drduh / **macOS-Security-and-Privacy-Guide**

A practical guide to securing macOS.

#apple #macos #security #privacy #osx #disk-encryption #macos-setup #macos-security #macbook-security #dnscrypt-proxy #macbook-configuration

431 commits **1** branch **0** releases **53** contributors MIT

Branch: **master** New pull request Find file Clone or download

drduh Update flash instructions, fix #247 Latest commit 9991683 Jan 22, 2018

14F27_launchd.csv	Describe a few more services, and suggest 'Yosemite-Stop-Launch'. Fix #…	Oct 31, 2015
15B42_launchd.csv	Add 10.11.1 services csv.	Oct 25, 2015
16A323_launchd.csv	Remove broken line	Oct 9, 2016
CNAME	Create CNAME	May 9, 2017
InstallESD_Hashes.csv	Add 10.13.2 (17C88) hashes	Jan 8, 2018
LICENSE	Initial commit	Aug 31, 2015
README-cn.md	Track upstream updates	May 15, 2017
README.md	Update flash instructions, fix #247	Jan 22, 2018
_config.yml	Set theme jekyll-theme-minimal	May 9, 2017
comments.csv	Re-usable comments	Nov 13, 2015
read_launch_plists.py	Fix for broken jetsamproperties file.	Oct 9, 2016

README.md

This guide is a collection of thoughts on and techniques for securing a modern Apple Mac computer ("MacBook") using macOS (formerly known as *OS X*) version 10.12 "Sierra", as well as steps to generally improving privacy.

This guide is targeted to “power users” who wish to adopt enterprise-standard security, but is also suitable for novice users with an interest in improving their privacy and security on a Mac.

A system is only as secure as its administrator is capable of making it. There is no one single technology, software, nor technique to guarantee perfect computer security; a modern operating system and computer is very complex, and requires numerous incremental changes to meaningfully improve one's security and privacy posture.

This guide is provided on an 'as is' basis without any warranties of any kind. Only **you** are responsible if you break a Mac by following any of the steps herein.

If you wish to make a correction or improvement, please send a pull request or open an issue.

This guide is also available in 简体中文.

- Basics
- Firmware
- Preparing and Installing macOS

Basics

The standard best security practices apply:

- Create a threat model
  - What are you trying to protect and from whom? Is your adversary a three letter agency (if so, you may want to consider using OpenBSD instead), a nosy eavesdropper on the network, or determined apt orchestrating a campaign against you?
  - Study and recognize threats and how to reduce attack surface against them.
- Keep the system up to date
  - Patch, patch, patch your system and software.
  - macOS system updates can be completed using the App Store application, or the `softwareupdate` command-line utility - neither requires registering an Apple account.
  - Subscribe to announcement mailing lists (e.g., Apple security-announce) for programs you use often.
- Encrypt sensitive data
  - In addition to full disk encryption, create one or many encrypted containers to store passwords, keys, personal documents, and other data at rest.
  - This will mitigate damage in case of compromise and data exfiltration.
- Frequent backups
  - Create regular backups of your data and be ready to reimage in case of compromise.
  - Always encrypt before copying backups to external media or the "cloud".
  - Verify backups work by testing them regularly, for example by accessing certain files or performing a hash based comparison.
- Click carefully
  - Ultimately, the security of a system can be reduced to its administrator.
  - Care should be taken when installing new software. Always prefer free and open source software (which macOS is not).

Firmware

Setting a firmware password prevents your Mac from starting up from any device other than your startup disk. It may also be set to be required on each boot.

This feature can be helpful if your laptop is lost or stolen, protects against Direct Memory Access (DMA) attacks which can read your FileVault passwords and inject kernel modules such as pcileech, as the only way to reset the firmware password is through an Apple Store, or by using an SPI programmer, such as Bus Pirate or other flash IC programmer.

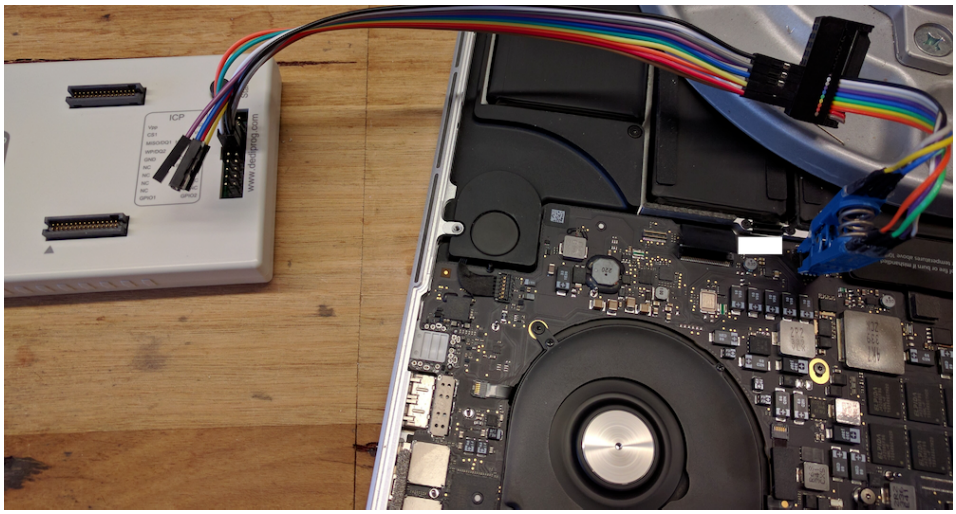
1. Start up pressing `Command` `R` keys to boot to Recovery Mode mode.
2. When the Recovery window appears, choose **Firmware Password Utility** from the Utilities menu.
3. In the Firmware Utility window that appears, select **Turn On Firmware Password**.
4. Enter a new password, then enter the same password in the **Verify** field.
5. Select **Set Password**.
6. Select **Quit Firmware Utility** to close the Firmware Password Utility.
7. Select the Apple menu and choose Restart or Shutdown.

The firmware password will activate at next boot. To validate the password, hold `Alt` during boot - you should be prompted to enter the password.

The firmware password can also be managed with the `firmwarepasswd` utility while booted into the OS. For example, to prompt for the firmware password when attempting to boot from a different volume:

```
$ sudo firmwarepasswd -setpasswd -setmode command
```

Enter a password and reboot.



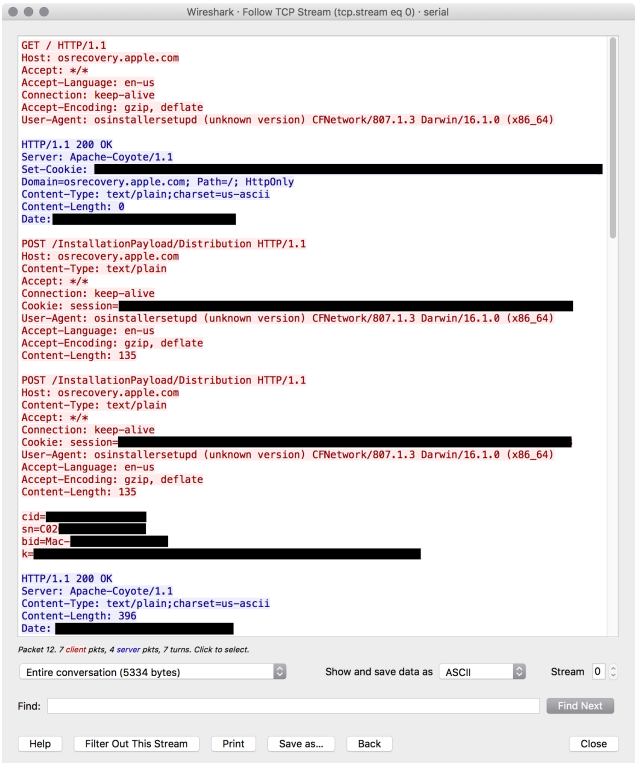
Using a Dediprog SF600 to dump and flash a 2013 MacBook SPI Flash chip to remove a firmware password, sans Apple

See HT204455, LongSoft/UEFITool and chipsec/chipsec for more information.

Preparing and Installing macOS

There are several ways to install a fresh copy of macOS.

The simplest way is to boot into Recovery Mode by holding `Command` `R` keys at boot. A system image can be downloaded and applied directly from Apple. However, this way exposes the serial number and other identifying information over the network in plaintext.



Packet capture of an unencrypted HTTP conversation during macOS recovery

Another way is to download **macOS Sierra** from the App Store or some other place and create a custom, installable system image.

The macOS Sierra installer application is code signed, which should be verified to make sure you received a legitimate copy, using the `spctl -a -v` or `pkgutil --check-signature` commands:

```
$ pkgutil --check-signature /Applications/Install\ macOS\ Sierra.app
Package "Install macOS Sierra.app":
   Status: signed by a certificate trusted by Mac OS X
   Certificate Chain:
    1. Apple Mac OS Application Signing
       SHA1 fingerprint: B9 3B DA AA F1 A8 84 6B 34 BA 32 33 26 35 CB 2B 84 85 3D A8
       ------------------------------------------------------------------------------
    2. Apple Worldwide Developer Relations Certification Authority
       SHA1 fingerprint: FF 67 97 79 3A 3C D7 98 DC 5B 2A BE F5 6F 73 ED C9 F8 3A 64
       ------------------------------------------------------------------------------
    3. Apple Root CA
       SHA1 fingerprint: 61 1E 5B 66 2C 59 3A 08 FF 58 D1 4A E2 24 52 D1 98 DF 6C 60
```

You may also use the `codesign` command to examine an application's code signature:

```
$ codesign -dvv /Applications/Install\ macOS\ Sierra.app
Executable=/Applications/Install macOS Sierra.app/Contents/MacOS/InstallAssistant
Identifier=com.apple.InstallAssistant.Sierra
Format=app bundle with Mach-O thin (x86_64)
CodeDirectory v=20200 size=297 flags=0x200(kill) hashes=5+5 location=embedded
Signature size=4167
Authority=Apple Mac OS Application Signing
Authority=Apple Worldwide Developer Relations Certification Authority
Authority=Apple Root CA
Info.plist entries=30
TeamIdentifier=K36BKF7T3D
Sealed Resources version=2 rules=7 files=137
Internal requirements count=1 size=124
```

macOS installers can be made with the `createinstallmedia` utility included in `Install macOS Sierra.app/Contents/Resources/` . See Create a bootable installer for macOS, or run the utility without arguments to see how it works.

Note Apple's installer does not appear to work across OS versions. If you want to build a 10.12 image, for example, the following steps must be run on a 10.12 machine!

To create a **bootable USB installer**, mount a USB drive, and erase and partition it, then use the `createinstallmedia` utility:

```
$ diskutil list
[Find disk matching correct size, usually "disk2"]

$ diskutil unmountDisk /dev/disk2

$ diskutil partitionDisk /dev/disk2 1 JHFS+ Installer 100%

$ cd /Applications/Install\ macOS\ Sierra.app

$ sudo ./Contents/Resources/createinstallmedia --volume /Volumes/Installer --applicationpath /Applications
/Install\ macOS\ Sierra.app --nointeraction
Erasing Disk: 0%... 10%... 20%... 30%... 100%...
Copying installer files to disk...
Copy complete.
Making disk bootable...
Copying boot files...
Copy complete.
Done.
```

To create a **custom installable image** which can be restored to a Mac, you will need to find the file `InstallESD.dmg` , which is also inside `Install macOS Sierra.app` .

With Finder, right click on the app, select **Show Package Contents** and navigate to **Contents** > **SharedSupport** to find the file `InstallESD.dmg` .

You can verify the following cryptographic hashes to ensure you have the same copy with `openssl sha1 InstallESD.dmg` or `shasum -a 1 InstallESD.dmg` or `shasum -a 256 InstallESD.dmg` (in Finder, you can drag the file into a Terminal window to provide the full path).

To determine which macOS versions and builds originally shipped with or are available for your Mac, see HT204319.

See InstallESD_Hashes.csv in this repository for a list of current and previous file hashes. You can also Google the cryptographic hashes to ensure the file is genuine and has not been tampered with.

To create the image, use MagerValp/AutoDMG, or to create it manually, mount and install the operating system to a temporary image:

```
$ hdiutil attach -mountpoint /tmp/install_esd ./InstallESD.dmg

$ hdiutil create -size 32g -type SPARSE -fs HFS+J -volname "macOS" -uid 0 -gid 80 -mode 1775
/tmp/output.sparseimage

$ hdiutil attach -mountpoint /tmp/os -owners on /tmp/output.sparseimage

$ sudo installer -pkg /tmp/install_esd/Packages/OSInstall.mpkg -tgt /tmp/os -verbose
```

This part will take a while, so be patient. You can `tail -F /var/log/install.log` in another Terminal window to check progress.

(Optional) Install additional software, for example Wireshark:

```
$ hdiutil attach Wireshark\ 2.2.0\ Intel\ 64.dmg

$ sudo installer -pkg /Volumes/Wireshark/Wireshark\ 2.2.0\ Intel\ 64.pkg -tgt /tmp/os

$ hdiutil unmount /Volumes/Wireshark
```

See MagerValp/AutoDMG/wiki/Packages-Suitable-for-Deployment for caveats and chilcote/outset to instead processes packages and scripts at first boot.

When you're done, detach, convert and verify the image:

```
$ hdiutil detach /tmp/os

$ hdiutil detach /tmp/install_esd

$ hdiutil convert -format UDZO /tmp/output.sparseimage -o ~/sierra.dmg

$ asr imagescan --source ~/sierra.dmg
```

Now `sierra.dmg` is ready to be applied to one or many Macs. One could futher customize the image to include premade users, applications, preferences, etc.

This image can be installed using another Mac in Target Disk Mode or from a bootable USB installer.

To use **Target Disk Mode**, boot up the Mac you wish to image while holding the `T` key and connect it to another Mac using a Firewire, Thunderbolt or USB-C cable.

If you don't have another Mac, boot to a USB installer, with `sierra.dmg` and other required files copied to it, by holding the *Option* key at boot.

