIB
```

Copy

```
codesign -dv --verbose=4 <suspect_dylib>
```

Copy

```
codesign -d --entitlements :- <host_app> 2>/dev/null
```

Copy

Host app launch / dynamic load

Medium FP · Baseline: Helpful · Use: Execution hijack

Distinct from a simple LC_LOAD_DYLIB addition. Two related techniques: (1) **dylib hijacking** exploits a host that searches multiple directories for a dylib (or has a weak / optional dependency that does not exist), letting the attacker drop a malicious dylib of the same name in the first searched / writable location; (2) **dylib proxying** replaces a real dependency with a malicious dylib that carries an `LC_REEXPORT_DYLIB` command pointing to a renamed copy of the original, so functionality is preserved while the proxy executes its constructor inside the trusted host. Both inherit the host's TCC permissions (camera, mic, etc.). Detection pivots: presence of `LC_REEXPORT_DYLIB` in an app's bundled dylib, dylibs in writable per-app directories whose signer differs from the host, and hardened-runtime hosts carrying `com.apple.security.cs.disable-library-validation`.

Login Hooks

[S22]

Mixed | Deprecated since 10.11

User or system plist; verify on target OS

Mixed | Disk | High

Artifact / Path

- /Library/Preferences/com.apple.loginwindow.plist
- ~/Library/Preferences/com.apple.loginwindow.plist
- LoginHook
- LogoutHook

Resolves to

- LoginHook / LogoutHook script path
- Command or script executed by loginwindow policy

Enumerate

```
sudo defaults read /Library/Preferences/com.apple.loginwindow 2>/dev/null
```

Copy

```
find /Users -path '*/Library/Preferences/com.apple.loginwindow.plist' 2>/dev/null
```

Copy

```
defaults read ~/Library/Preferences/com.apple.loginwindow 2>/dev/null
```

Copy

Inspect

```
plutil -p /Library/Preferences/com.apple.loginwindow.plist
```

Copy

```
plutil -p ~/Library/Preferences/com.apple.loginwindow.plist
```

Copy

Login / logout

Low FP · Baseline: Optional · Use: Login hook

Deprecated since 10.11. Hooks can be installed either user-scoped (`~/Library/Preferences/com.apple.loginwindow.plist`, current user only) or system-wide (`/Library/Preferences/com.apple.loginwindow.plist`, requires root, fires for every user at login). The system-wide form is the more dangerous DFIR variant and is often missed when only the user plist is checked. Execution behavior and practical support are historical / target-OS dependent, so verify on the endpoint rather than assuming a hard upper bound from the badge alone.

Startup Items

[S23]

System

Legacy; mainly pre-10.4 compatibility

Pre-launchd / compatibility

Root | Disk | High

Artifact / Path

- /Library/StartupItems
- StartupParameters.plist

Resolves to

- StartupItem script or executable
- StartupParameters.plist-defined item actually launched during boot

Enumerate

