MacOS 安全加固指南

译者：rodster(看雪 ID：leixyou)、everettjf

MacOS 安全加固指南

前言

尽管 MacOS 默认配置依然十分自由，但是 MacOS 通常强制加固安全等级到接近 IOS 的程度（但仍然不是十分牢固）。

任何安全加固都影响系统性能和可用性。高度建议在充分地在所有的产品环境应用这些方法之前，先在测试系统中逐渐尝试这些推荐方法

这里有许多的加固方法和一些包括已存在自动化工具（例如 osx-config-check）的指南（例如 CIS APPLE OS X 安全指标）。然而，几乎只要将投入一点点到一些建议上（例如禁止浏览器中的 javascript 就能提高安全--激励天真无邪的用户进入到十九世纪九十年代的怀旧的万维网）。这些建议在这里提出在尽可能将用户烦恼和所遭受的痛苦减少时，要尽力加强全面安全的态度。这里同样有一些被公开文档所忽视的建议。我尽力去关注“内建（built-in）”和“开箱即用（out-of-box）”的函数和设置，并且避免涉及反病毒、反恶意软件或者指明已存在第三方产品以及他们的实用性。

因为途径不同，我最初不考虑在我书中添加一个这样的指南。然而，随着 sebastien Volpe 的问题我意思到为这本书添加一个非正式的总结是一个很好的补充，我非常感激 sebas 为本指南所做的一切。我同样地对 Amit Serper(@0xAmit)表达我最大的感激，他/她在出版之前审核了这篇文献并且分享了他/她一些深刻的见解！

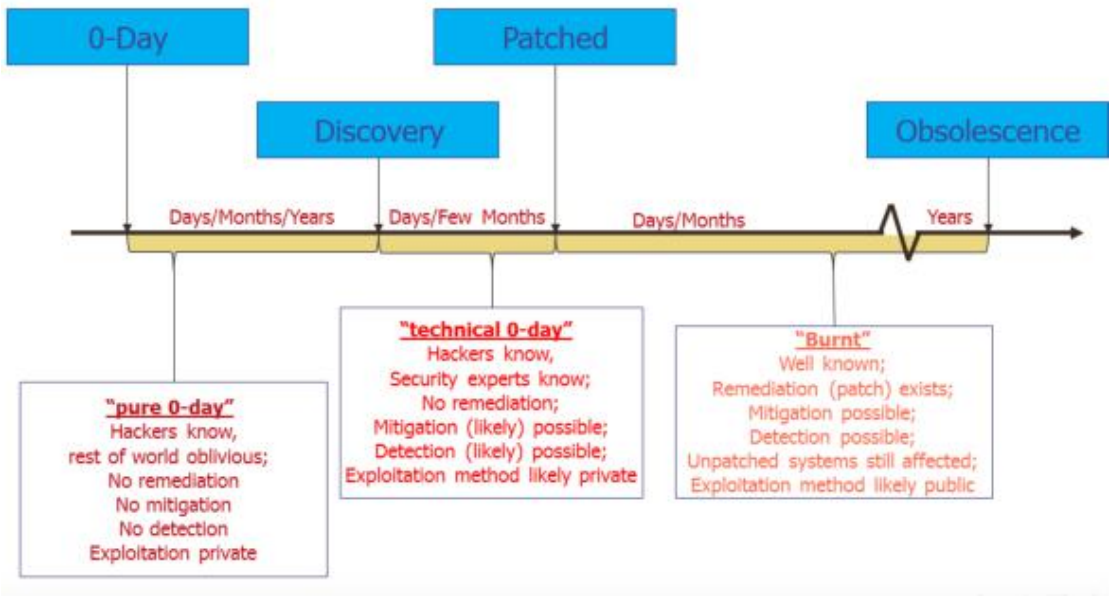
这是“*OS Internals:Volume III-Security&Insecurity”（*OS 操作系统内部:第三卷——安全与不安全）的附件

Patching, Patching and Patching

如果你没有阅读过在本卷第 12 章深度解析 MacOS 缺陷，那就让我们打破这个悬念，让你知道它的结论：核心操作系统中的缺陷是不可避免的，并且你将自动受到它们的影响。尽管安全防范措施概述了纵观附录能够起到一定地帮助，但是他们会在面对单个的内核漏洞利用全部失败。