Run `diskutil list` to identify the connected Mac's disk, usually `/dev/disk2`

(Optional) Securely erase the disk with a single pass (if previously FileVault-encrypted, the disk must first be unlocked and mounted as `/dev/disk3s2`):

```
$ sudo diskutil secureErase freespace 1 /dev/disk3s2
```

Partition the disk to Journaled HFS+:

```
$ sudo diskutil unmountDisk /dev/disk2

$ sudo diskutil partitionDisk /dev/disk2 1 JHFS+ macOS 100%
```

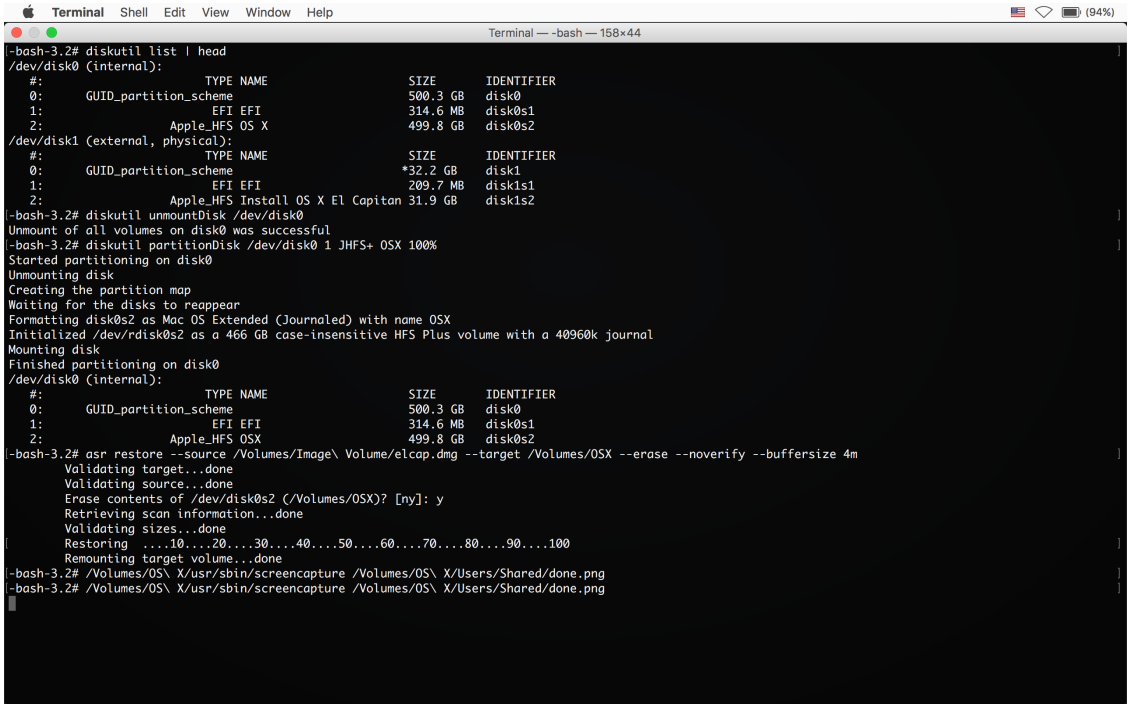
Restore the image to the new volume:

```
$ sudo asr restore --source ~/sierra.dmg --target /Volumes/macOS --erase --buffersize 4m
```

You can also use the **Disk Utility** application to erase the connected Mac's disk, then restore `sierra.dmg` to the newly created partition.

If you've followed these steps correctly, the target Mac should now have a new install of macOS Sierra.

If you want to transfer any files, copy them to a shared folder like `/Users/Shared` on the mounted disk image, e.g. `cp Xcode_8.0.dmg /Volumes/macOS/Users/Shared`



Finished restore install from USB recovery boot

We're not done yet! Unless you have built the image with AutoDMG, or installed macOS to a second partition on your Mac, you will need to create a recovery partition (in order to use full disk encryption). You can do so using MagerValp/Create-Recovery-Partition-Installer or using the following manual steps:

Download the file RecoveryHDUpdate.dmg.

```
RecoveryHDUpdate.dmg
SHA-256: f6a4f8ac25eaa6163aa33ac46d40f223f40e58ec0b6b9bf6ad96bdbfc771e12c
SHA-1:   1ac3b7059ae0fcb2877d22375121d4e6920ae5ba
```

Attach and expand the installer, then run it:

```
$ hdiutil attach RecoveryHDUpdate.dmg

$ pkgutil --expand /Volumes/Mac\ OS\ X\ Lion\ Recovery\ HD\ Update/RecoveryHDUpdate.pkg /tmp/recovery

$ hdiutil attach /tmp/recovery/RecoveryHDUpdate.pkg/RecoveryHDMeta.dmg

$ /tmp/recovery/RecoveryHDUpdate.pkg/Scripts/Tools/dmtest ensureRecoveryPartition /Volumes/macOS/ /Volumes
/Recovery\ HD\ Update/BaseSystem.dmg 0 0 /Volumes/Recovery\ HD\ Update/BaseSystem.chunklist
```

Replace `/Volumes/macOS` with the path to the target disk mode-booted Mac as necessary.

This step will take several minutes. Run `diskutil list` again to make sure **Recovery HD** now exists on `/dev/disk2` or equivalent identifier.

Once you're done, eject the disk with `hdiutil unmount /Volumes/macOS` and power down the target disk mode-booted Mac.

Virtualization

To install macOS as a virtual machine (vm) using VMware Fusion, follow the instructions above to create an image. You will **not** need to download and create a recovery partition manually.

```
VMware-Fusion-10.1.0-7370838.dmg
SHA-256: 5e968c5f88eb929740115374e0162779cbccd0383bc70e7bc52a0a680bf8fe2b
SHA-1:   ef694e2bba7205253d5fde6e68e8ba78fad82952
```

For the Installation Method, select *Install macOS from the recovery partition*. Customize any memory or CPU requirements and complete setup. The guest vm should boot into Recovery Mode by default.

Note If the virtual machine does not boot due to a kernel panic, adjust the memory and process resource settings.

In Recovery Mode, select a language, then select Utilities > Terminal from the menubar.

In the guest vm, type `ifconfig | grep inet` - you should see a private address like `172.16.34.129`

On the host Mac, type `ifconfig | grep inet` - you should see a private gateway address like `172.16.34.1` . From the host Mac, you should be able to `ping 172.16.34.129` or the equivalent guest vm address.

From the host Mac, serve the installable image to the guest vm by editing `/etc/apache2/httpd.conf` and adding the following line to the top (using the gateway address assigned to the host Mac and port 80):

```
Listen 172.16.34.1:80
```

On the host Mac, link the image to the default Apache Web server directory:

```
$ sudo ln ~/sierra.dmg /Library/WebServer/Documents
```

From the host Mac, start Apache in the foreground:

```
$ sudo httpd -X
```

From the guest VM, install the disk image to the volume over the local network using `asr` :

```
-bash-3.2# asr restore --source http://172.16.34.1/sierra.dmg --target /Volumes/Macintosh\ HD/ --erase
--buffersize 4m
        Validating target...done
        Validating source...done
        Erase contents of /dev/disk0s2 (/Volumes/Macintosh HD)? [ny]: y
        Retrieving scan information...done
        Validating sizes...done
        Restoring  ....10....20....30....40....50....60....70....80....90....100
        Verifying  ....10....20....30....40....50....60....70....80....90....100
        Remounting target volume...done
```

When it's finished, stop the Apache Web server on the host Mac by pressing `Control` `C` at the `sudo httpd -X` window and remove the image copy with `sudo rm /Library/WebServer/Documents/sierra.dmg`

In the guest vm, select *Startup Disk* from the menubar top-left, select the hard drive and restart. You may wish to disable the Network Adapter in VMware to configure the guest vm initially.

Take and Restore from saved guest vm snapshots before and after attempting risky browsing, for example, or use a guest vm to install and operate questionable software.

First boot

Note Before setting up macOS, consider disconnecting networking and configuring a firewall(s) first. However, late 2016 MacBooks with Touch Bar hardware require online OS activation.

On first boot, hold `Command` `Option` `P` `R` keys to clear NVRAM.

When macOS first starts, you'll be greeted by **Setup Assistant**.

When creating your account, use a strong password without a hint.

If you enter your real name at the account setup process, be aware that your computer's name and local hostname will comprise that name (e.g., *John Appleseed's MacBook*) and thus will appear on local networks and in various preference files. You can change them both in **System Preferences > Sharing** or with the following commands:

```
$ sudo scutil --set ComputerName your_computer_name
$ sudo scutil --set LocalHostName your_hostname
```

Admin and standard user accounts

The first user account is always an admin account. Admin accounts are members of the admin group and have access to `sudo`, which allows them to usurp other accounts, in particular root, and gives them effective control over the system. Any program that the admin executes can potentially obtain the same access, making this a security risk. Utilities like `sudo` have weaknesses that can be exploited by concurrently running programs and many panes in System Preferences are unlocked by default (pdf) (p. 61–62) for admin accounts. It is considered a best practice by Apple and others (pdf) (p. 41–42) to use a separate standard account for day-to-day work and use the admin account for installations and system configuration.

It is not strictly required to ever log into the admin account via the macOS login screen. The system will prompt for authentication when required and Terminal can do the rest. To that end, Apple provides some recommendations for hiding the admin account and its home directory. This can be an elegant solution to avoid having a visible 'ghost' account. The admin account can also be removed from FileVault.

Caveats

1. Only administrators can install applications in `/Applications` (local directory). Finder and Installer will prompt a standard user with an authentication dialog. Many applications can be installed in `~/Applications` instead (the directory can be created manually). As a rule of thumb: applications that do not require admin access – or do not complain about not being installed in `/Applications` – should be installed in the user directory, the rest in the local directory. Mac App Store applications are still installed in `/Applications` and require no additional authentication.

2. `sudo` is not available in shells of the standard user, which requires using `su` or `login` to enter a shell of the admin account. This can make some maneuvers trickier and requires some basic experience with command-line interfaces.

3. System Preferences and several system utilities (e.g. Wi-Fi Diagnostics) will require root privileges for full functionality. Many panels in System Preferences are locked and need to be unlocked separately by clicking on the lock icon. Some applications will simply prompt for authentication upon opening, others must be opened by an admin account directly to get access to all functions (e.g. Console).

4. There are third-party applications that will not work correctly because they assume that the user account is an admin. These programs may have to be executed by logging into the admin account, or by using the `open` utility.

Setup

Accounts can be created and managed in System Preferences. On settled systems, it is generally easier to create a second admin account and then demote the first account. This avoids data migration. Newly installed systems can also just add a standard account. Demoting an account can be done either from the the new admin account in System Preferences – the other account must be logged out – or by executing these commands (it may not be necessary to execute both, see issue #179):

```
$ sudo dscl . -delete /Groups/admin GroupMembership <username>
$ sudo dscl . -delete /Groups/admin GroupMembers <GeneratedUID>
```

You can find the “GeneratedUID” of your account with:

```
$ dscl . -read /Users/<username> GeneratedUID
```

See also this post for more information about how macOS determines group membership.

Full disk encryption

FileVault provides full disk (technically, full *volume*) encryption on macOS.

FileVault encryption protects data at rest and hardens (but not always prevents) someone with physical access from stealing data or tampering with your Mac.

With much of the cryptographic operations happening efficiently in hardware, the performance penalty for FileVault is not noticeable.

Like all cryptosystems, the security of FileVault greatly depends on the quality of the pseudo random number generator (PRNG).

> The random device implements the Yarrow pseudo random number generator algorithm and maintains its entropy pool. Additional entropy is fed to the generator regularly by the SecurityServer daemon from random jitter measurements of the kernel.

> SecurityServer is also responsible for periodically saving some entropy to disk and reloading it during startup to provide entropy in early system operation.

See `man 4 random` for more information.

Turning on FileVault in System Preferences **after** installing macOS, rather than creating an encrypted partition for the installation first, is more secure, because more PRNG entropy is available then.

Additionally, the PRNG can be manually seeded with entropy by writing to /dev/random **before** enabling FileVault. This can be done by simply using the Mac for a little while before activating FileVault.

To manually seed entropy *before* enabling FileVault:

```
$ cat > /dev/random
[Type random letters for a long while, then press Control-D]
```

Enable FileVault with `sudo fdesetup enable` or through **System Preferences** > **Security & Privacy** and reboot.

If you can remember your password, there's no reason to save the **recovery key**. However, your encrypted data will be lost forever if you can't remember the password or recovery key.

If you want to know more about how FileVault works, see the paper Infiltrate the Vault: Security Analysis and Decryption of Lion Full Disk Encryption (pdf) and related presentation (pdf). Also see IEEE Std 1619-2007 “The XTS-AES Tweakable Block Cipher” (pdf).

You may wish to enforce **hibernation** and evict FileVault keys from memory instead of traditional sleep to memory:

```
$ sudo pmset -a destroyfvkeyonstandby 1
$ sudo pmset -a hibernatemode 25
```

> All computers have firmware of some type—EFI, BIOS—to help in the discovery of hardware components and ultimately to properly bootstrap the computer using the desired OS instance. In the case of Apple hardware and the use of EFI, Apple stores relevant information within EFI to aid in the functionality of macOS. For example, the FileVault key is stored in EFI to transparently come out of standby mode.

> Organizations especially sensitive to a high-attack environment, or potentially exposed to full device access when the device is in standby mode, should mitigate this risk by destroying the FileVault key in firmware. Doing so doesn’t destroy the use of FileVault, but simply requires the user to enter the password in order for the system to come out of standby mode.

If you choose to evict FileVault keys in standby mode, you should also modify your standby and power nap settings. Otherwise, your machine may wake while in standby mode and then power off due to the absence of the FileVault key. See issue #124 for more information. These settings can be changed with:

```
$ sudo pmset -a powernap 0
$ sudo pmset -a standby 0
$ sudo pmset -a standbydelay 0
$ sudo pmset -a autopoweroff 0
```

For more information, see Best Practices for Deploying FileVault 2 (pdf) and paper Lest We Remember: Cold Boot Attacks on Encryption Keys (pdf)

Firewall

Before connecting to the Internet, it's a good idea to first configure a firewall.

There are several types of firewall available for macOS.

Application layer firewall

Built-in, basic firewall which blocks **incoming** connections only.

Note, this firewall does not have the ability to monitor, nor block **outgoing** connections.

It can be controlled by the **Firewall** tab of **Security & Privacy** in **System Preferences**, or with the following commands.

Enable the firewall:

```
$ sudo /usr/libexec/ApplicationFirewall/socketfilterfw --setglobalstate on
```

Enable logging:

```
$ sudo /usr/libexec/ApplicationFirewall/socketfilterfw --setloggingmode on
```

You may also wish to enable stealth mode:

```
$ sudo /usr/libexec/ApplicationFirewall/socketfilterfw --setstealthmode on
```

> Computer hackers scan networks so they can attempt to identify computers to attack. You can prevent your computer from responding to some of these scans by using **stealth mode**. When stealth mode is enabled, your computer does not respond to ICMP ping requests, and does not answer to connection attempts from a closed TCP or UDP port. This makes it more difficult for attackers to find your computer.

Finally, you may wish to prevent *built-in software* as well as *code-signed, downloaded software from being whitelisted automatically*:

```
$ sudo /usr/libexec/ApplicationFirewall/socketfilterfw --setallowsigned off

$ sudo /usr/libexec/ApplicationFirewall/socketfilterfw --setallowsignedapp off
```

> Applications that are signed by a valid certificate authority are automatically added to the list of allowed apps, rather than prompting the user to authorize them. Apps included in macOS are signed by Apple and are allowed to receive incoming connections when this setting is enabled. For example, since iTunes is already signed by Apple, it is automatically allowed to receive incoming connections through the firewall.

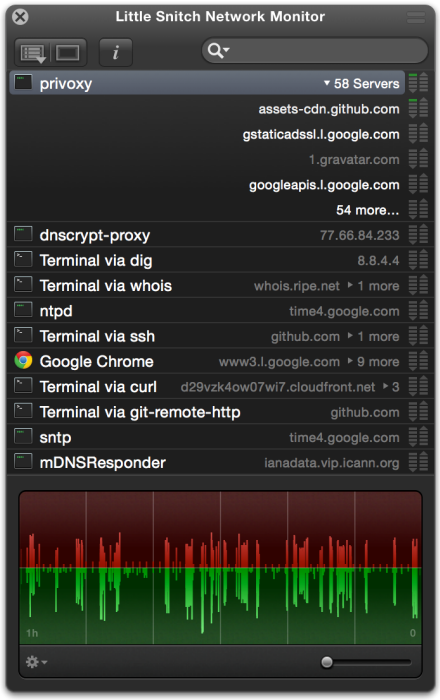
> If you run an unsigned app that is not listed in the firewall list, a dialog appears with options to Allow or Deny connections for the app. If you choose "Allow", macOS signs the application and automatically adds it to the firewall list. If you choose "Deny", macOS adds it to the list but denies incoming connections intended for this app.

After interacting with `socketfilterfw`, you may want to restart (or terminate) the process:

```
$ sudo pkill -HUP socketfilterfw
```

Third party firewalls

Programs such as Little Snitch, Hands Off, Radio Silence and Security Growler provide a good balance of usability and security.