```
find /Library/StartupItems -maxdepth 2 -type f
```

Copy

Boot

Low FP · Baseline: Optional · Use: Boot autostart

Legacy mechanism mainly relevant for OS X 10.3 and earlier compatibility. From 10.4 onward, low-level service startup largely moved to launchd, but the artifact still matters in older images, migrations, and niche DFIR cases.

				Inspect `plutil -p /Library/StartupItems/*/StartupParameters.plist` Copy		

RC Scripts
[S24]
System · <=10.3
Pre-launchd
	Root	Disk	High	Artifact / Path
				`rc.local`
				`rc.common`
				`/private/etc/rc*`
				Resolves to
				`Script body or chained command path`
				`Actual command invoked from rc.local / rc.common`
				Enumerate
				`find /private/etc -maxdepth 2 -name 'rc*'` Copy
				Inspect
				`grep -n '' /private/etc/rc*` Copy
					Boot / init path	Low FP · Baseline: Optional · Use: Boot script
						Mostly historical on current macOS: like Login Hooks and Startup Items, these mechanisms are deprecated boot paths whose presence on a modern host is itself anomalous and therefore high-signal. Keep for completeness and older-host triage; on modern fleets, treat any non-empty rc.local / rc.common as immediate priority rather than as a first-pass enumeration target.

05 Configuration Profiles & MDM

Configuration profiles (`.mobileconfig`) and MDM-pushed payloads are a deployment surface, not a launchd-style autostart, but they materially shape what runs and what is allowed to run on a Mac. Treat them as a first-class persistence and policy surface: a profile can install login items, allowlist a system extension or kext, grant TCC / PPPC exceptions, install root certificates, push managed preferences for any app, or enroll the device in MDM with remote-management privileges. On a compromised or social-engineered host, a single rogue profile can re-establish other persistence and silently relax defenses.

Mechanism	Required privilege	Source of Truth	Signal	Collection / Triage	Trigger	What to review

Configuration Profiles (.mobileconfig)
[S25]
Mixed · All
User or device scope; install-time consent
	Mixed	Both	Medium	Artifact / Path
				`/var/db/ConfigurationProfiles/Store/`
				`/var/db/ConfigurationProfiles/Settings/`
				`/Library/Managed Preferences/ (resulting managed prefs)`
				`/Library/Managed Preferences/<user>/ (per-user managed prefs)`
				`Stray .mobileconfig files in Downloads / Desktop / temp`
				Resolves to
				`Installed configuration profile, its identifier, and the issuing organization`
				`Effective managed-preference values that override per-app or per-system defaults`
				Enumerate
				`sudo profiles list` Copy
				`sudo profiles -P` Copy
				`sudo profiles show -type configuration` Copy
				`sudo profiles show -type enrollment` Copy
				`find /Library/Managed\ Preferences -type f 2>/dev/null` Copy
				`find ~/Downloads ~/Desktop /tmp /var/tmp -name '*.mobileconfig' 2>/dev/null` Copy
				Inspect
				`sudo profiles -P -o /tmp/profiles_export.plist` Copy
				`sudo ls -la /var/db/ConfigurationProfiles/Store/ /var/db/ConfigurationProfiles/Settings/` Copy
				`security cms -D -i <profile.mobileconfig>` Copy
				`plutil -p /Library/Managed\ Preferences/<domain>.plist` Copy
					Profile install / MDM push / managed pref load	Medium FP · Baseline: Critical · Use: Policy / deployment
						Two install paths exist: a user double-clicks a `.mobileconfig` and approves it in System Settings, or an MDM server pushes it silently to an enrolled device. In both cases the on-disk truth lives under `/var/db/ConfigurationProfiles/` while the enforced effects surface in `/Library/Managed Preferences/`. Use `sudo profiles list` as the first pivot: any profile whose source organization, identifier, or payload set does not match the documented enterprise baseline is high-priority. Profiles installed outside an MDM enrollment (i.e. user-approved local profiles on a managed device) are particularly suspicious. Decode raw `.mobileconfig` files with `security cms -D -i` to read their CMS-signed XML payloads. Treated as mixed scope because profiles can target the device, the current user, or both.

High-impact MDM payloads
[S25a]
Mixed · All; payload set evolves per release
PPPC, SystemExtensions, kext allowlists, certs, login items
	Mixed	Both	High	**Artifact / Path**
				`PayloadType com.apple.TCC.configuration-profile-policy (PPPC / privacy)`
				`PayloadType com.apple.system-extension-policy`
				`PayloadType com.apple.syspolicy.kernel-extension-policy`
				`PayloadType com.apple.servicemanagement (managed login items)`
				`PayloadType com.apple.security.pkcs1 / .pem / .root (installed certs)`
				`PayloadType com.apple.ManagedClient.preferences (custom managed prefs for any app)`
				
				Resolves to
				`TCC / PPPC exception that lets a chosen binary bypass user prompts for camera, mic, full disk access, AppleEvents, accessibility, etc.`
				`Allowlisted Team ID / Bundle ID for system extensions or kexts`
				`Trusted root certificate added to the system / user keychain`
				`Forced login item or background service registered via SMAppService`
				
				Enumerate
				`sudo profiles show -type configuration | grep -E 'PayloadType|PayloadIdentifier|PayloadOrganization'` Copy
				`sudo sqlite3 /Library/Application\ Support/com.apple.TCC/TCC.db 'SELECT service, client, auth_value, policy_id FROM access WHERE policy_id IS NOT NULL;'` Copy
				`security find-certificate -a -p /Library/Keychains/System.keychain` Copy
				`systemextensionsctl list` Copy
				
				Inspect
				`security cms -D -i <profile.mobileconfig> | plutil -p -` Copy
				`security cms -V -i <profile.mobileconfig>` Copy
				`codesign -dv --verbose=4 <binary_granted_TCC_via_profile>` Copy
					Profile install / payload activation	Medium FP · Baseline: Critical · Use: Policy / privilege grant