对于缺陷点来说没有简单的解决办法，并且对于所有 0-day 漏洞没有应对措施。然而，一旦 0-day 漏洞被披露，它们就会（通常）迅速地被修补好。但是补丁是没有价值的，除非补丁被应用。仅仅当补丁被应用时才能让缺陷点被移除，不再是一个风险。

图例 1：缺陷点生存周期



然而，作为图例 1 所示，当每个系统被打上补丁或者更新时，移除掉一个缺陷点是完全能够实现的。尽管对于平常安全意识高的用户不是一个问题，但是对于也许是几百上千的机器并且尽力保持更新版本的公司环境来说会是一个重大的挑战。结果在这样的环境中发现老旧过时的操作系统一点也不罕见，在这些操作系统中也许会藏着使用 exploit-db.com 中脚本能够简单地利用的缺陷点。

幸运的是，MacOS 中软件更新的执行是非交互式的、在命令行下的，因此能够被自动运行执行。使用软件更新工具的-i（--install）和-r（--recommended）选项不需要用户交互便下载和安装好能找得到的补丁，因此，用户没有干涉进程的权限。

注意，苹果“放弃”和停止支持 MacOS 老版本的做法并不罕见，即使苹果知道缺陷点存在。一个好例子就是 MacOS10.10，它的最新（也是-最终）版本-10.10.5 仍然有许多 muymacho 和 tpwn 漏洞利用点，这些在本书前面部分描述过。尽管有些时候会提示你安全更新，但苹果通常认为大众会更新至 MacOS 10.11 (和现在的 12)，因此不会操心一个提供缺陷点的一个简单的补丁。

Logging and Auditing

小心地监视 MacOS 日志和审计的子系统，它们通常会在 hack 成功之前提示警告：一些黑客攻击行为会在第一次尝试时尝试“get it right”，但是仅仅通过观察失败尝试的记录和它们能侦查到的不正常的活动是不够的。

syslog/asl

苹果的日志系统（直到 MacOS10.12）都遵照 UN*X syslog(1)加上一些苹果特殊扩展（被成为 ASL，苹果系统日志）的结构。扩展在使维持传统机制的兼容性和第三方服务的同时，使更大程度上显示信息和过滤成为可能。

Syslog 最强大的特点之一就是能够记录远程主机。Syslog 绑定 UDP514 端口，并且能在/etc/syslogd.conf 配置，/etc/syslogd.conf 使用一个“@”作为标识物（伴随 IP 或者主机名），同样在 DNS 或者/etc/hosts 中标明了一个 loghost 入口。远程主机必须运行 syslogd 且网络可用，因为在 MacOS X 默认日志记录是越过本地主机和非网络套接字的。

远程日志记录提供两种不同的优势：

Centralized logging（集中记录）： 对于单台服务器大大地简化了日志监测，能够自动加载第三方工具或者复合 UN*X 标准的公共工具 grep(1),awk(1),perl(1)和其他过滤器。

Write-only access: 如果登录主机在网络上不是可访问的（例如没有 SSH,远程登录，或者或者其他的工具），记录会被添加进日志，而且日志不可读、不可移动。这大大地增加了安全性，因为攻击者不能该得到日志收集任何配置和敏感信息。此外这使得日志更可信，

因为作为攻击者不能抹去和修改任何之前记入的记录。注意，一个攻击者仍然能够用伪造记录来淹没日志，但是不能撤销先前任何记录。

log (MacOS 12+)

MacOS 12 反对 syslog/asl 支持心得 os_log 子系统。这是一个更强大的基础设施，它丢弃了传统的基于记录的文本文件，支持通常地记录在内存日志（in-memory logging）和数据库。显然，如果苹果决定在 os_log 上实现 syslog(3)和 asl(3) API，可以预见经过一段时间， ASL 将被整个地遗弃，

os_log 子系统不支持网络日志记录（在写入的时候）。然而，这是一个相当简单的问题，运行 log(1)客户端命令和用管道输送它的输出给 nc(1)：

```
log –stream | nc remote.log.host ###
```