Example of Little Snitch-monitored session

```
LittleSnitch-4.0.5.dmg
SHA-256: a954a269596c9a8e9efb3efadf843a6ae419fe218145c5b8d877e2acb0692981
SHA-1:   f642900c9c4f82a0fec38a0c826133e54cfbc0dc
```

These programs are capable of monitoring and blocking **incoming** and **outgoing** network connections. However, they may require the use of a closed source kernel extension.

If the number of choices of allowing/blocking network connections is overwhelming, use **Silent Mode** with connections allowed, then periodically check your settings to gain understanding of what various applications are doing.

It is worth noting that these firewalls can be bypassed by programs running as **root** or through OS vulnerabilities (pdf), but they are still worth having - just don't expect absolute protection. However, some malware actually deletes itself and doesn't execute if Little Snitch, or other security software, is installed.

For more on how Little Snitch works, see the Network Kernel Extensions Programming Guide and Shut up snitch! – reverse engineering and exploiting a critical Little Snitch vulnerability.

Kernel level packet filtering

A highly customizable, powerful, but also most complicated firewall exists in the kernel. It can be controlled with `pfctl` and various configuration files.

pf can also be controlled with a GUI application such as IceFloor or Murus.

There are many books and articles on the subject of pf firewall. Here's is just one example of blocking traffic by IP address.

Add the following into a file called `pf.rules`, modifying `en0` to be your outbound network adapter:

```
set block-policy drop
set fingerprints "/etc/pf.os"
set ruleset-optimization basic
set skip on lo0
scrub in all no-df
table <blocklist> persist
block in log
block in log quick from no-route to any
pass out proto tcp from any to any keep state
pass out proto udp from any to any keep state
pass out proto icmp from any to any keep state
```

```
block log on en0 from {<blocklist>} to any
block log on en0 from any to {<blocklist>}
```

Then use the following commands to manipulate the firewall:

- `sudo pfctl -e -f pf.rules` to enable the firewall
- `sudo pfctl -d` to disable the firewall
- `sudo pfctl -t blocklist -T add 1.2.3.4` to an IP address to the blocklist
- `sudo pfctl -t blocklist -T show` to view the blocklist
- `sudo ifconfig pflog0 create` to create an interface for logging
- `sudo tcpdump -ni pflog0` to view the filtered packets.

Unless you're already familiar with packet filtering, spending too much time configuring pf is not recommended. It is also probably unnecessary if your Mac is behind a NAT on a secure home network.

It is possible to use the pf firewall to block network access to entire ranges of network addresses, for example to a whole organization:

Query Merit RADb for the list of networks in use by an autonomous system, like Facebook:

```
$ whois -h whois.radb.net '!gAS32934'
```

Copy and paste the list of networks returned into the blocklist command:

```
$ sudo pfctl -t blocklist -T add 31.13.24.0/21 31.13.64.0/24 157.240.0.0/16
```

Confirm the addresses were added:

```
$ sudo pfctl -t blocklist -T show
No ALTQ support in kernel
ALTQ related functions disabled
   31.13.24.0/21
   31.13.64.0/24
   157.240.0.0/16
```

Confirm network traffic is blocked to those addresses (note that DNS requests will still work):

```
$ dig a +short facebook.com
157.240.2.35

$ curl --connect-timeout 5 -I http://facebook.com/
*   Trying 157.240.2.35...
* TCP_NODELAY set
* Connection timed out after 5002 milliseconds
* Closing connection 0
curl: (28) Connection timed out after 5002 milliseconds

$ sudo tcpdump -tqni pflog0 'host 157.240.2.35'
IP 192.168.1.1.62771 > 157.240.2.35.80: tcp 0
IP 192.168.1.1.62771 > 157.240.2.35.80: tcp 0
IP 192.168.1.1.62771 > 157.240.2.35.80: tcp 0
IP 192.168.1.1.62771 > 157.240.2.35.80: tcp 0
IP 192.168.1.1.162771 > 157.240.2.35.80: tcp 0
```

Outgoing TCP SYN packets are blocked, so a TCP connection is not established and thus a Web site is effectively blocked at the IP layer.

To use pf to audit "phone home" behavior of user and system-level processes, see fix-macosx/net-monitor.

Services

Before you connect to the Internet, you may wish to disable some system services, which use up resources or phone home to Apple.

See fix-macosx/yosemite-phone-home, l1k/osxparanoia and karek314/macOS-home-call-drop for further recommendations.

Services on macOS are managed by **launchd**. See launchd.info, as well as Apple's Daemons and Services Programming Guide and Technical Note TN2083

You can also run KnockKnock that shows more information about startup items.

- Use `launchctl list` to view running user agents
- Use `sudo launchctl list` to view running system daemons
- Specify the service name to examine it, e.g. `launchctl list com.apple.Maps.mapspushd`
- Use `defaults read` to examine job plists in `/System/Library/LaunchDaemons` and `/System/Library/LaunchAgents`
- Use `man` , `strings` and Google to learn about what the agent/daemon runs

For example, to learn what a system launch daemon or agent does, start with:

```
$ defaults read /System/Library/LaunchDaemons/com.apple.apsd.plist
```

Look at the `Program` or `ProgramArguments` section to see which binary is run, in this case `apsd` . To find more information about that, look at the man page with `man apsd`

For example, if you're not interested in Apple Push Notifications, disable the service:

```
$ sudo launchctl unload -w /System/Library/LaunchDaemons/com.apple.apsd.plist
```

Note Unloading services may break usability of some applications. Read the manual pages and use Google to make sure you understand what you're doing first.

Be careful about disabling any system daemons you don't understand, as it may render your system unbootable. If you break your Mac, use single user mode to fix it.

Use Console and Activity Monitor applications if you notice your Mac heating up, feeling sluggish, or generally misbehaving, as it may have resulted from your tinkering.

To view currently disabled services:

```
$ find /var/db/com.apple.xpc.launchd/ -type f -print -exec defaults read {} \; 2>/dev/null
```

Annotated lists of launch daemons and agents, the respective program executed, and the programs' hash sums are included in this repository.

(Optional) Run the `read_launch_plists.py` script and `diff` output to check for any discrepancies on your system, e.g.:

```
$ diff <(python read_launch_plists.py) <(cat 16A323_launchd.csv)
```

See also cirrusj.github.io/Yosemite-Stop-Launch for descriptions of services and Provisioning OS X and Disabling Unnecessary Services for another explanation.

Spotlight Suggestions

Disable **Spotlight Suggestions** in both the Spotlight preferences and Safari's Search preferences to avoid your search queries being sent to Apple.

Also disable **Bing Web Searches** in the Spotlight preferences to avoid your search queries being sent to Microsoft.

See fix-macosx.com for detailed instructions.

> If you've upgraded to OS X 10.10 "Yosemite" and you're using the default settings, each time you start typing in Spotlight (to open an application or search for a file on your computer), your local search terms and location are sent to Apple and third parties (including Microsoft).

Note This Web site and instructions may no longer work on macOS Sierra - see issue 164.

To download, view and apply their suggested fixes:

```
$ curl -O https://fix-macosx.com/fix-macosx.py

$ less fix-macosx.py

$ python fix-macosx.py
All done. Make sure to log out (and back in) for the changes to take effect.
```

For comparison, also see https://fix10.isleaked.com/

Homebrew

Consider using Homebrew to make software installations easier and to update userland tools (see Apple's great GPL purge).

Note If you have not already installed Xcode or Command Line Tools, use `xcode-select --install` to download and install them from Apple.

To install Homebrew:

```
$ mkdir homebrew && curl -L https://github.com/Homebrew/brew/tarball/master | tar xz --strip 1 -C homebrew
```

Edit `PATH` in your shell or shell rc file to use `~/homebrew/bin` and `~/homebrew/sbin`. For example, `echo 'PATH=$PATH:~/homebrew/sbin:~/homebrew/bin' >> .zshrc`, then change your login shell to Z shell with `chsh -s /bin/zsh`, open a new Terminal window and run `brew update`.

Homebrew uses SSL/TLS to talk with GitHub and verifies checksums of downloaded packages, so it's fairly secure.

Remember to periodically run `brew update` and `brew upgrade` on trusted and secure networks to download and install software updates. To get information on a package before installation, run `brew info <package>` and check its recipe online.

According to Homebrew's Anonymous Aggregate User Behaviour Analytics, Homebrew gathers anonymous aggregate user behaviour analytics and reporting these to Google Analytics.

To opt out of Homebrew's analytics, you can set `export HOMEBREW_NO_ANALYTICS=1` in your environment or shell rc file, or use `brew analytics off`.

You may also wish to enable additional security options, such as `HOMEBREW_NO_INSECURE_REDIRECT=1` and `HOMEBREW_CASK_OPTS=--require-sha`.

DNS

Hosts file

Use the hosts file to block known malware, advertising or otherwise unwanted domains.

Edit the hosts file as root, for example with `sudo vi /etc/hosts`. The hosts file can also be managed with the GUI app 2ndalpha/gasmask.

To block a domain, append `0 example.com` or `0.0.0.0 example.com` or `127.0.0.1 example.com` to `/etc/hosts`

Note IPv6 uses the `AAAA` DNS record type, rather than `A` record type, so you may also want to block those connections by *also* including `::1 example.com` entries, like shown here.

There are many lists of domains available online which you can paste in, just make sure each line starts with `0`, `0.0.0.0`, `127.0.0.1`, and the line `127.0.0.1 localhost` is included.

For hosts lists, see someonewhocares.org, l1k/osxparanoia/blob/master/hosts, StevenBlack/hosts and gorhill/uMatrix/hosts-files.json.

To append a list of hosts from a list, use the `tee` command, then confirm by editing `/etc/hosts` or counting the number of lines in it:

```
$ curl "https://raw.githubusercontent.com/StevenBlack/hosts/master/hosts" | sudo tee -a /etc/hosts

$ wc -l /etc/hosts
47476
```

```
$ egrep -ve "^#|^255.255.255|^0.0.0.0|^127.0.0.1|^0 " /etc/hosts | sort | uniq | sort
::1 localhost
fe80::1%lo0 localhost
[should not return any other IP addresses]
```

See `man hosts` and FreeBSD Configuration Files for more information.

See the dnsmasq section of this guide for more hosts blocking options.

DNSCrypt

To encrypt outgoing DNS traffic, consider using dnscrypt. In combination with Dnsmasq and DNSSEC, the security of both outbounding and inbounding dns traffic are strengthened.

If you prefer a GUI application, see alterstep/dnscrypt-osxclient. Below are the guide for installation and configuration of the command-line DNSCrypt.

Install DNSCrypt from Homebrew:

```
$ brew install dnscrypt-proxy
```

If using in combination with Dnsmasq, find the file `homebrew.mxcl.dnscrypt-proxy.plist` by running

```
$ brew info dnscrypt-proxy
```

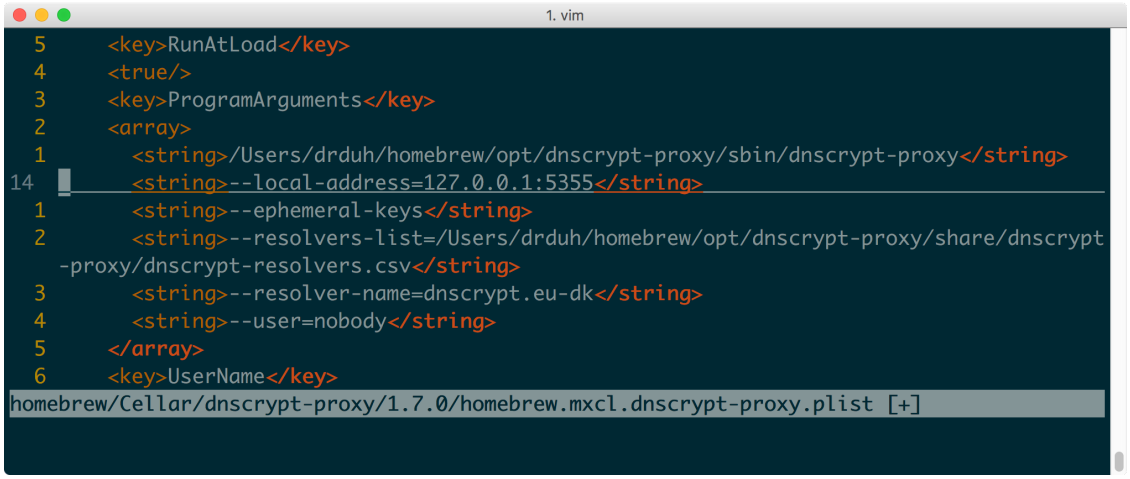
which will shows the location like `/usr/local/Cellar/dnscrypt-proxy/1.9.5_1` and `homebrew.mxcl.dnscrypt-proxy.plist` is in this folder.

Edit it to have the line:

```
<string>--local-address=127.0.0.1:5355</string>
```

Below the line:

```
<string>/usr/local/opt/dnscrypt-proxy/sbin/dnscrypt-proxy</string>
```



Append a local-address line to use DNScrypt on a port other than 53, like 5355

This can also be done using Homebrew, by installing `gnu-sed` and using the `gsed` command:

```
$ sudo gsed -i "/sbin\\/dnscrypt-proxy<\\/string>/a<string>--local-address=127.0.0.1:5355<\\/string>\n"
$(find ~/homebrew -name homebrew.mxcl.dnscrypt-proxy.plist)
```

By default, the `resolvers-list` will point to the dnscrypt version specific resolvers file. When dnscrypt is updated, this version may no longer exist, and if it does, may point to an outdated file. This can be fixed by changing the resolvers file in `homebrew.mxcl.dnscrypt-proxy.plist` (found earlier using find) to the symlinked version in `/usr/local/share`:

```
<string>--resolvers-list=/usr/local/share/dnscrypt-proxy/dnscrypt-resolvers.csv</string>
```

Below the line:

```
<string>/usr/local/opt/dnscrypt-proxy/sbin/dnscrypt-proxy</string>
```

Start DNSCrypt:

```
$ sudo brew services restart dnscrypt-proxy
```

Make sure DNSCrypt is running:

```
$ sudo lsof -Pni UDP:5355
COMMAND      PID   USER   FD   TYPE             DEVICE SIZE/OFF NODE NAME
dnscrypt-  13415 nobody    6u  IPv4 0x1773f85ff9f8bbef      0t0  UDP 127.0.0.1:5355

$ ps A | grep '[d]nscrypt'
13415   ??  Ss    13:57.21 /usr/local/opt/dnscrypt-proxy/sbin/dnscrypt-proxy --local-
address=127.0.0.1:5355 --ephemeral-keys --resolvers-list=/usr/local/share/dnscrypt-proxy/dnscrypt-
resolvers.csv --resolver-name=d0wn-us-ns4 --user=nobody
```

> By default, dnscrypt-proxy runs on localhost (127.0.0.1), port 53, and under the "nobody" user using the dnscrypt.eu-dk DNSCrypt-enabled resolver. If you would like to change these settings, you will have to edit the plist file (e.g., --resolver-address, --provider-name, --provider-key, etc.)

This can be accomplished by editing `homebrew.mxcl.dnscrypt-proxy.plist`

You can run your own dnscrypt server (see also drduh/Debian-Privacy-Server-Guide#dnscrypt) from a trusted location or use one of many public servers instead.