						These payload types are the ones that meaningfully change a host's security posture and that an attacker would actually want to abuse. PPPC payloads can grant a binary persistent TCC / privacy exceptions that the user would otherwise have to click through (the attacker's payload then runs with full disk access, accessibility control, or AppleEvents send rights without prompts). System-extension and kext policy payloads can pre-allowlist a third-party Team ID, removing the approval friction that normally protects extension installs. `com.apple.servicemanagement` can register login items / SMAppService entries from policy. Certificate payloads can install root CAs that defeat TLS pinning expectations. `com.apple.ManagedClient.preferences` is a wildcard: it can push managed-preference key/value pairs into any app's preference domain. Operational caveat: querying the system TCC.db requires the terminal (or your DFIR tool) to hold Full Disk Access, even with sudo, because the file is itself protected by TCC; entries with non-null `policy_id` are the ones installed by configuration profile rather than by user click. For each profile, decode and review the full payload list, not just the profile name; map every PPPC entry to the actual binary it empowers; correlate any new system-extension or kext allowlist with a real activation event in `systemextensionsctl list` or `kmutil showloaded`.

Quick IR Checklist

01. Baseline all **launchd** jobs and query live state with `launchctl print` and `print-disabled`.
02. Dump BTM / Login Items and map every helper to a real, signed bundle and expected install source.
03. Check pre-login persistence first: LaunchDaemons, System Extensions, KEXTs, and (on macOS ≤12 only) `emond`.
04. Diff shell startup files and resolve `ZDOTDIR` before concluding nothing changed.
05. Enumerate scheduled execution through `crontab`, `atq`, and `/etc/periodic`.
06. On noisy surfaces, triage by signer, Team ID, policy context, path quality, and install provenance rather than by path alone.
07. For every artifact, capture signer, Team ID, path, first-seen or write time, parent process, and user context.
08. Enumerate installed configuration profiles via `sudo profiles list` / `sudo profiles show -type configuration`; flag any profile not pushed by your MDM and review high-impact payloads (PPPC, system-extension and kext allowlists, certificate installs, managed login items).

Weak Signals / Priority Indicators

- Recent plist in `~/Library/LaunchAgents` or `/Library/LaunchDaemons` that points into `/tmp`, `/private/tmp`, `/Users/Shared`, a hidden directory, or an Apple-looking filename in a user-writable path.
- `RunAtLoad` + `KeepAlive` on an unsigned or newly written target, especially when paired with `WatchPaths` or `QueueDirectories` on `~/Downloads`, Desktop, temp, or shared folders.
- BTM entry or embedded helper whose path is stale, user-writable, or whose signer / Team ID does not match the enclosing app.
- New SSH key on a sensitive account with no corresponding admin workflow, ticket, or rotation event.
- `ZDOTDIR` set to an unusual path or shell rc files that silently source another script from a hidden or transient location.
- Unexpected third-party entry in `systemextensionsctl list` or a kext that appears on a host where one is not operationally expected.
- New Chromium extension ID, policy-forced install, or managed profile entry; Safari appex whose signer or owner app does not fit the baseline.
- `.pth` file containing executable `import` lines, or a new `sitecustomize.py` / `usercustomize.py` under an unexpected site-packages directory.
- Trusted binary that suddenly loads a dylib from a user-writable path, hidden directory, or a location inconsistent with the app's normal install layout.

Corroborating Evidence

01. **Signature / Team ID**: confirm the persisted component, its owner app, and any helper all chain to the same signer where that is expected.
02. **Quarantine / provenance**: inspect `xattr -l` for `com.apple.quarantine` and review how the artifact first arrived on the host.
03. **Timestamps**: compare birthtime, mtime, and first-seen telemetry against the suspected execution or install window.
04. **Unified Logs**: pivot to system and install events around the write or enablement time, especially for launchd, extensions, login items, and package installs.
05. **Process ancestry**: identify what created or modified the artifact, then compare that parent process to expected install or admin workflow.
06. **Policy context**: separate enterprise-managed state from suspicious state by checking MDM, managed preferences, forced browser installs, and approved extension workflows.
07. **Receipts / app ownership**: correlate pkg receipts, app bundle metadata, and on-disk owner app presence before calling an embedded helper or extension malicious.

Logs / Telemetry to Check

01. Unified Logs for launch activity, especially suspicious launchd labels, repeated failures, and unexpected restarts.
02. backgroundtaskmanagementagent and related BTM records when Login Items or helpers look inconsistent with the installed app set.
03. sysextd, systemextensionsctl output, and approval-related records when a system extension or driver stack is in scope.
04. install.log and package provenance when preinstall / postinstall or helper deployment appears to have introduced persistence.

05 sshd logs plus auth context for unexpe

06 Signer, Team ID, notarization, quarantir

---

macOS Persistence Cheatsheet by Jaybird1291 cted authorized_keys changes, custom Auth

ne xattrs, mtime / birthtime, and first-seen tel orizedKeysFile usage, or unusual remote

emetry to validate whether the artifact m access timing.

atches its claimed parent app or install flow.

DFIR

Triage

Forensic

Detection

Engineering