然而，这仅仅是重定向输出基础方法，还有一个更加弹性的解决方法-认为应该接受网络故障事件和可量测性。

Enabling Auditing

MacOS 的最强大的安全特征就是不可抵赖的审计性。尽管不具有前瞻性，但是跟踪安全敏感操作和实时的事件还是很有效的。不像上述所提及的日志记录子系统，它请求应用程序所生成的志愿记录，审计记录被操作系统自身所生成。

尽管记录直接地从内核空间到审计日志，但是审计的一个主要缺点是它的本地性质。如果一个系统被破坏，它的审计日志就不能认为具有可信价值。幸运地是，一点点 UN*X shell 脚本创造性就能重定向审计记录到一个服务器。与 nc(1)欺骗能够应用在 log(1)上面一样也能应用在/dev/duditpipe 上。事实上，审计管道的日志也许被 praudit(1)管理，使其成为一个二进制流，而不是第一次转变给人类可读的格式。在这里，一个更具有弹性的包装器（在 shell 脚本或者其他的里面）被推荐。

> Superaudit 工具能够从本书的网站上获得（但是需要允许合法使用的许可证），这个工具内嵌了网络功能。同样有能力在/dev/auditpipe 上设置不同于默认策略的过滤器。它准许减少影响系统性能的快速审计，因为更少的审计记录

审计策略的精确规范在推荐的范围之外，因为它也许很依赖于组织策略。然而，第一条规则，要记住审计与性能是成反比例的。建议最低限度记录 lo（login/logout(登录/注销)）,aa(authentication/authorization（认证/授权）),ex(execution（执行）)和 pc(process lifetime（进程存活时间）)。因为在高安全性设备上的审计是很严苛的，要考虑到使用 ahlt 标志，它能够在审计失败时中止系统。

User-level Security

Login banner

除了使用常用的/etc/motd 外，图形登录界面能被设置显示成一个通知。当然这不能决定阻止任何黑客，但是在使用策略上确实起到了一个警告作用，并且在某些位置能够被合法的请求。

```
defaults write /Library/Preferences/com.apple.loginwindow LoginwindowText “lorem ipsum..."
```

Password Hints

在任何的密码提示出现前，有规则的调整密码尝试失败次数是合理的。这能够用来禁用整个的密码提示。

```
defaults write /Library/Preferences/com.apple.loginwindow RetriesUntilHint -integer ###
```

Login/Logout hooks

MacOS 一个小知识但是却非常有用就是 Login/Logout hooks。这些路径的二进制程序可

```
defaults write com.apple.loginwindow LoginHook /path/to/execute
defaults write com.apple.loginwindow LogoutHook /path/to/execute
```

可以使用登录钩子运行一个程序，例如监视用户登录和记录或者实时的警告管理员。同样的，一个注销钩子能够用来确保删除临时文件（例如，使用 srm 清除回收站中所有文件）。

注意，登录/注销钩子是有潜力持续扫描隐藏恶意软件位置，并且应该定期检查非法修改（最好在每一个用户会话中）。

Password Policies

企业的 MacOS 系统由于允许集中认证服务，会在许多情景中用控制器自动同步他们的密码策略。Mac OS X Server App(或者更早的系统，工作组也能配置这样的系统)。

从命令行模式中，powerlicy(8)工具里可以找到设置密码策略的所有方面。

这个工具（在第一章中被提到的）恰好在它的手册中有说明。实际建议策略将会是不同的。

Screen Saver locking（屏幕保护程序锁）

大多数用户离开他们的电脑都没来得及锁屏，并且当过路人下意识的点击一下窗口就能偷取信息和运行命令时，无人值守的会话状态意味着安全风险。因此建议设置屏幕保护选项-通过系统首选项或者 defaults(1)命令：

```
defaults write com.apple.screensaver askForPassword -int 1
defaults write com.apple.screensaver askForPasswordDelay -int 0
```

Disable su

su(1)是没有安全敏感，但比 sudo(8) 起更重要作用，因此应该被禁止。禁用它就像一个 chmod u-s 操作一样简单，但是更久远的建议是在 pam_deny.so 添加一行（如第一章中的“修改 PAM 配置文件”）。

