Sleep Securely

Sacha

March 26, 2025

Contents

1 Synopsis

When a macOS system goes to sleep, it preserves the environment *exactly* as it was when the system sleep event was initiated. This is excellent for most people, as it is predictable and comfortable, convenient. Power users have other needs beyond this, often wanting alternative conveniences for their *less usual* habits, such as muting the system (we don't know what space we'll be in next), proactively turning off bluetooth and WiFi (reducing our security vulnerability), as well as other conveniences.

This journal is a *literate* journal describing the process in detail, with the intention that it is *tangled* into the target file.

2 Basic requirements

To simplify script handling and installation, a `justfile` will be created with basice *recipes*.

```
[private]
@help:
 just --list

# Add shell script executable status
[private]
chmod:
 @echo "==> Setting executable flag"
 chmod +x ./scripts/sleep_securely.sh
 @echo "--> Executable flag successfully set\n"

# Install script
install: chmod
 @echo "==> Installing script"
 cp ./scripts/sleep_securely.sh ~/bin/sleep_securely.sh
 @echo "--> Script successfully installed\n"
```

3 Actions to take upon sleep

Our target script file is a shell script, so we need to specify the script header first.

```
#!/bin/bash
```

Now, we'll incrementally add to the functionality of that script, through addressing a range of security and privacy-enhancing categories, one at a time.

3.1 Immediately implement security measures

As macOS, or rather 'sleepwatcher', limits how long a sleep command (or our entire script) may be allowed to run

> sleepcommand must not take longer than 15 seconds because after this timeout the sleep mode is forced by the system

then it's important to ensure that critical tasks are carried out immediately, so they wouldn't be *cut off* by the daemon.

3.1.1 Quit KeePassXC

Arguably the most valuable and confidential resource is the password library, hosted on the excellent KeePassXC application. As good as it is, this application still has to store the key somewhere in memory in exchange for giving us quick and convenient access to the password database. Let's ensure that this is the very first—and most important—action taken of all the security steps. Even if all aspects of this script fail, this one is the most valuable and impactful: kill KeePassXC, ejecting it from memory.

```
# Quit KeePassXC upon sleep
/usr/bin/killall KeePassXC
```

3.1.2 Quit Parallels Desktop

Prior to more destructive changes, let us also initiate a controlled quitting of Parallels Desktop. This application is used to run any number of servers for isolation *and* security. Its networking is able to interfere with the establishment of wifi connections, particularly some behind *captive* wifi setups, found in many coffee shops and locations advertising *free* wireless.

```
# Tell Parallels Desktop application to quit gently, if it is already open
if pgrep -q prl_client; then
    osascript -e 'tell application "Parallels Desktop" to quit'
fi
```

3.1.3 Restrict access to Keychain

For years, macOS has implemented its own internal *password vault*, Keychain. When a user logs into macOS, the Keychain adds convenience by ***keeping*** the password in memory, so that applications needing access to services can transparently obtain the password(s) they need, which is arguably a security vulnerability.

Login Keychain is the primary password vault for macOS, storing passwords for:

- Websites (via Safari)
- Email accounts
- Wi.Fi networks
- Secure notes
- App-specific passwords
- SSH keys and certificates

The security enhancement that will be had by locking the keychain—at a cost to some added inconveience—is the prevention unauthorised access to your stored credentials, asking for your password to unlock the keychain again when needed. This create a protection layer if someone accesses the computer while sleeping.

When locked, applications can't silently access stored credentials; the next time an application needs a password, macOS will prompt for your login password. Applications can't automatically authenticate to services without your manual intervention. If your computer is accessed while sleeping—e.g. through an ***evil maid*** attack—the attacker can't use stored passwords, preventing unauthorised access to accounts, even if physical access is gained. Applications can no longer continue to use credentials in the background.