Confirm outgoing DNS traffic is encrypted:

```
$ sudo tcpdump -qtni en0
IP 10.8.8.8.59636 > 107.181.168.52: UDP, length 512
IP 107.181.168.52 > 10.8.8.8.59636: UDP, length 368

$ dig +short -x 128.180.155.106.49321
d0wn-us-ns4
```

dnscrypt-proxy also has the capability to blacklist domains, including the use of wildcards. See the Sample configuration file for dnscrypt-proxy for the options.

Note Applications and programs may resolve DNS using their own provided servers. If dnscrypt-proxy is used, it is possible to disable all other, non-dnscrypt DNS traffic with the following pf rules:

```
block drop quick on !lo0 proto udp from any to any port = 53
block drop quick on !lo0 proto tcp from any to any port = 53
```

See also What is a DNS leak, the mDNSResponder manual page and ipv6-test.com.

Dnsmasq

Among other features, dnsmasq is able to cache replies, prevent upstreaming queries for unqualified names, and block entire TLDs.

Use in combination with DNSCrypt to additionally encrypt outgoing DNS traffic.

If you don't wish to use DNSCrypt, you should at least use DNS not provided by your ISP. Two popular alternatives are Google DNS and OpenDNS.

(Optional) DNSSEC is a set of extensions to DNS which provide to DNS clients (resolvers) origin authentication of DNS data, authenticated denial of existence, and data integrity. All answers from DNSSEC protected zones are digitally signed. The signed records are authenticated via a chain of trust, starting with a set of verified public keys for the DNS root-zone. The current root-zone trust anchors may be downloaded from IANA website. There are a number of resources on DNSSEC, but probably the best one is dnssec.net website.

Install Dnsmasq (DNSSEC is optional):

```
$ brew install dnsmasq --with-dnssec
$ cp /usr/local/etc/dnsmasq.conf.default /usr/local/etc/dnsmasq.conf
```

Edit the configuration:

```
$ vim /usr/local/etc/dnsmasq.conf
```

Examine all the options. Here are a few recommended settings to enable:

```
# Forward queries to DNSCrypt on localhost port 5355
server=127.0.0.1#5355

# Uncomment to forward queries to Google Public DNS, if DNSCrypt is not used
# You may also use your own DNS server or other public DNS server you trust
#server=8.8.8.8
#server=8.8.4.4

# Never forward plain (local) names
domain-needed

# Examples of blocking TLDs or subdomains
#address=/.onion/0.0.0.0
#address=/.local/0.0.0.0
#address=/.mycoolnetwork/0.0.0.0
#address=/.facebook.com/0.0.0.0
#address=/.push.apple.com/0.0.0.0

# Never forward addresses in the non-routed address spaces
bogus-priv

# Reject private addresses from upstream nameservers
stop-dns-rebind

# Query servers in order
strict-order

# Set the size of the cache
# The default is to keep 150 hostnames
cache-size=8192

# Optional logging directives
log-async
log-dhcp
log-facility=/var/log/dnsmasq.log

# Log all queries
#log-queries

# Path to list of additional hosts
#addn-hosts=/etc/blacklist

# Enable DNSSEC (see https://www.iana.org/dnssec/files)
#dnssec
#trust-anchor=.,19036,8,2,49AAC11D7B6F6446702E54A1607371607A1A41855200FD2CE1CDDE32F24E8FB5
#dnssec-check-unsigned
```

Install and start the program (sudo is required to bind to privileged port 53):

```
$ sudo brew services start dnsmasq
```

To set Dnsmasq as your local DNS server, open **System Preferences** > **Network** and select the active interface, then the **DNS** tab, select **+** and add `127.0.0.1`, or use:

```
$ sudo networksetup -setdnsservers "Wi-Fi" 127.0.0.1
```

Make sure Dnsmasq is correctly configured:

```
$ scutil --dns
DNS configuration

resolver #1
  search domain[0] : whatever
  nameserver[0] : 127.0.0.1
  flags    : Request A records, Request AAAA records
  reach    : Reachable, Local Address, Directly Reachable Address

$ networksetup -getdnsservers "Wi-Fi"
127.0.0.1
```

Note Some VPN software overrides DNS settings on connect. See issue #24 for more information.

Test DNSSEC validation

Test DNSSEC validation succeeds for signed zones:

```
$ dig +dnssec icann.org
```

Reply should have `NOERROR` status and contain `ad` flag. For instance,

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 47039
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1
```

Test DNSSEC validation fails for zones that are signed improperly:

```
$ dig www.dnssec-failed.org
```

Reply should have `SERVFAIL` status. For instance,

```
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 15190
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1
```

Captive portal

When macOS connects to new networks, it **probes** the network and launches a Captive Portal assistant utility if connectivity can't be determined.

An attacker could trigger the utility and direct a Mac to a site with malware without user interaction, so it's best to disable this feature and log in to captive portals using your regular Web browser, provided you have first disable any custom dns and/or proxy settings.

```
$ sudo defaults write /Library/Preferences/SystemConfiguration/com.apple.captive.control Active -bool
false
```

See also Apple OS X Lion Security: Captive Portal Hijacking Attack, Apple's secret "wispr" request, How to disable the captive portal window in Mac OS Lion, and An undocumented change to Captive Network Assistant settings in OS X 10.10 Yosemite.

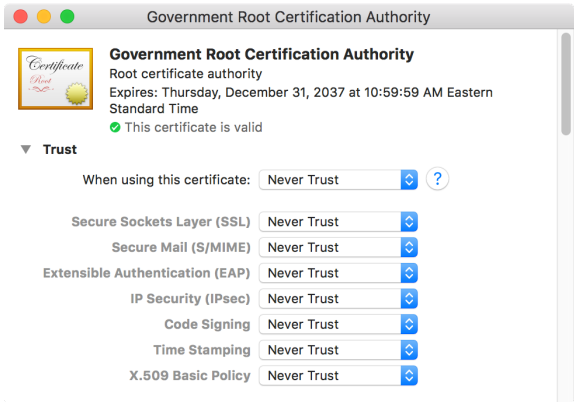
Certificate authorities

macOS comes with over 200 root authority certificates installed from for-profit corporations like Apple, Verisign, Thawte, Digicert and government agencies from China, Japan, Netherlands, U.S., and more! These Certificate Authorities (CAs) are capable of issuing SSL/TLS certificates for any domain, code signing certificates, etc.

For more information, see Certification Authority Trust Tracker, Analysis of the HTTPS certificate ecosystem (pdf), and You Won't Be Needing These Any More: On Removing Unused Certificates From Trust Stores (pdf).

You can inspect system root certificates in **Keychain Access**, under the **System Roots** tab or by using the `security` command line tool and `/System/Library/Keychains/SystemRootCertificates.keychain` file.

You can disable certificate authorities through Keychain Access by marking them as **Never Trust** and closing the window:



The risk of a man in the middle attack in which a coerced or compromised certificate authority trusted by your system issues a fake/rogue SSL certificate is quite low, but still possible.

OpenSSL

The version of OpenSSL in Sierra is `0.9.8zh` which is not current. It doesn't support TLS 1.1 or newer, elliptic curve ciphers, and more.

Apple declares OpenSSL **deprecated** in their Cryptographic Services Guide document. Their version also has patches which may surprise you.

If you're going to use OpenSSL on your Mac, download and install a recent version of OpenSSL with `brew install openssl` . Note, linking brew to be used in favor of `/usr/bin/openssl` may interfere with built-in software. See issue #39.

Compare the TLS protocol and cipher between the homebrew version and the system version of OpenSSL:

```
$ ~/homebrew/bin/openssl version; echo | ~/homebrew/bin/openssl s_client -connect github.com:443 2>&1 |
grep -A2 SSL-Session
OpenSSL 1.0.2j  26 Sep 2016
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES128-GCM-SHA256

$ /usr/bin/openssl version; echo | /usr/bin/openssl s_client -connect github.com:443 2>&1 | grep -A2 SSL-
Session
OpenSSL 0.9.8zh 14 Jan 2016
SSL-Session:
    Protocol  : TLSv1
    Cipher    : AES128-SHA
```

See also Comparison of TLS implementations, How's My SSL, Qualys SSL Labs Tools and for detailed explanations and with latest vulnerabilities tests ssl-checker.online-domain-tools.com.

Curl

The version of Curl which comes with macOS uses Secure Transport for SSL/TLS validation.

If you prefer to use OpenSSL, install with `brew install curl --with-openssl` and ensure it's the default with `brew link --force curl`

Here are several recommended options to add to `~/.curlrc` (see `man curl` for more):

```
user-agent = "Mozilla/5.0 (Windows NT 6.1; rv:45.0) Gecko/20100101 Firefox/45.0"
referer = ";auto"
connect-timeout = 10
progress-bar
max-time = 90
verbose
show-error
remote-time
ipv4
```

Web

Privoxy

Consider using Privoxy as a local proxy to filter Web browsing traffic.

Note macOS proxy settings are not universal; apps and services may or may not honor system proxy settings. Ensure the app you wish to proxy is correctly configured and manually verify connections don't leak. Additionally, it may be possible to configure the *pf* firewall to transparently proxy all traffic.

A signed installation package for privoxy can be downloaded from silvester.org.uk or Sourceforge. The signed package is more secure than the Homebrew version, and attracts full support from the Privoxy project.

Alternatively, install and start privoxy using Homebrew:

```
$ brew install privoxy

$ brew services start privoxy
```

By default, privoxy listens on local TCP port 8118.

Set the system **HTTP** proxy for your active network interface `127.0.0.1` and `8118` (This can be done through **System Preferences** > **Network** > **Advanced** > **Proxies**):

```
$ sudo networksetup -setwebproxy "Wi-Fi" 127.0.0.1 8118
```

(Optional) Set the system **HTTPS** proxy, which still allows for domain name filtering, with:

```
$ sudo networksetup -setsecurewebproxy "Wi-Fi" 127.0.0.1 8118
```

Confirm the proxy is set:

```
$ scutil --proxy
<dictionary> {
  ExceptionsList : <array> {
    0 : *.local
    1 : 169.254/16
  }
  FTPPassive : 1
  HTTPEnable : 1
  HTTPPort : 8118
  HTTPProxy : 127.0.0.1
}
```

Visit http://p.p/ in a browser, or with Curl:

```
$ ALL_PROXY=127.0.0.1:8118 curl -I http://p.p/
HTTP/1.1 200 OK
Content-Length: 2401
Content-Type: text/html
Cache-Control: no-cache
```

Privoxy already comes with many good rules, however you can also write your own.

Edit `~/homebrew/etc/privoxy/user.action` to filter elements by domain or with regular expressions.

Here are some examples:

```
{ +block{social networking} }
www.facebook.com/(extern|plugins)/(login_status|like(box)?|activity|fan)\.php
.facebook.com

{ +block{unwanted images} +handle-as-image }
.com/ads/
/.*1x1.gif
/.*fb-icon.[jpg|gif|png]
/assets/social-.*
/cleardot.gif
/img/social.*
ads.*.co.*/
ads.*.com/

{ +redirect{s@http://@https://@} }
.google.com
.wikipedia.org
code.jquery.com
imgur.com
```

Verify Privoxy is blocking and redirecting:

```
$ ALL_PROXY=127.0.0.1:8118 curl ads.foo.com/ -IL
HTTP/1.1 403 Request blocked by Privoxy
Content-Type: image/gif
Content-Length: 64
Cache-Control: no-cache

$ ALL_PROXY=127.0.0.1:8118 curl imgur.com/ -IL
HTTP/1.1 302 Local Redirect from Privoxy
Location: https://imgur.com/
Content-Length: 0
Date: Sun, 09 Oct 2016 18:48:19 GMT

HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
```

You can replace ad images with pictures of kittens, for example, by starting the a local Web server and redirecting blocked requests to localhost.

Browser

The Web browser poses the largest security and privacy risk, as its fundamental job is to download and execute untrusted code from the Internet. This is an important statement. The unique use case of Web Browsers of operation in hostile environments, has forced them to adopt certain impressive security features. The cornerstone of Web Browser security is the Same Origin Policy (SOP). In a few words, SOP prevents a malicious script on one page from obtaining access to sensitive data on another web page through that page's Document Object Model (DOM). If SOP is compromised, the security of the whole Web Browser is compromised.

The best tip to ensure secure browsing regardless your choice of Web Browser is proper security hygiene. The majority of Web Browser exploits require social engineering attacks to achieve native code execution. Always be mindful of the links you click and be extra careful when websites ask you to download and install software. 99% percent of the time that software is malware.

Another important consideration about Web Browser security is Web Extensions. Web Extensions greatly increase the attack surface of the Web Browser. This is an issue that plagues Firefox and Chrome alike. Luckily, Web Extensions can only access specific browser APIs that are being governed by their manifest. That means we can quickly audit their behavior and remove them if they request access to information they shouldn't (why would an Ad blocker require camera access?). In the interest of security, it is best to limit your use of Web Extensions.

Google Chrome , Firefox and Safari are the Web Browsers that are being covered in this guide. Each Web Browser offers certain benefits and drawbacks regarding their security and privacy. It is best to make an informed choice before committing to one.

Google Chrome

Google Chrome is based on the Open Source Chromium project with certain proprietary components. The proprietary components are the following:

1. Automatic updates through the GoogleSoftwareUpdateDaemon.
2. Usage tracking and crash reporting, which can be disabled through Chrome's settings.
3. Chrome Web Store
4. Non-optional tracking. Google Chrome installer includes a randomly generated token. The token is sent to Google after the installation completes in order to measure the success rate. The RLZ identifier stores information – in the form of encoded strings – like the source of chrome download and installation week. It doesn't include any personal information and it's used to measure the effectiveness of a promotional campaign. **Chrome downloaded from Google's website doesn't have the RLZ identifier**. The source code to decode the strings is made open by Google.
5. Adobe Flash Plugin. Google Chrome supports a Pepper API version of Adobe Flash which gets updated automatically with Chrome.
6. Media Codec support. Adds support for proprietary codecs.
7. Chrome's PDF viewer.

Chrome offers separate profiles, sandboxing, frequent updates (including Flash, although you should disable it - see below), and carries impressive credentials. In addition, Google offers a very lucrative bounty program for reporting vulnerabilities along with its own Project Zero. This means that a large number of highly talented and motivated people are constantly auditing Chrome's code base.

Chrome offers account sync between multiple devices. Part of the sync data are stored website logins. The login passwords are encrypted and in order to access them, a user's Google account password is required. You can use your Google account to sign to your Chrome customized settings from other devices while retaining your the security of your passwords.

Chrome's Web store for extensions requires a $5 dollar lifetime fee in order to submit extensions. The low cost allows the development of many quality Open Source Web Extensions that do not aim to monetize through usage.

Chrome has the largest share of global usage and is the preferred target platform for the majority of developers. Major technologies are based on Chrome's Open Source components, such as node.js which uses Chrome's V8 Engine and the Electron framework, which is based on Chromium and node.js. Chrome's vast user base makes it the most attractive target for threat actors and security researchers. Despite under constants attacks, Chrome has retained an impressive security track record over the years. This is not a small feat.

To improve the privacy and security posture of the browser, create at least three profiles, one for browsing **trusted** Web sites (email, banking), another for **mostly trusted** Web sites (link aggregators, news sites), and a third for a completely **cookie-less** and **script-less** experience.

- One profile **without cookies or Javascript** enabled (e.g., turned off in `chrome://settings/content`) which should be the preferred profile to visiting untrusted Web sites. However, many pages will not load at all without Javascript enabled.
- One profile with uMatrix or uBlock Origin (or both). Use this profile for visiting **mostly trusted** Web sites. Take time to learn how these firewall extensions work. Other frequently recommended extensions are Privacy Badger, HTTPSEverywhere and CertPatrol (Firefox only).
- One or more profile(s) for secure and trusted browsing needs, such as banking and email only.

The idea is to separate and compartmentalize data so that an exploit or privacy violation in one "session" does not necessarily affect data in another.

In each profile, visit `chrome://settings/content` and enable **Block sites from running Flash** so Flash applications do not run by default without explicit permission.

Incognito mode in Chrome disables extensions, since extensions such as Ad blockers have access to Chrome's network requests. Extensions have to be enabled manually. Moreover, while in Incognito mode, Chrome does not use session data from previous sessions. Incognito mode is another option if you want to access sensitive information without setting up separate profiles.

Take some time to read through Chromium Security and Chromium Privacy.

For example, you may wish to disable DNS prefetching (see also DNS Prefetching and Its Privacy Implications (pdf)).

It is best to remember that Google is an advertising company and its major source of revenue is AdSense. It makes sense that an advertising company would leverage its services to collect information and use that information to maximize its profit. That means that while using Google services certain personal information are being stored. Google is open about the data it stores and how it used them. Users can opt out from many of those services and see what type of information Google has stored from their account settings.

Firefox

Firefox is an excellent browser as well as being completely open source. Currently, Firefox is in a renaissance period. It replaces major parts of its infrastructure and code base under projects Quantum and Photon. Part of the Quantum project is to replace C++ code with Rust. Rust is a systems programming language with a focus on security and thread safety. It is expected that Rust adoption will greatly improve the overall security posture of Firefox.

Firefox offers a similar security model to Chrome. It offers bounty program, although it is not a lucrative as Chrome's. Firefox follows a six-week release cycle similar to Chrome.

See discussion in issues #2, #90 for more information about certain differences in Firefox and Chrome.

If using Firefox, see TheCreeper/PrivacyFox, pyllyukko/user.js and ghacksuserjs/ghacks-user.js for recommended privacy preferences and other hardening measures. Also be sure to check out NoScript for Mozilla-based browsers, which allows whitelist-based, pre-emptive script blocking.

Firefox is focussed on user privacy. It supports tracking protection during Private browsing by default. The tracking protection can be enabled for the default account, although it may break the browsing experience on some websites. Another feature for added privacy unique to Firefox is Containers. Containers lets you create profiles in Firefox for different activities, such as online shopping, travel planning, or checking work email. Containers store cookies separately, you can log into the same site with a different account in each Container, and online trackers can't connect your browsing in one container to another.

Previous versions of Firefox used a Web Extension SDK that was quite invasive and offered immense freedom to developers. Sadly, that freedom also introduced a number of vulnerabilities in Firefox that greatly affected its users. You can find more information about vulnerabilities introduced by Firefox's legacy extensions in this paper. Currently, Firefox only supports Web Extensions through the Web Extension Api, which is very similar to Chrome's.

Submission of Web Extensions is Firefox is free. Web Extensions in Firefox most of the time are Open Source, although certain Web Extensions are proprietary.

Note. Similar to Chrome and Safari, Firefox allows account sync across multiple devices. While stored login passwords are encrypted, Firefox does not require a password to reveal their plain text format. Firefox only displays as yes/no prompt. This is an important security issue. Keep that in mind if you sign in to your Firefox account from devices that do not belong to you and leave them unattended. The issue has been raised among the Firefox community and hopefully will be resolved in the coming versions.

Safari

Safari is the default Web Browser of macOS. It is also the most optimized browser for reducing battery use. Safari, like Chrome, has both Open Source and proprietary components. Safari is based on the open source Web Engine WebKit, which is ubiquitous among the macOS ecosystem. WebKit is used by Apple apps such as Mail, iTunes, iBooks, and the App store. Chrome's Blink engine is a fork of WebKit and both engines share a number of similarities.

Safari supports certain unique features that benefit user security and privacy. Content blockers enables the creation of content blocking rules without using Javascript. This rule based approach greatly improves memory user, security, and privacy. Safari 11 will introduce an Intelligent Tracking Prevention system. This feature will automatically remove tracking data stored in Safari after a period of non-interaction by the user from the tracker's website.

Similar to Chrome and Firefox, Safari offers an invite only bounty program for bug reporting to a select number of security researchers. The bounty program was announced during Apple's presentation at BlackHat 2016.

Web Extensions in Safari have an additional option to use native code in the Safari's sandbox environment, in addition to Web Extension APIs. Web Extensions in Safari are also distributed through Apple's App store. App store submission comes with the added benefit of Web Extension code being audited by Apple. On the other hand App store submission comes at a steep cost. Yearly developer subscription fee costs $100 (in contrast to Chrome's $5 lifetime fee and Firefox's free submission). The high cost is prohibitive for the majority of Open Source developers. As a result, Safari has very few extensions to choose from. However, you should keep the high cost in mind when installing extensions. It is expected that most Web Extensions will have some way of monetizing usage in order to cover developer costs. And be extra careful when the Web Extension's source code is not Open Source. On a side note, some Safari extensions are Open Source and freely available. Be grateful to those developers.

Safari syncs user's preferences and stored logins through the iCloud Keychain. Stored passwords are encrypted with 256-bit AES . In order to be viewed in plain text, a user must input the account password of the current device. This means that users can sync data across devices with added security.

Safari follows a slower release cycle than Chrome and Firefox (3-4 minor releases, 1 major release, per year). Newer features are slower to be adopted to the stable channel. Although security updates in Safari are handled independent of the stable release schedule and issued automatically through the App store. The Safari channel that follows a six-week release cycle (similar to as Chrome and Firefox) is called Safari Technology Preview and it is the recommended option instead of the stable channel of Safari.

An excellent open source ad blocker for Safari that fully leverages Content blockers is Ka-Block. Ka-Block is focussed on user privacy. The only time the extension makes a network connection is when a new version of the extension is released. You can view the extension's repository here.

Other Web Browsers

Many Chromium-derived browsers are not recommended. They are usually closed source, poorly maintained, have bugs, and make dubious claims to protect privacy. See The Private Life of Chromium Browsers.

Other miscellaneous browsers, such as Brave, are not evaluated in this guide, so are neither recommended nor actively discouraged from use.

Web Browsers and Privacy

All Web Browsers retain certain information about our browsing habits. That information is used for a number of reasons. One of them is to improve the overall performance of the Web Browser. Most Web Browsers offer predictions services to resolve typos or URL redirections, store analytics data of browsing patterns, crash reports and black listing of known malicious servers. Those options can be turned on and off from each Web Browser's settings panel.

Since Web Browsers execute untrusted code from the server, it is important to understand what type of information can be accessed. The Navigator interface gives access to information about the Web Browsers user agent. Those include information such as the operating system, Websites' permissions, and the device's battery level. For more information about security conscious browsing and what type of information is being "leaked" by your browser, see HowTo: Privacy & Security Conscious Browsing, browserleaks.com and EFF Panopticlick.

To hinder third party trackers, it is recommended to disable third-party cookies from your Web Browser settings. A third party cookie is a cookie associated with a file requested by different domain than the one the user is currently viewing. Most of the time third party are used to create browsing profiles by tracking a user's movement on the web. Disabling third-party cookies prevents HTTP responses and scripts from other domains from setting cookies. Moreover, cookies are removed from requests to domains that are not the document origin domain, so cookies are only sent to the current site that is being viewed.

Also be aware of WebRTC, which may reveal your local or public (if connected to VPN) IP address(es). This can be disabled with extensions such as uBlock Origin and rentamob/WebRTC-Leak-Prevent.

Plugins

Adobe Flash, **Oracle Java**, **Adobe Reader**, **Microsoft Silverlight** (Netflix now works with HTML5) and other plugins are security risks and should not be installed.

If they are necessary, only use them in a disposable virtual machine and subscribe to security announcements to make sure you're always patched.

See Hacking Team Flash Zero-Day, Java Trojan BackDoor.Flashback, Acrobat Reader: Security Vulnerabilities, and Angling for Silverlight Exploits, for example.

PGP/GPG

PGP is a standard for encrypting email end to end. That means only the chosen recipients can decrypt a message, unlike regular email which is read and forever archived by providers.

GPG, or **GNU Privacy Guard**, is a GPL licensed program compliant with the standard.

GPG is used to verify signatures of software you download and install, as well as symmetrically or asymmetrically encrypt files and text.

Install from Homebrew with `brew install gnupg`.

If you prefer a graphical application, download and install GPG Suite.

Here are several recommended options to add to `~/.gnupg/gpg.conf`:

```
auto-key-locate keyserver
keyserver hkps://hkps.pool.sks-keyservers.net
keyserver-options no-honor-keyserver-url
personal-cipher-preferences AES256 AES192 AES CAST5
personal-digest-preferences SHA512 SHA384 SHA256 SHA224
default-preference-list SHA512 SHA384 SHA256 SHA224 AES256 AES192 AES CAST5 ZLIB BZIP2 ZIP Uncompressed
cert-digest-algo SHA512
s2k-digest-algo SHA512
s2k-cipher-algo AES256
charset utf-8
fixed-list-mode
no-comments
no-emit-version
keyid-format 0xlong
list-options show-uid-validity
verify-options show-uid-validity
with-fingerprint
```

These settings will configure GnuPG to use SSL when fetching new keys and prefer strong cryptographic primitives.

See also ioerror/duraconf/configs/gnupg/gpg.conf. You should also take some time to read OpenPGP Best Practices.

If you don't already have a keypair, create one using `gpg --gen-key`. Also see drduh/YubiKey-Guide.

Read online guides and practice encrypting and decrypting email to yourself and your friends. Get them interested in this stuff!

OTR

OTR stands for **off-the-record** and is a cryptographic protocol for encrypting and authenticating conversations over instant messaging.

You can use OTR on top of any existing XMPP chat service, even Google Hangouts (which only encrypts conversations between users and the server using TLS).

The first time you start a conversation with someone new, you'll be asked to verify their public key fingerprint. Make sure to do this in person or by some other secure means (e.g. GPG encrypted mail).

A popular macOS GUI client for XMPP and other chat protocols is Adium.

```
Adium_1.5.10.4.dmg
SHA-256: 31fa3fd32b86dd3381b60e0d5aafbc2a9452036f0fb4963bffbc2a6c64a9458b
SHA-1:   8a674a642447839ea287aed528194e4fd32763b8
```

Remember to disable logging for off the record chats with Adium.

A good console-based XMPP client is profanity, which can be installed with `brew install profanity`

For improved anonymity, check out Tor Messenger, although it is still in beta, as well as Ricochet (which has recently received a thorough security audit (pdf)), which both use the Tor network rather than relying on messaging servers.

If you want to know how OTR works, read the paper Off-the-Record Communication, or, Why Not To Use PGP (pdf)

Tor

Tor is an anonymizing proxy which can be used for browsing the Web.

Download Tor Browser from the offical Tor Project Web site.

Do **not** attempt to configure other browsers or applications to use Tor as you will likely make a mistake which will compromise your anonymity.

Download both the `dmg` and `asc` signature files, then verify the disk image has been signed by Tor developers:

```
$ cd ~/Downloads

$ file Tor*
TorBrowser-7.0.10-osx64_en-US.dmg:     bzip2 compressed data, block size = 900k
TorBrowser-7.0.10-osx64_en-US.dmg.asc: PGP signature Signature (old)

$ gpg Tor*asc
gpg: assuming signed data in 'TorBrowser-7.0.10-osx64_en-US.dmg'
gpg: Signature made Thu Nov  9 08:58:11 2017 PST
gpg:                using RSA key 0xD1483FA6C3C07136
gpg: Can't check signature: No public key

$ gpg --recv 0x4E2C6E8793298290
gpg: key 0x4E2C6E8793298290: public key "Tor Browser Developers (signing key) <torbrowser@torproject.org>"
imported
gpg: no ultimately trusted keys found
gpg: Total number processed: 1
gpg:               imported: 1

$ gpg --verify Tor*asc
gpg: assuming signed data in 'TorBrowser-7.0.10-osx64_en-US.dmg'
gpg: Signature made Thu Nov  9 08:58:11 2017 PST
gpg:                using RSA key 0xD1483FA6C3C07136
gpg: Good signature from "Tor Browser Developers (signing key) <torbrowser@torproject.org>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: EF6E 286D DA85 EA2A 4BA7  DE68 4E2C 6E87 9329 8290
     Subkey fingerprint: A430 0A6B C93C 0877 A445  1486 D148 3FA6 C3C0 7136
```

Make sure `Good signature from "Tor Browser Developers (signing key) <torbrowser@torproject.org>"` appears in the output. The warning about the key not being certified is benign, as it has not yet been manually assigned trust.

See How to verify signatures for packages for more information.

To finish installing Tor Browser, open the disk image and drag the it into the Applications folder, or with:

```
$ hdiutil mount TorBrowser-7.0.10-osx64_en-US.dmg

$ cp -rv /Volumes/Tor\ Browser/TorBrowser.app /Applications
```

Verify the Tor application's code signature was made by with The Tor Project's Apple developer ID **MADPSAYN6T**, using the `spctl -a -v` and/or `pkgutil --check-signature` commands:

```
$ spctl -a -vv /Applications/TorBrowser.app
/Applications/TorBrowser.app: accepted
source=Developer ID
origin=Developer ID Application: The Tor Project, Inc (MADPSAYN6T)

$ pkgutil --check-signature /Applications/TorBrowser.app
Package "TorBrowser.app":
   Status: signed by a certificate trusted by Mac OS X
   Certificate Chain:
    1. Developer ID Application: The Tor Project, Inc (MADPSAYN6T)
       SHA1 fingerprint: 95 80 54 F1 54 66 F3 9C C2 D8 27 7A 29 21 D9 61 11 93 B3 E8
       ------------------------------------------------------------------------------
    2. Developer ID Certification Authority
       SHA1 fingerprint: 3B 16 6C 3B 7D C4 B7 51 C9 FE 2A FA B9 13 56 41 E3 88 E1 86
       ------------------------------------------------------------------------------
```

```
    3. Apple Root CA
       SHA1 fingerprint: 61 1E 5B 66 2C 59 3A 08 FF 58 D1 4A E2 24 52 D1 98 DF 6C 60
```

You may also use the `codesign` command to examine an application's code signature:

```
$ codesign -dvv /Applications/TorBrowser.app
Executable=/Applications/TorBrowser.app/Contents/MacOS/firefox
Identifier=org.torproject.torbrowser
Format=app bundle with Mach-O thin (x86_64)
CodeDirectory v=20200 size=249 flags=0x0(none) hashes=5+3 location=embedded
Library validation warning=OS X SDK version before 10.9 does not support Library Validation
Signature size=4247
Authority=Developer ID Application: The Tor Project, Inc (MADPSAYN6T)
Authority=Developer ID Certification Authority
Authority=Apple Root CA
Signed Time=Nov 9, 2017, 12:47:58 AM
Info.plist entries=22
TeamIdentifier=MADPSAYN6T
Sealed Resources version=2 rules=12 files=130
Internal requirements count=1 size=188
```

To view full certificate details, extract them with `codesign` and decode it with `openssl` :

```
$ codesign -d --extract-certificates /Applications/TorBrowser.app
Executable=/Applications/TorBrowser.app/Contents/MacOS/firefox

$ file codesign*
codesign0: data
codesign1: data
codesign2: data

$ openssl x509 -inform der -in codesign0 -subject -issuer -startdate -enddate -noout
subject= /UID=MADPSAYN6T/CN=Developer ID Application: The Tor Project, Inc (MADPSAYN6T)/OU=MADPSAYN6T
/O=The Tor Project, Inc/C=US
issuer= /CN=Developer ID Certification Authority/OU=Apple Certification Authority/O=Apple Inc./C=US
notBefore=Apr 12 22:40:13 2016 GMT
notAfter=Apr 13 22:40:13 2021 GMT