Harden sudo

虽然无参的 sudo(8)比基本的 su(1)要更安全,但是 sudo 的默认配置能够且应该被加固

因此建议以下步骤：

在共同环境中，只有指定的命令能够被 sudo 使用。这些命令应该包括比 shutdown(8)和 reboot(8) 更加安全的命令。在任何时候下，绝不应该所有的 shell 都能被运行，因为这有效的绕过了 sudo 命令作为用户能简单地 sudo bash 或相似命令的限制条件。

Sudo 有一些已知功能在 tty_tickets，它绑定了超级用户许可终端(tty)在那些最终已认证 sudo 命令上。如果没有这个功能，一旦他们之一被认证，两个不同用户会话终端将自动获得超级用户权限。

其他有用的特性是 log_input 和 log_output。这些应该被全局设置或者被设置在一个单命令模式基础上（使用 [NO]LOG_[INPUT/OUTPUT]）。Sudo 命令甚至能够应用到发送电子邮件的成功或失败上。这些命令和其他命令大量文献能够在 sudoers(5)找到。

Periodically check start up and login items

搜寻驻留的恶意软件，因此定期检查用户启动项和登录选项是一个好主意。确切的时间也许不同，但是可以一周一次，或者尽力用钩子相关联登录项。任何新发现的项应该被考虑到审计价值。这包括检查登录/注销钩子甚至是它们自己的工作，因为恶意软件也许会寻求驻留，通过 cron 或者 AT（计划的安排、任务的管理、工作事务）调度它们自身。

Use MDM (or parental controls) to manage user sessions and capabilities

如第六章所描述，MacOS 有一些软件限制机制，通过 mcxlr 二进制程序实施并且与它的内核扩展相关联。软件限制十分强大，它仅仅允许被选上白名单的 app 安装运行或者甚至还原工作站到“kiosk”模式。

精确限制是留给管理员策略的，但是这个机制是被高度推荐的。

Data Protection

Periodically obtain cryptographic snapshots of important files

重要系统文件，例如/etc/hosts（绕过 DNS），/etc/passwd 和其他经常被恶意软件或带有各种各样目的的黑客修改。仅仅依靠文件大小和时间戳是不够的，因为调整文件大小和时间戳十分简单。

加密哈希值功能例如 MD5 和 SHA-1 不能够简单地碰撞。因此对重要系统配置文件运行一个周期检查是一个好主意。当然对于文件认为是不可变的（例如，/bin,/usr/bin 中的各种各样的二进制等等）。重要文件的确切名单有所不同(因为会需要更新操作系统补丁或升级)。重要文件任何可探测的改变应该立即提示一个警报。

Periodically backup user data

用户数据很容易丢失---被意外删除、有针对性破坏或者勒索软件勒索。定期备份数据能够减缓潜在损失。备份脚本能够手动配置，或者使用第三方管理工具。在这些配置手册中，使用 find / -mtime … | xargs tar zcvf 能够运行得很好。

当一个单个、可信的备份服务器能与网络上的机器无密码、使用公钥的 SSH 会话通信，通过网络备份是最好的。

Cloud Saving

MacOS 与 iCoud 结合越来越紧密，这对普通用户通常非常方便，但是在一些情境中会成为一个潜在的数据泄漏点。iCloud 存储需要被禁止，用一下命令很容易做到：

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

Enable hibernation

Pmset 手册描述了 hibernatemode 不同选项尤其是模式 25，它仅仅可以通过命令行可设置 utility.hibernatemode=25 是通过 pmset 唯一可以设置的。系统会存储一份内存映像到持续性存储介质上（磁盘），并且移除内存电源。系统会从磁盘中还原映像。如果你想“hiberna-tion”-慢休眠，慢唤醒和更好的电池寿命，你应该使用这个设置。

Secure Deletion

一个 HFS（或 APFS）中的文件实际上不会被删除-它们的文件系统节点是无链接的，但是数据块直到在低级空间条件下（a low disk space condition，这里实在不清楚是什么条件）才会被清除或回收。强迫安全删除是可能的-通过使用 srm(1)或 rm –p 复写数据块内容。注意到这个方法不推荐用户在 Flash or Fusion Drives 中使用，因为大大地增加了 P/E 循环次数和缩短了存储介质寿命。