The following command provides a crucial layer of protection by requiring re-authentication after sleep. It is one of the most effective security measures in the script, creating a "clean break" in credential access, preventing automated or background access to accounts, forcing manual re-authentication for sensitive operations

This is particularly valuable if your computer might be accessed by others while in sleep mode, as it prevents the use of stored credentials without your knowledge or permission. When you wake your computer, you'll need to enter your password the first time an application tries to access the keychain, which is a small inconvenience for a significant security benefit.

```
# Temporary keychain access restrictions
security lock-keychain ~/Library/Keychains/login.keychain
```

3.2 Disable network services

3.2.1 Disable wireless networking (Wi-Fi)

As I bring work to a close in a location I felt comfortable in, and ***trusted***, I cannot know with certainty where I will be waking the system up again. As the wireless radio comes back to life, it starts out by scanning for known access points, giving away unintended information. In the interests of reducing exposure in unknown public settings, let's turn off the wireless network and force a positive decision to bring it up manually when desired.

```
# Disable Wi-Fi
networksetup -setairportpower en0 off
```

3.2.2 Disable bluetooth connectivity

In the same vein, as well as for the pure convenience of not having the laptop try to *snatch* bluetooth connections to headphones and mice during its sleep, I want to bring down the bluetooth connection.

```
# Disable Bluetooth
/opt/homebrew/bin/blueutil -p 0
```

3.2.3 Disable AirDrop connectivity

Another service that can expose unnecessary information to those in its proximity is AirDrop. AirDrop is a very useful service for easy sharing of files and information with trusted people, but most people *forget* to turn it off after use, and almost everybody forgets to reduce its permission from allowing connection attempts from 'Everyone', back down to the reduced 'Contacts Only'.

In the following block, we will first reduce the permission to accept attempts from contacts, followed by disabling the AirDrop service.

```
# Lower connection permissions to 'Contacts Only'
defaults write com.apple.sharingd DiscoverableMode -string "Contacts Only"

# Disable AirDrop discoverability
defaults write com.apple.sharingd DiscoverableMode -string "Off"

# Restart the sharingd service for the above changes to take effect
killall sharingd
```

As a side-note, for potential future improvement, AirDrop relies on a dedicated interface, 'awdlo' or *Apple Wireless Direct Link* (*AWDL*), which is more secure than merely disabling the service. This can be done at sleep by running the following command: `sudo ifconfig awdl0 down`, followed by re-enabling it on wake: `sudo ifconfig awdl0 up`. Unfortunately, this *must* be run with administrative privileges, opening the script up to serious vulnerabilities, which I *do not* want to do here!

3.2.4 Unmount all externally mounted volumes

Unmount all external currently mounted volumes. This is needed to unmount any SSH file system mounted volumes relying on Tailscale (see below), which is why this step *must* run before Tailscale is taken offline. From a security perspective, it is also desirable to unmount any external volumes *especially* the encrypted ones, forcing me to manually remount them, reentering any encryption keys when needed once again.

```
# Unmount all external volumes
diskutil list external | grep -E '^\/' | while read -r volume; do
    diskutil unmount "$volume"
done
```

One known issue with privacy and use of secure VPNs, is that Tailscale, while incredibly useful, can result in alternative, unsecured, network routing. Before going to sleep, take Tailscale offline; it's easy to take it back online when needed again.

3.2.5 Disable the Tailscale VPN

```
# Disable Tailscale
if [ -x "/Applications/Tailscale.app/Contents/MacOS/Tailscale" ]; then
    "/Applications/Tailscale.app/Contents/MacOS/Tailscale" down
fi
```

3.2.6 Mute system volume

When closing the lid, literally and figuratively, on a project, there's no way of knowing where we'll be when we reawaken the computer. Just imagine the embarrassment of opening the lid and a movie or song resuming at full blast in a library, or worse, a quiet business meeting! Let's always mute the volume before going to sleep, it's safer.

```
# Mute system volume to prevent unpleasant surprises!
osascript -e "set volume with output muted"
```