$ openssl x509 -inform der -in codesign0  -fingerprint -noout
SHA1 Fingerprint=95:80:54:F1:54:66:F3:9C:C2:D8:27:7A:29:21:D9:61:11:93:B3:E8

$ openssl x509 -inform der -in codesign0 -fingerprint -sha256 -noout
SHA256
Fingerprint=B5:0D:47:F0:3E:CB:42:B6:68:1C:6F:38:06:2B:C2:9F:41:FA:D6:54:F1:29:D3:E4:DD:9C:C7:49:35:FF:F5:D9
```

Tor traffic is **encrypted** to the exit node (i.e., cannot be read by a passive network eavesdropper), but Tor use **can** be identified - for example, TLS handshake "hostnames" will show up in plaintext:

```
$ sudo tcpdump -An "tcp" | grep "www"
listening on pktap, link-type PKTAP (Apple DLT_PKTAP), capture size 262144 bytes
............". ...www.odezz26nvv7jeqz1xghzs.com........
............#.!...www.bxbko3qi7vacgwyk4ggulh.com........
.6....m.....>...:........|../*
Z....W....X=..6...C../.................................0...0..0.......'....F./0..
*.H........0%1#0!..U....www.b6zazzahl3h3faf4x2.com0...160402000000Z..170317000000Z0'1%0#..U....www.tm3ddrghe2
```

See Tor Protocol Specification and Tor/TLSHistory for more information.

You may wish to additionally obfuscate Tor traffic using a pluggable transport, such as Yawning/obfs4proxy or SRI-CSL/stegotorus.

This can be done by setting up your own Tor relay or finding an existing private or public bridge to serve as an obfuscating entry node.

For extra security, use Tor inside a VirtualBox or VMware virtualized GNU/Linux or BSD machine.

Finally, remember the Tor network provides anonymity, which is not necessarily synonymous with privacy. The Tor network does not guarantee protection against a global observer capable of traffic analysis and correlation. See also Seeking Anonymity in an Internet Panopticon (pdf) and Traffic Correlation on Tor by Realistic Adversaries (pdf).

Also see Invisible Internet Project (I2P) and its Tor comparison.

VPN

If you use your Mac on untrusted networks - airports, cafes, etc. - your network traffic is being monitored and possibly tampered with.

It is a good idea to use a VPN which encrypts **all** outgoing network traffic (i.e., not **split tunnel**) with a provider you trust. For an example of how to set up and host your own VPN, see drduh/Debian-Privacy-Server-Guide.

Don't just blindly sign up for a VPN service without understanding the full implications and how your traffic will be routed. If you don't understand how the VPN works or are not familiar with the software used, you are probably better off without it.

When choosing a VPN service or setting up your own, be sure to research the protocols, key exchange algorithms, authentication mechanisms, and type of encryption being used. Some protocols, such as PPTP, should be avoided in favor of OpenVPN, for example.

Some clients may send traffic over the next available interface when VPN is interrupted or disconnected. See scy/8122924 for an example on how to allow traffic only over VPN.

Another set of scripts to lock down your system so it will only access the internet via a VPN can be found as part of the Voodoo Privacy project - sarfata/voodooprivacy and there is an updated guide to setting up an IPSec VPN on a virtual machine (hwdsl2/setup-ipsec-vpn) or a docker container (hwdsl2/docker-ipsec-vpn-server).

Viruses and malware

There is an ever-increasing amount of Mac malware in the wild. Macs aren't immune from viruses and malicious software!

Some malware comes bundled with both legitimate software, such as the Java bundling Ask Toolbar, and some with illegitimate software, such as Mac.BackDoor.iWorm bundled with pirated programs. Malwarebytes Anti-Malware for Mac is an excellent program for ridding oneself of "garden-variety" malware and other "crapware".

See Methods of malware persistence on Mac OS X (pdf) and Malware Persistence on OS X Yosemite to learn about how garden-variety malware functions.

You could periodically run a tool like Knock Knock to examine persistent applications (e.g. scripts, binaries). But by then, it is probably too late. Maybe applications such as Block Block and Ostiarius will help. See warnings and caveats in issue #90 first, however.

Anti-virus programs are a double-edged sword -- not useful for **advanced** users and will likely increase attack surface against sophisticated threats, however possibly useful for catching "garden variety" malware on **novice** users' Macs. There is also the additional processing overhead to consider.

See Sophail: Applied attacks against Antivirus (pdf), Analysis and Exploitation of an ESET Vulnerability, a trivial Avast RCE, Popular Security Software Came Under Relentless NSA and GCHQ Attacks, How Israel Caught Russian Hackers Scouring the World for U.S. Secrets and AVG: "Web TuneUP" extension multiple critical vulnerabilities.

Therefore, the best anti-virus is **Common Sense 2018**. See more discussion in issue #44.

CylancePROTECT may be worth running for the exploit mitigation features and (when locked down) is much harder to locally bypass than traditional AV, but it's effectiveness at detecting malware on MacOS is questionable. It's core feature is an algorithm derived from a machine-learning process which aims to identify malware based on various characteristics of a binary executable. Cylance have a whitepaper with information about how it works. Single licenses are available from third party resellers such as Cyberforce or Malware Managed and there is also a home/personal edition in the works but it is currently only available for companies to make available to their employees. On MacOS it complements Apple's built-in XProtect by continuously vmmap'ing the memory of active processes to watch for patterns that indicate bad things happening.

Local privilege escalation bugs are plenty on macOS, so always be careful when downloading and running untrusted programs or trusted programs from third party websites or downloaded over HTTP (example).

Have a look at The Safe Mac for past and Malwarebytes Blog for current Mac security news.

Also check out Hacking Team malware for Mac OS: root installation for MacOS, Support driver for Mac Agent and RCS Agent for Mac, which is a good example of advanced malware with capabilities to hide from **userland** (e.g., `ps` , `ls`), for example. For more, see A Brief Analysis of an RCS Implant Installer and reverse.put.as

System Integrity Protection

System Integrity Protection (SIP) is a security feature since OS X 10.11 "El Capitan". It is enabled by default, but can be disabled, which may be necessary to change some system settings, such as deleting root certificate authorities or unloading certain launch daemons. Keep this feature on, as it is by default.

From What's New in OS X 10.11:

> A new security policy that applies to every running process, including privileged code and code that runs out of the sandbox. The policy extends additional protections to components on disk and at run-time, only allowing system binaries to be modified by the system installer and software updates. Code injection and runtime attachments to system binaries are no longer permitted.

Also see What is the “rootless” feature in El Capitan, really?

Some MacBook hardware has shipped with SIP disabled. To verify SIP is enabled, use the command `csrutil status`, which should return: `System Integrity Protection status: enabled.` Otherwise, enable SIP through Recovery Mode.

Gatekeeper and XProtect

Gatekeeper and the **quarantine** system try to prevent unsigned or "bad" programs and files from running and opening.

XProtect prevents the execution of known bad files and outdated plugin versions, but does nothing to cleanup or stop existing malware.

Both offer trivial protection against common risks and are fine at default settings.

See also Mac Malware Guide : How does Mac OS X protect me? and Gatekeeper, XProtect and the Quarantine attribute.

Note Quarantine stores information about downloaded files in `~/Library/Preferences/com.apple.LaunchServices.QuarantineEventsV2`, which may pose a privacy risk. To examine the file, simply use `strings` or the following command:

```
$ echo 'SELECT datetime(LSQuarantineTimeStamp + 978307200, "unixepoch") as LSQuarantineTimeStamp,
LSQuarantineAgentName, LSQuarantineOriginURLString, LSQuarantineDataURLString from LSQuarantineEvent;' |
sqlite3 /Users/$USER/Library/Preferences/com.apple.LaunchServices.QuarantineEventsV2
```

See here for more information.

To permanently disable this feature, clear the file and make it immutable:

```
$ :>~/Library/Preferences/com.apple.LaunchServices.QuarantineEventsV2

$ sudo chflags schg ~/Library/Preferences/com.apple.LaunchServices.QuarantineEventsV2
```

Metadata and artifacts

macOS attaches metadata (HFS+ extended attributes) to downloaded files, which can be viewed with the `mdls` and `xattr` commands:

```
$ ls -l@ ~/Downloads/TorBrowser-6.0.8-osx64_en-US.dmg
-rw-r--r--@ 1 drduh  staff  59322237 Dec  1 12:00 TorBrowser-6.0.8-osx64_en-US.dmg
com.apple.metadata:kMDItemWhereFroms         186
com.apple.quarantine          68

$ mdls ~/Downloads/TorBrowser-6.0.8-osx64_en-US.dmg
_kMDItemOwnerUserID            = 501
kMDItemContentCreationDate     = 2016-12-01 12:00:00 +0000
kMDItemContentModificationDate = 2016-12-01 12:00:00 +0000
kMDItemContentType             = "com.apple.disk-image-udif"
kMDItemContentTypeTree         = (
    "public.archive",
    "public.item",
    "public.data",
    "public.disk-image",
```

```
    "com.apple.disk-image",
    "com.apple.disk-image-udif"
)
kMDItemDateAdded               = 2016-12-01 12:00:00 +0000
kMDItemDisplayName             = "TorBrowser-6.0.8-osx64_en-US.dmg"
kMDItemFSContentChangeDate     = 2016-12-01 12:00:00 +0000
kMDItemFSCreationDate          = 2016-12-01 12:00:00 +0000
kMDItemFSCreatorCode           = ""
kMDItemFSFinderFlags           = 0
kMDItemFSHasCustomIcon         = (null)
kMDItemFSInvisible             = 0
kMDItemFSIsExtensionHidden     = 0
kMDItemFSIsStationery          = (null)
kMDItemFSLabel                 = 0
kMDItemFSName                  = "TorBrowser-6.0.8-osx64_en-US.dmg"
kMDItemFSNodeCount             = (null)
kMDItemFSOwnerGroupID          = 5000
kMDItemFSOwnerUserID           = 501
kMDItemFSSize                  = 60273898
kMDItemFSTypeCode              = ""
kMDItemKind                    = "Disk Image"
kMDItemLogicalSize             = 60273898
kMDItemPhysicalSize            = 60276736
kMDItemWhereFroms              = (
    "https://dist.torproject.org/torbrowser/6.0.8/TorBrowser-6.0.8-osx64_en-US.dmg",
    "https://www.torproject.org/projects/torbrowser.html.en"
)

$ xattr -l TorBrowser-6.0.8-osx64_en-US.dmg
com.apple.metadata:kMDItemWhereFroms:
00000000  62 70 6C 69 73 74 30 30 A2 01 02 5F 10 4D 68 74  |bplist00..._.Mht|
00000010  74 70 73 3A 2F 2F 64 69 73 74 2E 74 6F 72 70 72  |tps://dist.torpr|
00000020  6F 6A 65 63 74 2E 6F 72 67 2F 74 6F 72 62 72 6F  |oject.org/torbro|
00000030  77 73 65 72 2F 36 2E 30 2E 38 2F 54 6F 72 42 72  |wser/6.0.8/TorBr|
00000040  6F 77 73 65 72 2D 36 2E 30 2E 38 2D 6F 73 78 36  |owser-6.0.8-osx6|
00000050  34 5F 65 6E 2D 55 53 2E 64 6D 67 5F 10 36 68 74  |4_en-US.dmg_.6ht|
00000060  74 70 73 3A 2F 2F 77 77 77 2E 74 6F 72 70 72 6F  |tps://www.torpro|
00000070  6A 65 63 74 2E 6F 72 67 2F 70 72 6F 6A 65 63 74  |ject.org/project|
00000080  73 2F 74 6F 72 62 72 6F 77 73 65 72 2E 68 74 6D  |s/torbrowser.htm|
00000090  6C 2E 65 6E 08 0B 5B 00 00 00 00 00 00 01 01 00  |l.en..[.........|
000000A0  00 00 00 00 00 00 03 00 00 00 00 00 00 00 00 00  |................|
000000B0  00 00 00 00 00 00 94                             |.......|
000000b7
com.apple.quarantine: 0081;58519ffa;Google Chrome.app;1F032CAB-F5A1-4D92-84EB-CBECA971B7BC
```

Metadata attributes can also be removed with the `-d` flag:

```
$ xattr -d com.apple.metadata:kMDItemWhereFroms ~/Downloads/TorBrowser-6.0.5-osx64_en-US.dmg

$ xattr -d com.apple.quarantine ~/Downloads/TorBrowser-6.0.5-osx64_en-US.dmg

$ xattr -l ~/Downloads/TorBrowser-6.0.5-osx64_en-US.dmg
[No output after removal.]
```

Other metadata and artifacts may be found in the directories including, but not limited to, `~/Library/Preferences/`, `~/Library/Containers/<APP>/Data/Library/Preferences`, `/Library/Preferences`, some of which is detailed below.

`~/Library/Preferences/com.apple.sidebarlists.plist` contains historical list of volumes attached. To clear it, use the command `/usr/libexec/PlistBuddy -c "delete :systemitems:VolumesList" ~/Library/Preferences/com.apple.sidebarlists.plist`

`/Library/Preferences/com.apple.Bluetooth.plist` contains Bluetooth metadata, including device history. If Bluetooth is not used, the metadata can be cleared with:

```
sudo defaults delete /Library/Preferences/com.apple.Bluetooth.plist DeviceCache
sudo defaults delete /Library/Preferences/com.apple.Bluetooth.plist IDSPairedDevices
sudo defaults delete /Library/Preferences/com.apple.Bluetooth.plist PANDevices
sudo defaults delete /Library/Preferences/com.apple.Bluetooth.plist PANInterfaces
sudo defaults delete /Library/Preferences/com.apple.Bluetooth.plist SCOAudioDevices
```

`/var/spool/cups` contains the CUPS printer job cache. To clear it, use the commands:

```
sudo rm -rfv /var/spool/cups/c0*
sudo rm -rfv /var/spool/cups/tmp/*
sudo rm -rfv /var/spool/cups/cache/job.cache*
```

To clear the list of iOS devices connected, use:

```
sudo defaults delete /Users/$USER/Library/Preferences/com.apple.iPod.plist "conn:128:Last Connect"
sudo defaults delete /Users/$USER/Library/Preferences/com.apple.iPod.plist Devices
sudo defaults delete /Library/Preferences/com.apple.iPod.plist "conn:128:Last Connect"
sudo defaults delete /Library/Preferences/com.apple.iPod.plist Devices
sudo rm -rfv /var/db/lockdown/*
```

QuickLook thumbnail data can be cleared using the `qlmanage -r cache` command, but this writes to the file `resetreason` in the Quicklook directories, and states that the Quicklook cache was manually cleared. It can also be manually cleared by getting the directory names with `getconf DARWIN_USER_CACHE_DIR` and `sudo getconf DARWIN_USER_CACHE_DIR`, then removing them:

```
rm -rfv $(getconf DARWIN_USER_CACHE_DIR)/com.apple.QuickLook.thumbnailcache/exclusive
rm -rfv $(getconf DARWIN_USER_CACHE_DIR)/com.apple.QuickLook.thumbnailcache/index.sqlite
rm -rfv $(getconf DARWIN_USER_CACHE_DIR)/com.apple.QuickLook.thumbnailcache/index.sqlite-shm
rm -rfv $(getconf DARWIN_USER_CACHE_DIR)/com.apple.QuickLook.thumbnailcache/index.sqlite-wal
rm -rfv $(getconf DARWIN_USER_CACHE_DIR)/com.apple.QuickLook.thumbnailcache/resetreason
rm -rfv $(getconf DARWIN_USER_CACHE_DIR)/com.apple.QuickLook.thumbnailcache/thumbnails.data
```

Similarly, for the root user:

```
sudo rm -rfv $(getconf DARWIN_USER_CACHE_DIR)/com.apple.QuickLook.thumbnailcache/thumbnails.fraghandler
sudo rm -rfv $(getconf DARWIN_USER_CACHE_DIR)/com.apple.QuickLook.thumbnailcache/exclusive
sudo rm -rfv $(getconf DARWIN_USER_CACHE_DIR)/com.apple.QuickLook.thumbnailcache/index.sqlite
sudo rm -rfv $(getconf DARWIN_USER_CACHE_DIR)/com.apple.QuickLook.thumbnailcache/index.sqlite-shm
sudo rm -rfv $(getconf DARWIN_USER_CACHE_DIR)/com.apple.QuickLook.thumbnailcache/index.sqlite-wal
sudo rm -rfv $(getconf DARWIN_USER_CACHE_DIR)/com.apple.QuickLook.thumbnailcache/resetreason
sudo rm -rfv $(getconf DARWIN_USER_CACHE_DIR)/com.apple.QuickLook.thumbnailcache/thumbnails.data
sudo rm -rfv $(getconf DARWIN_USER_CACHE_DIR)/com.apple.QuickLook.thumbnailcache/thumbnails.fraghandler
```

To clear Finder preferences:

```
defaults delete ~/Library/Preferences/com.apple.finder.plist FXDesktopVolumePositions
defaults delete ~/Library/Preferences/com.apple.finder.plist FXRecentFolders
defaults delete ~/Library/Preferences/com.apple.finder.plist RecentMoveAndCopyDestinations
defaults delete ~/Library/Preferences/com.apple.finder.plist RecentSearches
defaults delete ~/Library/Preferences/com.apple.finder.plist SGTRecentFileSearches
```

Additional diagnostic files may be found in the following directories - but caution should be taken before removing any, as it may break logging or cause other issues:

```
/var/db/CoreDuet/
/var/db/diagnostics/
/var/db/systemstats/
/var/db/uuidtext/
/var/log/DiagnosticMessages/
```

macOS stored preferred Wi-Fi data (including credentials) in nvram. To clear it, use the following commands:

```
sudo nvram -d 36C28AB5-6566-4C50-9EBD-CBB920F83843:current-network
sudo nvram -d 36C28AB5-6566-4C50-9EBD-CBB920F83843:preferred-networks
sudo nvram -d 36C28AB5-6566-4C50-9EBD-CBB920F83843:preferred-count
```

macOS may collect sensitive information about what you type, even if user dictionary and suggestions are off. To remove them, and prevent them from being created again, use the following commands:

```
rm -rfv "~/Library/LanguageModeling/*" "~/Library/Spelling/*" "~/Library/Suggestions/*"
chmod -R 000 ~/Library/LanguageModeling ~/Library/Spelling ~/Library/Suggestions
```

```
chflags -R uchg ~/Library/LanguageModeling ~/Library/Spelling ~/Library/Suggestions
```

QuickLook application support metadata can be cleared and locked with the following commands:

```
rm -rfv "~/Library/Application Support/Quick Look/*"
chmod -R 000 "~/Library/Application Support/Quick Look"
chflags -R uchg "~/Library/Application Support/Quick Look"
```

Document revision metadata is stored in `/.DocumentRevisions-V100` and can be cleared and locked with the following commands - caution should be taken as this may break some core Apple applications:

```
sudo rm -rfv /.DocumentRevisions-V100/*
sudo chmod -R 000 /.DocumentRevisions-V100
sudo chflags -R uchg /.DocumentRevisions-V100
```

Saved application state metadata may be cleared and locked with the following commands:

```
rm -rfv "~/Library/Saved Application State/*"
rm -rfv "~/Library/Containers/<APPNAME>/Saved Application State"
chmod -R 000 "~/Library/Saved Application State/"
chmod -R 000 "~/Library/Containers/<APPNAME>/Saved Application State"
chflags -R uchg "~/Library/Saved Application State/"
chflags -R uchg "~/Library/Containers/<APPNAME>/Saved Application State"
```

Autosave metadata can be cleared and locked with the following commands:

```
rm -rfv "~/Library/Containers/<APP>/Data/Library/Autosave Information"
rm -rfv "~/Library/Autosave Information"
chmod -R 000 "~/Library/Containers/<APP>/Data/Library/Autosave Information"
chmod -R 000 "~/Library/Autosave Information"
chflags -R uchg "~/Library/Containers/<APP>/Data/Library/Autosave Information"
chflags -R uchg "~/Library/Autosave Information"
```

The Siri analytics database, which is created even if the Siri launch agent disabled, can be cleared and locked with the following commands:

```
rm -rfv ~/Library/Assistant/SiriAnalytics.db
chmod -R 000 ~/Library/Assistant/SiriAnalytics.db
chflags -R uchg ~/Library/Assistant/SiriAnalytics.db
```

`~/Library/Preferences/com.apple.iTunes.plist` contains iTunes metadata. Recent iTunes search data may be cleared with the following command:

```
defaults delete ~/Library/Preferences/com.apple.iTunes.plist recentSearches
```

If you do not use Apple ID-linked services, the following keys may be cleared, too, using the following commands:

```
defaults delete ~/Library/Preferences/com.apple.iTunes.plist StoreUserInfo
defaults delete ~/Library/Preferences/com.apple.iTunes.plist WirelessBuddyID
```

`~/Library/Containers/com.apple.QuickTimePlayerX/Data/Library/Preferences/com.apple.QuickTimePlayerX.plist` contains all media played in QuickTime Player.

Additional metadata may exist in the following files:

```
~/Library/Containers/com.apple.appstore/Data/Library/Preferences/com.apple.commerce.knownclients.plist
~/Library/Preferences/com.apple.commerce.plist
~/Library/Preferences/com.apple.QuickTimePlayerX.plist
```

Passwords

You can generate strong passwords with OpenSSL:

```
$ openssl rand -base64 30
LK9xkjUEAemc1gV2Ux5xqku+PDmMmCbSTmwfiMRI
```

Or GPG:

```
$ gpg --gen-random -a 0 30
4/bGZL+yUEe8fOqQhF5V01HpGwFSpUPwFcU3aOWQ
```

Or `/dev/urandom` output:

```
$ dd if=/dev/urandom bs=1 count=30 2>/dev/null | base64
CbRGKASFI4eTa96NMrgyamj8dLZdFYBaqtWUSxKe
```

With control over character sets:

```
$ LANG=C tr -dc 'a-zA-Z0-9' < /dev/urandom | fold -w 40 | head -n 1
jm0iKn7ngQST8I0mMMCbbi6SKPcoUWwCb5lWEjxK

$ LANG=C tr -dc 'DrDuh0-9' < /dev/urandom | fold -w 40 | head -n 1
686672u2Dh7r754209uD312hhh23uD7u41h3875D
```

You can also generate passwords, even memorable ones, using **Keychain Access** password assistant, or a command line equivalent like anders/pwgen.

Keychains are encrypted with a PBKDF2 derived key and are a *pretty safe* place to store credentials. See also Breaking into the OS X keychain. Also be aware that Keychain does not encrypt the names corresponding to password entries.

Alternatively, you can manage an encrypted passwords file yourself with GnuPG (shameless plug for my drduh/pwd.sh password manager script).

In addition to passwords, ensure eligible online accounts, such as GitHub, Google accounts, banking, have two factor authentication enabled.

Look to Yubikey for a two factor and private key (e.g., ssh, gpg) hardware token. See drduh/YubiKey-Guide and trmm.net/Yubikey. One of two Yubikey's slots can also be programmed to emit a long, static password (which can be used in combination with a short, memorized password, for example).

In Addition to Login and other pam modules you can use Yubikey to secure your login and sudo, here is a pdf guide from Yubico. Yubikey are a bit pricey, there is cheaper alternative, but not as capable, U2F Zero. Here is a great guide to set it up

Backup

Always encrypt files locally before backing them up to external media or online services.

One way is to use a symmetric cipher with GPG and a password of your choosing.

To encrypt a directory:

```
$ tar zcvf - ~/Downloads | gpg -c > ~/Desktop/backup-$(date +%F-%H%M).tar.gz.gpg
```

To decrypt an archive:

```
$ gpg -o ~/Desktop/decrypted-backup.tar.gz -d ~/Desktop/backup-2015-01-01-0000.tar.gz.gpg && \
  tar zxvf ~/Desktop/decrypted-backup.tar.gz
```

You may also create encrypted volumes using **Disk Utility** or `hdiutil` :

```
$ hdiutil create ~/Desktop/encrypted.dmg -encryption -size 1g -volname "Name" -fs JHFS+
```

Also see the following applications and services: SpiderOak, Arq, Espionage, and restic.

Wi-Fi

macOS remembers access points it has connected to. Like all wireless devices, the Mac will broadcast all access point names it remembers (e.g., *MyHomeNetwork*) each time it looks for a network, such as when waking from sleep.

This is a privacy risk, so remove networks from the list in **System Preferences** > **Network** > **Advanced** when they're no longer needed.

Also see Signals from the Crowd: Uncovering Social Relationships through Smartphone Probes (pdf) and Wi-Fi told me everything about you (pdf).

Saved Wi-Fi information (SSID, last connection, etc.) can be found in `/Library/Preferences/SystemConfiguration/com.apple.airport.preferences.plist`

You may wish to spoof the MAC address of your network card before connecting to new and untrusted wireless networks to mitigate passive fingerprinting:

```
$ sudo ifconfig en0 ether $(openssl rand -hex 6 | sed 's%\(..\)%\1:%g; s%.$%%')
```

Note MAC addresses will reset to hardware defaults on each boot.

Also see feross/SpoofMAC.

Finally, WEP protection on wireless networks is not secure and you should favor connecting to **WPA2** protected networks only to mitigate the risk of passive eavesdroppers.

SSH

For outgoing ssh connections, use hardware- or password-protected keys, set up remote hosts and consider hashing them for added privacy.

Here are several recommended options to add to `~/.ssh/config`:

```
Host *
  PasswordAuthentication no
  ChallengeResponseAuthentication no
  HashKnownHosts yes
```

Note macOS Sierra permanently remembers SSH key passphrases by default. Append the option `UseKeyChain no` to turn this feature off.

You can also use ssh to create an encrypted tunnel to send your traffic through, which is similar to a VPN.

For example, to use Privoxy on a remote host:

```
$ ssh -C -L 5555:127.0.0.1:8118 you@remote-host.tld

$ sudo networksetup -setwebproxy "Wi-Fi" 127.0.0.1 5555

$ sudo networksetup -setsecurewebproxy "Wi-Fi" 127.0.0.1 5555
```

Or to use an ssh connection as a SOCKS proxy:

```
$ ssh -NCD 3000 you@remote-host.tld
```

By default, macOS does **not** have sshd or *Remote Login* enabled.

To enable sshd and allow incoming ssh connections:

```
$ sudo launchctl load -w /System/Library/LaunchDaemons/ssh.plist
```

Or use the **System Preferences** > **Sharing** menu.

If you are going to enable sshd, at least disable password authentication and consider further hardening your configuration.

To `/etc/sshd_config`, add:

```
PasswordAuthentication no
ChallengeResponseAuthentication no
UsePAM no
```

Confirm whether sshd is enabled or disabled:

```
$ sudo lsof -Pni TCP:22
```

Physical access

Keep your Mac physically secure at all times. Don't leave it unattended in hotels and such.

A skilled attacker with unsupervised physical access to your computer can infect the boot ROM to install a keylogger and steal your password - see Thunderstrike, for example.

A helpful tool is usbkill, which is *"an anti-forensic kill-switch that waits for a change on your USB ports and then immediately shuts down your computer"*.

Consider purchasing a privacy filter for your screen to thwart shoulder surfers.

System monitoring

OpenBSM audit

macOS has a powerful OpenBSM auditing capability. You can use it to monitor process execution, network activity, and much more.

To tail audit logs, use the `praudit` utility:

```
$ sudo praudit -l /dev/auditpipe
header,201,11,execve(2),0,Thu Sep  1 12:00:00 2015, + 195 msec,exec arg,/Applications/.evilapp
/rootkit,path,/Applications/.evilapp/rootkit,path,/Applications/.evilapp
/rootkit,attribute,100755,root,wheel,16777220,986535,0,subject,drduh,root,wheel,root,wheel,412,100005,5051173
header,88,11,connect(2),0,Thu Sep  1 12:00:00 2015, + 238 msec,argument,1,0x5,fd,socket-
inet,2,443,173.194.74.104,subject,drduh,root,wheel,root,wheel,326,100005,50331650,0.0.0.0,return,failure :
Operation now in progress,4354967105,trailer,88
header,111,11,OpenSSH login,0,Thu Sep  1 12:00:00 2015, + 16
msec,subject_ex,drduh,drduh,staff,drduh,staff,404,404,49271,::1,text,successful login
drduh,return,success,0,trailer,111,
```

See the manual pages for `audit`, `praudit`, `audit_control` and other files in `/etc/security`

Note although `man audit` says the `-s` flag will synchronize the audit configuration, it appears necessary to reboot for changes to take effect.

See articles on ilostmynotes.blogspot.com and derflounder.wordpress.com for more information.

DTrace

`iosnoop` monitors disk I/O

`opensnoop` monitors file opens

`execsnoop` monitors execution of processes

`errinfo` monitors failed system calls

`dtruss` monitors all system calls

See `man -k dtrace` for more information.

Note System Integrity Protection interferes with DTrace, so it may no longer be possible to use these tools.

Execution

`ps -ef` lists information about all running processes.

You can also view processes with **Activity Monitor**.

`launchctl list` and `sudo launchctl list` list loaded and running user and system launch daemons and agents.

Network

List open network files:

```
$ sudo lsof -Pni
```

List contents of various network-related data structures:

```
$ sudo netstat -atln
```

You can also use Wireshark from the command line.

Monitor DNS queries and replies:

```
$ tshark -Y "dns.flags.response == 1" -Tfields \
  -e frame.time_delta \
  -e dns.qry.name \
  -e dns.a \
  -Eseparator=,
```

Monitor HTTP requests and responses:

```
$ tshark -Y "http.request or http.response" -Tfields \
  -e ip.dst \
  -e http.request.full_uri \
  -e http.request.method \
  -e http.response.code \
  -e http.response.phrase \
  -Eseparator=/s
```

Monitor x509 certificates:

```
$ tshark -Y "ssl.handshake.certificate" -Tfields \
  -e ip.src \
  -e x509sat.uTF8String \
  -e x509sat.printableString \
  -e x509sat.universalString \
  -e x509sat.IA5String \
  -e x509sat.teletexString \
  -Eseparator=/s -Equote=d
```

Also see the simple networking monitoring application BonzaiThePenguin/Loading.

Binary Whitelisting

google/santa is a security software developed for Google's corporate Macintosh fleet and open sourced.

> Santa is a binary whitelisting/blacklisting system for macOS. It consists of a kernel extension that monitors for executions, a userland daemon that makes execution decisions based on the contents of a SQLite database, a GUI agent that notifies the user in case of a block decision and a command-line utility for managing the system and synchronizing the database with a server.

Santa uses the Kernel Authorization API to monitor and allow/disallow binaries from executing in the kernel. Binaries can be white- or black-listed by unique hash or signing developer certificate. Santa can be used to only allow trusted code execution, or to blacklist known malware from executing on a Mac, similar to Bit9 software for Windows.

Note Santa does not currently have a graphical user interface for managing rules. The following instructions are for advanced users only!

To install Santa, visit the Releases page and download the latest disk image, the mount it and install the contained package:

```
$ hdiutil mount ~/Downloads/santa-0.9.20.dmg

$ sudo installer -pkg /Volumes/santa-0.9.20/santa-0.9.20.pkg -tgt /
```

By default, Santa installs in "Monitor" mode (meaning, nothing gets blocked, only logged) and comes with two rules: one for Apple binaries and another for Santa software itself.

Verify Santa is running and its kernel module is loaded:

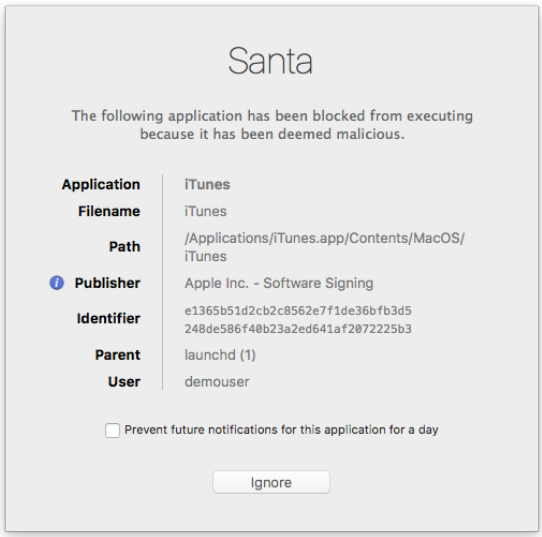
```
$ santactl status
>>> Daemon Info
  Mode                   | Monitor
  File Logging           | No
  Watchdog CPU Events    | 0  (Peak: 0.00%)
  Watchdog RAM Events    | 0  (Peak: 0.00MB)
>>> Kernel Info
  Kernel cache count     | 0
>>> Database Info
  Binary Rules           | 0
  Certificate Rules      | 2
  Events Pending Upload  | 0

$ ps -ef | grep "[s]anta"
    0   786     1   0 10:01AM ??         0:00.39 /Library/Extensions/santa-driver.kext/Contents/MacOS
/santad --syslog

$ kextstat | grep santa
  119    0 0xffffff7f822ff000 0x6000     0x6000     com.google.santa-driver (0.9.14) 693D8E4D-3161-30E0-
B83D-66A273CAE026 <5 4 3 1>
```

Create a blacklist rule to prevent iTunes from executing:

```
$ sudo santactl rule --blacklist --path /Applications/iTunes.app/
Added rule for SHA-256: e1365b51d2cb2c8562e7f1de36bfb3d5248de586f40b23a2ed641af2072225b3.
```

Try to launch iTunes - it will be blocked.

```
$ open /Applications/iTunes.app/
LSOpenURLsWithRole() failed with error -10810 for the file /Applications/iTunes.app.
```



To remove the rule:

```
$ sudo santactl rule --remove --path /Applications/iTunes.app/
```

```
Removed rule for SHA-256: e1365b51d2cb2c8562e7f1de36bfb3d5248de586f40b23a2ed641af2072225b3.
```

Open iTunes:

```
$ open /Applications/iTunes.app/
[iTunes will open successfully]
```

Create a new, example C program:

```
$ cat <<EOF > foo.c
> #include <stdio.h>
> main() { printf("Hello World\n”); }
> EOF
```

Compile the program with GCC (requires installation of Xcode or command-line tools):

```
$ gcc -o foo foo.c

$ file foo
foo: Mach-O 64-bit executable x86_64

$ codesign -d foo
foo: code object is not signed at all
```

Run it:

```
$ ./foo
Hello World
```

Toggle Santa into “Lockdown” mode, which only allows whitelisted binaries to run:

```
$ sudo defaults write /var/db/santa/config.plist ClientMode -int 2
```

Try to run the unsigned binary:

```
$ ./foo
bash: ./foo: Operation not permitted

Santa

The following application has been blocked from executing
because its trustworthiness cannot be determined.

Path:       /Users/demouser/foo
Identifier: 4e11da26feb48231d6e90b10c169b0f8ae1080f36c168ffe53b1616f7505baed
Parent:     bash (701)
```

To whitelist a specific binary, determine its SHA-256 sum:

```
$ santactl fileinfo /Users/demouser/foo
Path                 : /Users/demouser/foo
SHA-256              : 4e11da26feb48231d6e90b10c169b0f8ae1080f36c168ffe53b1616f7505baed
SHA-1                : 4506f3a8c0a5abe4cacb98e6267549a4d8734d82
Type                 : Executable (x86-64)
Code-signed          : No
Rule                 : Blacklisted (Unknown)
```

Add a whitelist rule:

```
$ sudo santactl rule --whitelist --sha256 4e11da26feb48231d6e90b10c169b0f8ae1080f36c168ffe53b1616f7505baed
Added rule for SHA-256: 4e11da26feb48231d6e90b10c169b0f8ae1080f36c168ffe53b1616f7505baed.
```

Run it:

```
$ ./foo
Hello World
```

It's allowed and works!

Applications can also be whitelisted by developer certificate (so that new binary versions will not need to be manually whitelisted on each update). For example, download and run Google Chrome - it will be blocked by Santa in "Lockdown" mode:

```
$ curl -sO https://dl.google.com/chrome/mac/stable/GGRO/googlechrome.dmg

$ hdiutil mount googlechrome.dmg

$ cp -r /Volumes/Google\ Chrome/Google\ Chrome.app /Applications/

$ open /Applications/Google\ Chrome.app/
LSOpenURLsWithRole() failed with error -10810 for the file /Applications/Google Chrome.app.
```

Whitelist the application by its developer certificate (first item in the Signing Chain):

```
$ santactl fileinfo /Applications/Google\ Chrome.app/
Path                 : /Applications/Google Chrome.app/Contents/MacOS/Google Chrome
SHA-256              : 0eb08224d427fb1d87d2276d911bbb6c4326ec9f74448a4d9a3cfce0c3413810
SHA-1                : 9213cbc7dfaaf7580f3936a915faa56d40479f6a
Bundle Name          : Google Chrome
Bundle Version       : 2883.87
Bundle Version Str   : 55.0.2883.87
Type                 : Executable (x86-64)
Code-signed          : Yes
Rule                 : Blacklisted (Unknown)
Signing Chain:
     1. SHA-256             : 15b8ce88e10f04c88a5542234fbdfc1487e9c2f64058a05027c7c34fc4201153
        SHA-1               : 85cee8254216185620ddc8851c7a9fc4dfe120ef
        Common Name         : Developer ID Application: Google Inc.
        Organization        : Google Inc.
        Organizational Unit : EQHXZ8M8AV
        Valid From          : 2012/04/26 07:10:10 -0700
        Valid Until         : 2017/04/27 07:10:10 -0700

     2. SHA-256             : 7afc9d01a62f03a2de9637936d4afe68090d2de18d03f29c88cfb0b1ba63587f
        SHA-1               : 3b166c3b7dc4b751c9fe2afab9135641e388e186
        Common Name         : Developer ID Certification Authority
        Organization        : Apple Inc.
        Organizational Unit : Apple Certification Authority
        Valid From          : 2012/02/01 14:12:15 -0800
        Valid Until         : 2027/02/01 14:12:15 -0800

     3. SHA-256             : b0b1730ecbc7ff4505142c49f1295e6eda6bcaed7e2c68c5be91b5a11001f024
        SHA-1               : 611e5b662c593a08ff58d14ae22452d198df6c60
        Common Name         : Apple Root CA
        Organization        : Apple Inc.
        Organizational Unit : Apple Certification Authority
        Valid From          : 2006/04/25 14:40:36 -0700
        Valid Until         : 2035/02/09 13:40:36 -0800
```

In this case, `15b8ce88e10f04c88a5542234fbdfc1487e9c2f64058a05027c7c34fc4201153` is the SHA-256 of Google's Apple developer certificate (team ID EQHXZ8M8AV). To whitelist it:

```
$ sudo santactl rule --whitelist --certificate --sha256
15b8ce88e10f04c88a5542234fbdfc1487e9c2f64058a05027c7c34fc4201153
Added rule for SHA-256: 15b8ce88e10f04c88a5542234fbdfc1487e9c2f64058a05027c7c34fc4201153.
```

Google Chrome should now launch, and subsequent updates to the application will continue to work as long as the code signing certificate doesn't change or expire.

To disable "Lockdown" mode:

```
$ sudo defaults delete /var/db/santa/config.plist ClientMode
```

See `/var/log/santa.log` to monitor ALLOW and DENY execution decisions.

A log and configuration server for Santa is available in Zentral, an open source event monitoring solution and TLS server for osquery and Santa. Zentral will support Santa in both MONITORING and LOCKDOWN operation mode. Clients need to be enrolled with a TLS connection to sync Santa Rules, all Santa events from endpoints are aggregated and logged back in Zentral. Santa events can trigger actions and notifications from within the Zentral Framework.

Note Python, Bash and other interpreters are whitelisted (since they are signed by Apple's developer certificate), so Santa will not be able to block such scripts from executing. Thus, a potential non-binary program which disables Santa is a weakness (not vulnerability, since it is so by design) to take note of.

Miscellaneous

If you wish, disable Diagnostics & Usage Data.

If you want to play **music** or watch **videos**, use VLC media player which is free and open source.

If you want to use **torrents**, use Transmission which is free and open source (note: like all software, even open source projects, malware may still find its way in). You may also wish to use a block list to avoid peering with known bad hosts - see Which is the best blocklist for Transmission and johntyree/3331662.

Manage default file handlers with duti, which can be installed with `brew install duti`. One reason to manage extensions is to prevent auto-mounting of remote filesystems in Finder (see Protecting Yourself From Sparklegate). Here are several recommended handlers to manage:

```
$ duti -s com.apple.Safari afp

$ duti -s com.apple.Safari ftp

$ duti -s com.apple.Safari nfs

$ duti -s com.apple.Safari smb
```

Monitor system logs with the **Console** application or `syslog -w` or `log stream` commands.

In systems prior to macOS Sierra (10.12), enable the tty_tickets flag in `/etc/sudoers` to restrict the sudo session to the Terminal window/tab that started it. To do so, use `sudo visudo` and add the line `Defaults tty_tickets`.

Set your screen to lock as soon as the screensaver starts:

```
$ defaults write com.apple.screensaver askForPassword -int 1

$ defaults write com.apple.screensaver askForPasswordDelay -int 0
```

Expose hidden files and Library folder in Finder:

```
$ defaults write com.apple.finder AppleShowAllFiles -bool true

$ chflags nohidden ~/Library
```

Show all filename extensions (so that "Evil.jpg.app" cannot masquerade easily).

```
$ defaults write NSGlobalDomain AppleShowAllExtensions -bool true
```

Don't default to saving documents to iCloud:

```
$ defaults write NSGlobalDomain NSDocumentSaveNewDocumentsToCloud -bool false
```

Enable Secure Keyboard Entry in Terminal (unless you use YubiKey or applications such as TextExpander).

Disable crash reporter (the dialog which appears after an application crashes and prompts to report the problem to Apple):

```
$ defaults write com.apple.CrashReporter DialogType none
```

Disable Bonjour multicast advertisements:

```
$ sudo defaults write /Library/Preferences/com.apple.mDNSResponder.plist NoMulticastAdvertisements -bool
YES
```

Disable Handoff and Bluetooth features, if they aren't necessary.

Consider sandboxing your applications. See fG! Sandbox Guide (pdf) and s7ephen/OSX-Sandbox--Seatbelt--Profiles.

Did you know Apple has not shipped a computer with TPM since 2006?

MacOS comes with this line in /etc/sudoers:

```
Defaults env_keep += "HOME MAIL"
```

Which stops sudo from changing the HOME variable when you elevate privileges. This means it will execute as root the bash dotfiles in the non-root user's home directory when you run "sudo bash". It is adviseable to comment this line out to avoid a potentially easy way for malware or a local attacker to escalate privileges to root.

If you want to retain the convenience of the root user having a non-root user's home directory, you can append an export line to /var/root/.bashrc, eg:

```
export HOME=/Users/blah
```

Related software

Santa - A binary whitelisting/blacklisting system for macOS.

kristovatlas/osx-config-check - checks your OSX machine against various hardened configuration settings.

Lockdown - audits and remediates security configuration settings.

Dylib Hijack Scanner - scan for applications that are either susceptible to dylib hijacking or have been hijacked.

F-Secure XFENCE (formerly Little Flocker) - "Little Snitch for files"; prevents applications from accessing files.

facebook/osquery - can be used to retrieve low level system information. Users can write SQL queries to retrieve system information.

google/grr - incident response framework focused on remote live forensics.

yelp/osxcollector - forensic evidence collection & analysis toolkit for OS X.

jipegit/OSXAuditor - analyzes artifacts on a running system, such as quarantined files, Safari, Chrome and Firefox history, downloads, HTML5 databases and localstore, social media and email accounts, and Wi-Fi access point names.

libyal/libfvde - library to access FileVault Drive Encryption (FVDE) (or FileVault2) encrypted volumes.

CISOfy/lynis - cross-platform security auditing tool and assists with compliance testing and system hardening.

Zentral - a log and configuration server for santa and osquery. Run audit and probes on inventory, events, logfiles, combine with point-in-time alerting. A full Framework and Django web server build on top of the elastic stack (formerly known as ELK stack).

Additional resources

In no particular order

MacOS Hardening Guide - Appendix of *OS Internals: Volume III - Security & Insecurity Internals (pdf)

Mac Developer Library: Secure Coding Guide

OS X Core Technologies Overview White Paper (pdf)

Reverse Engineering Mac OS X blog

Reverse Engineering Resources

Patrick Wardle's Objective-See blog

Managing Macs at Google Scale (LISA '13)

OS X Hardening: Securing a Large Global Mac Fleet (LISA '13)

DoD Security Technical Implementation Guides for Mac OS

The EFI boot process

The Intel Mac boot process

Userland Persistence on Mac OS X

Developing Mac OSX kernel rootkits

IOKit kernel code execution exploit

Hidden backdoor API to root privileges in Apple OS X

IPv6 Hardening Guide for OS X

Harden the World: Mac OSX 10.11 El Capitan

Hacker News discussion

Hacker News discussion 2

Apple Open Source

OS X 10.10 Yosemite: The Ars Technica Review

CIS Apple OSX 10.10 Benchmark (pdf)

How to Switch to the Mac

Security Configuration For Mac OS X Version 10.6 Snow Leopard (pdf)

EFF Surveillance Self-Defense Guide

MacAdmins on Slack

iCloud security and privacy overview

Demystifying the DMG File Format

There's a lot of vulnerable OS X applications out there (Sparkle Framework RCE)

iSeeYou: Disabling the MacBook Webcam Indicator LED

Mac OS X Forensics - Technical Report (pdf)

Mac Forensics: Mac OS X and the HFS+ File System (pdf)

Extracting FileVault 2 Keys with Volatility

Auditing and Exploiting Apple IPC

Mac OS X and iOS Internals: To the Apple's Core by Jonathan Levin

Demystifying the i-Device NVMe NAND (New storage used by Apple)

The macOS Phishing Easy Button: AppleScript Dangers

Over The Air - Vol. 2, Pt. 1: Exploiting The Wi-Fi Stack on Apple Devices

The Great DOM Fuzz-off of 2017

Remote code execution, git, and OS X

OSX.Pirrit Mac Adware Part III: The DaVinci Code

How to make macOS Spotlight fuck the fuck off and do your bidding