Physical Security

设置一个固件密码阻止任何启动配置改变，例如尝试从另一个引导设备引导。这大大地增加了你的 Mac 的安全性。在 Apple 文档知识库文章 HT204455 中有设置固件密码的过程，但必须通过 recovery 文件系统设置。

Find my Mac

许多人熟悉设置”find my i-Device”这样的特色，并且这也同样适用于 Mac。尽管经常对于固定位置的 Mac Pro 和 iMac 无用，但是这对于 macbook 来说是一个福利。不仅仅自动设置固件密码，如果被偷或丢失，它也能够远程锁定和清除 Mac。

FileVault 2

应该启用 FileVault 2（ 全磁盘加密）。这个重要特性在 Mac OS10.7 以上版本可以获得。这个功能可靠、易懂并且高效。尽管当在系统上运行时没有明显效果，但是它使得在设备被破坏或重启时非法授权的个人无法访问。

Remove key during standby

当一台 Mac 进入到待机状态时，FileVault2 密钥仍然在物理内存中明文存储。这使得有些基于硬件攻击类型能通过捕捉和转储 RAM 映像确定密钥。设置 pmset destroyfvkeyonstandby 1 可以从内存中删除密钥，但是当电脑从待机状态脱离时，就需要用户重新登录了。

注意这个设置已知会干涉电脑正常的待机和 power nap。所以这两种设置都应该被禁用（使用 pmset –a [standby/powernap]）0。

Disabling USB, BlueTooth, and other peripherals

软盘已经是遥远过去的遗迹，CD-ROM 同样也是。然而 USB 仍然被广泛地使用，并且是恶意软件一个潜在的入口。部署在高安全性环境下的 MacOS 希望禁用 USB 大容量存储设备。通过从/System/Library/Extensions 移除 IOUSBMassStorageClass.kext 可以做到，记得修改目录以便重建 kernelcache。使用类似的方法能禁用 USB，不过由于 USB 外接键盘而变得不切实际。在 IOBlueToothFamily 类似的方法可以移除掉蓝牙功能。

记住，这些方法尽管可以撤销但是有点极端（通过替代已移除的内核扩展和重建 cache）。只通过在扩展上使用 kextunload（以 root 运行），临时应用它们是可能的，除非它们离开原地。一个更好的方法仍然是对特定设备限制功能。通过第三方内核扩展能做到，它会首要拦截设备通知-大多类似 VMWare Fusion 和其他虚拟化程序抢夺 USB 控制权的方法。这样一个 kext 将会定义一个 IOKitPersonalities key，类似于下述：

```
<key>IOKitPersonalities</key>
	<dict>
		<key>UsbDevice</key>
		<dict>
			<key>CFBundleIdentifier</key>
			<string>....  </string>
			<key>IOClass</key>
			<string>....  </string>
			<key>IOProviderClass</key>
			<string>IOUSBDevice</string>
			<key>idProduct</key>
			<string>*</string>
			<key>idVendor</key>
			<string>*</string>
			<key>bcdDevice</key>
			<string>*</string>
			<key>IOProbeScore</key>
			<integer>9005</integer>
			<key>IOUSBProbeScore</key>
			<integer>4000</integer>
		</dict>
		<key>UsbInterface</key>
		<dict>
			<key>CFBundleIdentifier</key>
			<string>....  </string>
			<key>IOClass</key>
			<string>....  </string>
			<key>IOProviderClass</key>
			<string>IOUSBInterface</string>
			<key>idProduct</key>
			<string>*</string>
			<key>idVendor</key>
			<string>*</string>
			<key>bcdDevice</key>
			<string>*</string>
			<key>bConfigurationValue</key>
			<string>*</string>
			<key>bInterfaceNumber</key>
			<string>*</string>
			<key>IOProbeScore</key>
			<integer>9005</integer>
			<key>IOUSBProbeScore</key>
			<integer>6000</integer>
		</dict>
	</dict>
```

上图显示的字典可以匹配任意 USB 设备，但也可以很容易用来作为已知设备类别的黑名单或白名单。通过简单的忽略硬件设备来创建一个 kext 已经在卷 II 中讨论了。