3.3 Close all sensitive applications

Given that it's impossible to know who will next *open*, or wake up, your computer, it's safest to proactively *kill* all communications applications with confidential information.

```
# Close all email applications (MUAs)
pkill -x "Proton Mail"
pkill -x "Thunderbird"
pkill -x "thunderbird"

# Close sensitive communication applications
pkill -x "Discord"
pkill -x "FaceTime"
pkill -x "Messages"
pkill -x "Microsoft Teams"
pkill -x "Signal"
pkill -x "Skype"
pkill -x "Slack"
pkill -x "Telegram"
pkill -x "Trello"
pkill -x "Viber"
pkill -x "WhatsApp"

pkill -x "Zoom"
pkill -x "Zoom.us"
pkill -x "zoom"
pkill -x "zoom.us"

pkill -x "Authy Desktop"
```

3.4 Clearing the clipboard

The clipboard is a wonderful thing, it's there to help us move blocks of information around, but it is liable to store many things we don't want to share: passwords copied from a password manager, confidential text from an email or private message, sensitive information personally identifying you. From a convenience standpoint, we rely on this functionality to paste *recently* copied text; once the system has been suspended and resumed, there is no longer a reasonable context around what was recent.

Objectively the best thing to do is to clear the clipboard to prevent any data leakage.

```
# Clear clipboard contents
/usr/bin/pbcopy < /dev/null
```

3.5 SSH security

SSH is an invaluable system administration tool for logging on to remote machines securely. As this is an important function in maintenance, development and other administration tasks, an agent is provided for convenience, `ssh-agent`, to help track and load users' identity keys.

Once keys are loaded, the agent provides user convenience by keeping private keys loaded, so that the passphrase doesn't need to be entered repeatedly. Removing keys from the agent is a good security practice as anyone walking up to the machine after waking from sleep cannot simply log in to a remote machine; once the key is deleted from the agent, `ssh-agent` won't be able to use it. This means it can't expose any secret information or establish unauthorized SSH sessions on other devices. It's straightforward to delete the key using `ssh-add -D`.

```
# Clear SSH agent identities
ssh-add -D
```

Rather than stop there, why even keep `ssh-agent` running? Let's kill it, ensuring that not only keys but also any other information it may be holding in memory is released and unusable.

```
# Securely wipe SSH agent
killall ssh-agent
```

3.6 Privacy

3.6.1 Browsing privacy

macOS keeps a record of every file downloaded from the internet. These records include download dates, sources, and applications used. Clearing the responsible SQLite database removes this download history, evidence of downloaded files that might be sensitive, making it harder for malicious actors to see what has been downloaded, reducing leakage of internet activity.

The way it works is that when you download files, macOS adds a quarantine attribute, triggering "This file was downloaded from the internet" warnings. The database keeps metadata about these downloads

If you download a document from a secure source before sleep, this command ensures that when your system wakes up, there's no record of that download in the system's databases.

This action isn't destructive, only removing metadata about downloaded files, not the files themselves. The system will continue to function normally, and future downloads will simply start creating new entries in the database. This is an excellent privacy-enhancing measure that leaves no obvious trace of your download activity between sleep sessions, while having zero impact on system functionality.

Below is the command that will clear this database on close:

```
# Download trace elimination
sqlite3 ~/Library/Preferences/com.apple.LaunchServices.QuarantineEventsV* "DELETE FROM LSQuarantineE
```

1. How does it work?

   This is the path to the macOS quarantine database:

   ```
   ~/Library/Preferences/com.apple.LaunchServices.QuarantineEventsV*
   ```

   The database is located in the user's `Library/Preferences` folder, and the `*` is a wildcard matching any version number (e.g., QuarantineEventsV2, QuarantineEventsV3).