Enabling SIP (MacOS 11+)

系统完整性保护（SIP）是自从沙盒以来 MacOS 最重要安全机制。虽然它不是万能的，但它通过引入在 root 和 kernel 之间的另一种信任边界，使得操作系统层面的攻击更加困难。

MacOS 10.11 默认开启了 SIP，也没有任何实际理由来关闭它（除了在开发机器上）。csrutil 工具可以有选择性的关闭部分保护，当然这需要根据实际情况来决定。对未签名的 kext 的保护仍然会保留，但应该鼓励内核扩展的开发者只测试签名的扩展。

Enforcing code signing

就像第五章讨论的，MacOS 的代码签名可以与*OS 一样严格，但需要配置 sysctl(8)变量。可以在下面的输出中看到推荐的 sysctl 变量：

```
vm.cs_force_kill: 1    # Kill process if invalidated
vm.cs_force_hard: 1    # Fail operation if invalidated
vm.cs_all_vnodes: 1    # Apply on all Vnodes
vm.cs_enforcement: 1   # Globally apply code signing enforcement
```

Block-Block/Flock-Flock/etc-etc

许多第三方工具，无论是开源的还是商业的，都尝试在系统中提供实时的应用（进程）监控。一些是被动的，收集信息，另一些则深入到检查特殊的系统 API，例如阻止系统调用或 mach traps。这与反病毒软件或反蠕虫软件类似（虽然大多数都很 reactive，而不是 proactive）。推荐一般会避开提供这个或那个工具，尽管通常这些工具都提到过。

Sandboxing

就像第八章讨论的，沙盒是个很强大的容器机制，内置于 Apple 所有的操作系统。使用 sandbox-exec(1)工具，你可以强制让未知或未受信的二进制文件运行在容器环境中。你还可以使用沙盒的跟踪功能得到这个二进制文件执行的每一个操作（系统调用层次）。注意，对二进制文件应用一个受限制的 profile，通常会影响到它的功能，因为一般写代码时很少考虑限制。

Virtualization

计算能力在过去一代人里发生了迅猛的增长，对大多数用户来说，CPU 的能力已经超过了计算的需求。虚拟化可以通过一些方法无伤害的增强安全。

- **分离下载和附件：**虚拟机提供了容器环境，即使恶意程序胡作非为，也只能造成较小的破坏。可以在这个环境中预测程序以及什么时候下载，这个很容易忘记。

- **截图和克隆：**允许快速创建，并搭建已知、安全的配置。很容易通过点击一个按钮回到可信的配置。

Application Layer Firewall

从“安全和隐私”中启用防火墙，会启动 /usr/libexec/ApplicationFirewall/socketfilterfw (通过 launchd(8)的 com.apple.alf.agent.plist LaunchDaemon 属性配置)。launchd 会把 daemon 的标准错误重定向，并输出到/var/log/alf.log 文件，但是默认情况下日志会被记录到 /var/log/appfirewall.log 文件中。

Apple 在一篇知识库文章（https://support.apple.com/en-us/HT201642）中描述了应用防火墙的基础信息，没有暴露它是如何实现的。本书的卷 II 有更多关于这个强大机制的详细信息。此外，配置信息存储在/Library/Preferences/com.apple.alf.plist （可以通过 defaults(1)命令访问)。

重要的 key 如下：

allowsignedenabled: 自动放过签名的应用。

applications: 应用 ID 数组，通常为空。

exceptions: 字典的数组，每项是一个异常。应用通过路径标示，异常也有一个整型的 state。

explicitauths: 字典的数组，每项包含一个应用（bundle identifier）。用于解释器或执行环境，例如：Python, Ruby, a2p, Java, php, nc, and ksh 。

firewall: 关键服务的字典，每个 key 包含进程的名称和整数状态。

firewallunload: 表示防火墙是否被卸载的整数。必须为 0。

globalstate: 表示状态的整数，必须为 2 。

loggingenabled: 表示 alf.log 是否被使用了的整数。必须为 1 。

loggingoption: 整数表示日志标识。可能为 0 。

stealthenabled: 整数，标识主机是否响应 ICMP 协议（例如：ping）。应该为 1 。

pf

作为应用层防火墙的补充，macOS 也提供了 pf 作为数据包层面的过滤器。这是个内核设施，但可以被用户模式控制。可以通过/dev/pf[m] 字符设备，也可以通过 pfctrl(8) 命令，以及 /etc 目录中的文件 pf.conf(5)。这个文件是主要的配置文件，在启动时加载规则集，也可能会重定向或包含其他文件（通常是/etc/pf.anchors）。pf 是从 BSD 借鉴的，从某些层面上类似 Linux 的 netfilter 机制（iptables 防火墙的基础机制）。

卷 II 中更详细的解释了 pf 的操作。在数据包层面防火墙，比应用层防火墙能在 OSI 中应用层“看到”更多的信息。当然这带来了优势（例如 packet-level dropping, NAT, masquerading 等），也带来了缺点（不能访问 reassemble packets ）。

Eliminate all unnecessary services

应该禁用所有不必要的服务。例如，带来不安全因素的 Remote Apple Events 和 Internet Sharing 必须禁用。特别是要禁用 Apple Event，它会对安全带来严重的妥协。

Secure all necessary services

一些服务是必须的，例如备份或网络访问，可以考虑找到对应的更安全的替代品。macOS 不再允许不加密的 telnet 以及 ssh，ssh 的安全性可以通过 PKI 体系增强，补充或者替代密码（建议两者都使用）。同样，FTP 可以使用 SFTP 替代，任何不安全的过时的协议（POP，ICMP）都可以改为使用 SSL，或通过 SSH 或 SSL 传输。

Consider decoys for unused services

网络攻击者，尤其是自动化的蠕虫，尝试通过扫描网络识别远程主机，或者利用已知漏洞来传播。配置一个蜜罐（可以是 nc -l 这种形式）可以捕获这种攻击，可以立即给出攻击的报警。

Little Snitch/Big Brother/lsock

第三方工具，例如 little snitch 和 big brother ，可以在后台监控网络活动，同时还有个强大的操作界面。lscok 工具（可以从这本书的合作网站下载源码）和 Apple 自家的 nettop(1) 可以通过命令行实现几乎相似的功能。nettop 和 lsock 可以在终端下发挥所有功能，后者还可用于过滤器，例如配合 nc。

使用网络监控工具可以高效的监控到进和出的网络连接,可以监控到恶意程序与远程服务器通信的开放端口。这些工具的输出可以和防火墙或路由器的日志配合。

Recompiling the kernel

XNU 内核（macOS 使用）仍然是开源的，在 OpenSource.Apple.com ，我们可以自己编译出与发布版本一样的内核。可以修改编译配置，添加调试功能，也可以增强安全性。

安全的内核可以通过设置 SECURE_KERNEL #define 为 1（iOS 也是）。这对内核有以下影响：

核心转储不再被创建：bsd/kern/kern_exec.c's do_coredump 设置为 0，系统全局禁用核心转储。不安全的内核只禁用了 s[u/g]id 二进制文件。另外,对应的 sysctl(8)s (from bsd/kern/kern_sysctl.c) 也禁用了。

代码签名显著加固安全： cs_enforcement_enable 和 cs_library_val_enable 变量 (in bsd/kern/kern_cs.c) 变为 1, 并且定义为 const, 因此不能通过 sysctl 修改它们。另外, cs_enforcement_disable 启动参数不再生效。 相似的， 37xx 之前的内核的 vm.cs_validation sysctl(8) (from bsd/kern/ubc_subr.c 被移除了。