   The following SQL command removes all records from the `LSQuarantineEvent` table:

   ```
   DELETE FROM LSQuarantineEvent;
   ```

3.6.2 Finder metadata and usage trails

When navigating folders, macOS will creates `.DS_Store` files as needed based on the current view. `.DS_Store` files are hidden files created by Finder storing custom attributes of a folder, such as: icon positions, view settings (list, grid, etc.), folder background images, and sidebar width. These files contain metadata about how you organize your files, can reveal folder structure information and may expose browsing habits and organizational patterns.

Deleting these files before sleep removes metadata that could be leveraged by forensic analysis, is particularly important for external drives and folders shared with others, preventing information leakage about your file system organization. Deleting them removes this potential source of information leakage.

You might notice folder views reset to default appearance, but no actual data is lost.

It's an elegant way to reduce your digital footprint without affecting system functionality, essentially resetting the "fingerprints" left by your Finder activity.

```
# Traverse the user's home directory, searching for, and deleting all '.DS_Store' files
find ~/ -type f -name ".DS_Store" -delete
```

3.7 Sanitise application memory

The *Application memory sanitisation* section is a sophisticated security measure designed to clean up potentially sensitive data from your computer's memory, held by the kind of application that can possess sensitive data, such as web browsers. We aim to gracefully close specific applications and clear their data from memory, preventing potential memory scraping attacks or information leakage.

When applications of this type run, they store sensitive data in memory. This can include login credentials, session cookies, browsing history, email contents, form data, cached documents, and more.

This attack mitigation action prevents memory scraping attacks, reduces the attack surface for cold boot attacks, limits the effectiveness of memory forensics, and generally releases memory, reducing memory consumption ballooning that's so prevalent with web browsers.

Why these applications? Browsers (Safari, Chrome, Firefox) store web credentials and session data, mail applications contain email content and potentially sensitive attachments, typically caching a significant amount of user data in memory.

3.7.1 Technical security context

System memory retains data even after applications close. By gracefully closing apps, the system allows them to clear sensitive caches while closing themselves, and removing themselves from memory. The memory will eventually be overwritten by other processes in the future.

This is a non-destructive operation. By gracefully closing applications, it allows them to save their state properly. When you wake your computer, you'll need to reopen these applications, but they'll typically restore your previous sessions. AppleScript is used instead of `kill` commands, relying on AppleScript's "quit" command rather than forcefully terminating processes because it allows applications to clean up their own memory, it prevents potential data corruption, while properly saving application state for later reopening.

3.7.2 Advanced considerations

In a more comprehensive security setup, this could be followed by a memory compaction operation (like the `sudo purge` command) to further reduce the persistence of sensitive data in RAM. This technique is particularly valuable when combined with disk encryption, as it helps ensure that even if someone has physical access to your sleeping computer, they can't easily extract sensitive information from memory.

```
# Application memory sanitization
for app in "Safari" "Iridium" "Chrome" "Chromium" "firefox" "Firefox Nightly" "Mail" "zotero" "Claud
    if pgrep -q $app; then
        osascript -e "tell application \"$app\" to quit" 2>/dev/null
```

```
    fi
done
```

Firefox Nightly has the bad habit of also appearing as "firefox", which means that it doesn't quit as the main quit call goes to Firefox *proper*. Let's take care of this by killing it in a more targeted way.

```
# Kill Firefox Nightly
pgrep "firefox" | xargs --no-run-if-empty ps | grep -e 'Firefox Nightly' | cut -w -f1 | xargs kill -
```

Arc browser misbehaves following attempts to quit it in the above way, so it should be quit more abruptly, using the non-ignorable 'kill' command: `pkill -9`, or "KILL (non-catchable, non-ignorable kill)".

```
# Send a kill command to Arc browser
pkill -9 Arc
```

Next, let's kill Spotify if it's running. While admittedly this has nothing to do with security or confidentiality, it's a practical measure, as Spotify uses the Electron framework—essentially a headless browser invocation—which uses far to many system resources which I would like released.

```
# Kill Spotify desktop app if running, releasing system resources
pkill -9 Spotify
```