强制启用用户模式的 ASLR ：POSIX_SPAWN_DISABLE_ASLR 选项也不生效（在这个文件中 bsd/kern/kern_exec.c）

NX sysctl(8)被禁止：data 段将被标记为不可执行，并且不能修改。

vm.allow_[data/stack]_exec 都禁止(在 bsd/vm/vm_unix.c, MacOS 12 中已经移除)

kern.secure_kernel 和 kern.securelevel sysctl(8)s 被设置：前者设置为 1 (true) ，后者设置为 "security level". 这在文件 bsd/sys/system.h 中 如下定义:

```
/*
 * The `securelevel' variable controls the security level of the system.
 * It can only be decreased by process 1 (/sbin/init).
 *
 * Security levels are as follows:
 *   -1 permannently insecure mode - always run system in level 0 mode.
 *    0 insecure mode - immutable and append-only flags make be turned off.
 *      All devices may be read or written subject to permission modes.
 *    1 secure mode - immutable and append-only flags may not be changed;
 *      raw disks of mounted filesystems, /dev/mem, and /dev/kmem are
 *      read-only.
 *    2 highly secure mode - same as (1) plus raw disks are always
 *      read-only whether mounted or not. This level precludes tampering
 *      with filesystems by unmounting them, but also inhibits running
 *      newfs while the system is secured.
 *
 * In normal operation, the system runs in level 0 mode while single user
 * and in level 1 mode while multiuser. If level 2 mode is desired while
 * running multiuser, it can be set in the multiuser startup script
 * (/etc/rc.local) using sysctl(1). If it is desired to run the system
 * in level 0 mode while multiuser, initialize the variable securelevel
 * in /sys/kern/kern_sysctl.c to -1. Note that it is NOT initialized to
 * zero as that would allow the vmunix binary to be patched to -1.
 * Without initialization, securelevel loads in the BSS area which only
 * comes into existence when the kernel is loaded and hence cannot be
 * patched by a stalking hacker.
 */
```

大胆一点，我们可以修改源码来改变 XNU 的功能。例如，macOS 可以通过禁用 kexts 在用户模式的加载（通过 kext_request ： host priv #425）来改为使用*OS-like 的 kexts。已知的安全内核扩展可以优先链接入内核缓存。所有运行时链接都被禁用。记住，修改过的内核源码是通过 Apple 的代码中创建的分支，当新操作系统版本发布时能否及时跟进，这对我们是个挑战。

XNU 的开源代码可以根据 README 的说明编译。容易忽视的是一些依赖。卷 II 中讨论了这个内容，以下 package 是需要获取的：

Cxxfilt: 实际名字是 C++filt, 但是 + 在 DOS 文件名中是非法字符.

Dtrace: 当前版本: x.x. CTFMerge 需要.

Kext-tools: 当前版本:

bootstrap_cmds: Current version: xxx. Required for relpath and other commands

可以在用户主目录下 ~/Library/Preferences/com.apple.Terminal.plist 文件中找到 SecureKeyboardEntry 的布尔配置项，为 true.

Demote setuid binaries

经典的 UN*X setuid(2) 模型对于安全来说是个灾难。像输出中的 suid，macOS 在最近的版本已经逐渐减少二进制程序的数量，但是即使是那些也不一定每天都需要。二进制程序可以保留，但是应该被降级（通过 chmod u-s）。如十二章所说，在 dyly 中的漏洞就是由于在 setuid 下执行代码产生了即时 root。

降级 setuid 二进制程序会关闭本地提权的一个重要载体，但是也许或者不影响可用性。例如，如果 at(1)程序不在使用，/usr/bin/at 和/usr/bin/atq 可以无副作用移除。然而一些二进制程序尤其是 sudo(1)和 security_authtrampoline(8)不能在不影响整个功能情况下被降级。对于前者，如果程序不使用了，这不是一个问题。但是对于后者，二进制程序在系统内部中被使用，并且不能被降级。

Remove unnecessary binaries

用户普遍不会深入到使用终端和 shell 环境。即使当他们这样做了，他们的命令应该适当的被限制。在管理员判断下，主要的二进制程序-甚至整个 app-能够被移除，也应该小心的行使权利。移除/禁用 Terminal.app 对于安全有很大的帮助。

Consider patching/recompiling specific binaries

在系统中有一些二进制程序，即使它们本身无害，但也许对于 hack 过程也有用。

例如，细想一个攻击者以某种方式设法获得了 shell 访问。

一个多才多艺的黑客可能会在这些限制找到一种方法（但是像 chmod(1)所示的情况，因为它不可避免的是一个系统调用。即便如此建议还是在脚本注入和执行的自动攻击的情况下被证明是有用的，因为脚本是用来下载一个二进制程序并且依靠 chmod(1)来执行它）