

Practical Mac OS X Insecurity

Security Concepts, Problems, and Exploits on Your Mac

Angelo Laub
al@rechenknecht.net

January 1, 2005

1 Introduction

While rumors have it that Mac OS X is extremely secure due to its open-source Darwin core and the elaborate Unix security model, little is known about practical problems that hide under its hood. The fact that so far no serious worms or viruses exist for the Mac might give users a false sense of security. The system has its vulnerabilities and it is only a matter of time until they will be exploited. The openness of the Objective-C language and the Mach kernel allows the user to perform modifications deep in the system even at runtime. This paper intends to give an overview of the worst current security holes and possible fixes.

2 Security Concepts in Mac OS X

In Mac OS X, many of the more obvious security problems known from other operating systems such as MS Windows are not present. The most important among them is the fact, that the user is never logged in as root. Although the normal user is usually in the admin group, system critical access normally will not be granted without further authentication. Malware that wants to install itself system-wide therefore has to rely on some sort of privilege escalation exploit, or trick the user into entering his or her password. There is a danger though, that Mac users are too unsuspecting about entering their password. Most Mac users feel pretty safe due to their system having a reputation of being invulnerable. Otherwise damage is only inflicted on the current user without permanently damaging the system.

In the Mac OS X default installation, no unnecessary services are listening, so none of them can be exploited from remote. The real danger consists in local vulnerabilities, of which, unfortunately, Mac OS X currently has quite a few, as this paper shows. Most of them stem from the fact that Apple tries to combine the Unix security model with easy and convenient usability and closed source. The private **Admin Framework** that Apple uses for carrying out administration tasks is the most prominent and interesting among the closed-source components. For user authentication and privilege delegation, Apple has integrated the closed-source programs **Security Server** and **Security Agent** which rely on both on the private **Admin Framework** and the **Security Framework**. Since they internally rely on SUID root programs to carry out administration tasks, it is possible that one discovers more security holes as one tries to reverse-engineer them.

Currently there is no widely known way to create link viruses for Mach-O, Mac OS X's binary format. This, of course, can change as soon as the Macintosh platform gains more and more importance.

3 Vulnerabilities

We will now have a closer look at some of the more serious local vulnerabilities in Mac OS X. None of these are currently fixed in the latest **Security Update 2004-12-02** for Mac OS 10.3.6. Please understand that this list is by no means comprehensive. However, typical vulnerabilities illustrating fundamental security flaws in various domains were picked.

3.1 System Preferences

The application **System Preferences** is one example of Apple's problematic closed-source security components. In a default OS X installation, any user in the admin group can do pretty much without further authentication. He or she can start and stop services, change user interface settings and create new user accounts. While this does not look like a major problem so far, this ability can also be used to create users and add them to the admin group without authentication. With this, an attacker can immediately get root within five seconds. This is something the author discovered together with Jan Manuel Tosses. The author reported this to Apple together with a simple exploit [4] written in **AppleScript** in October. As it is **AppleScript** wrapped in shell, the exploit even works from remote via ssh and as a normal user who is not even in the admin group as long as a user of the admin group is logged in. So far, Apple has not fixed the issue.

A fix would be to enable the **Require password to unlock each secure system preference** radio button in the **Security** pane of the **System Preferences**.

3.2 Bad Installers and Wrong Permissions

Apple has specified the directory `/Library/StartupItems` for application specific startup items. The executables in this directory will be executed with root permissions during startup. Unfortunately, the directory does not exist by default and must be created by the installer. Many installers will set the wrong permissions, so that this directory is user- or even world-writable. Now every user can execute arbitrary code by putting it in `/Library/StartupItems` and restarting the machine. In order to counter-balance permissions problems, Apple provides **Disk Utility**, which lets you repair disk permissions. Unfortunately, it will not repair the permissions of `/Library/StartupItems`, which is a serious bug. To fix this temporarily, I recommend putting a

```
chown -R root:wheel /Library/StartupItems
chmod -R og-w /Library/StartupItems
```

into `/etc/daily` or `/etc/rc.local`.

3.3 Mach Injection

The Mach kernel API allows to inject code into other running processes and remotely creating a new thread to execute it. Several libraries exist for this purpose ([5], [6] and [7]).

With them it is possible to extend Objective-C classes in running applications with categories or replace them by using the posing language feature. Together with the Objective-C Runtime Library it is also possible to selectively replace one method with another in arbitrary classes, which is called `MethodSwizzling` [2]. Again, this can all be done in runtime, even with closed source system applications like e.g. the Finder. While this gives programmers a very powerful tool to patch system and other closed source applications, an attacker could modify system applications at runtime in a malicious way. The whole range of security implications is yet to be explored.

3.4 Open Firmware

An Open Firmware password can be set in order to protect your Mac from booting from a different than the default device. As a positive side-effect, it also disables Firewire DMA, so that the Firewire vulnerability described in the next section cannot be exploited anymore.

Of course, there is a way around it. The Open Firmware password can be cleared by changing the amount of physical RAM and then resetting the Parameter RAM ("zapping the PRAM") three times by pressing Option-Apple-P-R. There is also a comfortable way to read out the password from the shell:

```
sudo nvram security-password
```

Advice: There have been reports of corrupted passwords when installing firmware updates while the security-password is active. Make sure you disable it beforehand.

3.5 Firewire

In Mac OS X, Firewire devices have access to the whole virtual memory, since Firewire DMA is activated by default. One could imagine a "rogue iPod", that, once connected, reads out all passwords and steals data or crashes the ScreenSaver. As a fix, one can disable Firewire DMA by setting an Open Firmware password.

3.6 Single User Mode

By pressing Command-S during startup, one is immediately presented a root shell. When having hardware access, this is a pretty sure and fast way to get root.

One can force password authentication by changing the login style from `secure` to `insecure` in `/etc/ttys`. Since the `DirectoryServices` are not up yet when entering single-user mode, authentication will fail in any case, rendering the single-user mode unusable. In order to repair it one can generate a password with:

```
openssl passwd -salt $salt $password
```

and insert it in `/etc/master.passwd` next to `root:`.

3.7 Disguised Executables

It is possible to disguise executables as any other file type wished. If one tries to rename a `.app` file to `.mp3` in the Finder, it will still get the `.mp3.app` suffix. This security measure can be easily circumvented by renaming the file in the shell. One can now go ahead and

replace the application's symbol with the appropriate symbol for an mp3 file, and it will not be distinguishable from a real mp3 file by the average user.

This fact can easily be used to create a trojan horse. Consider an AppleScript .app package containing the actual .mp3 file as a resource. When double clicked upon, the AppleScript will be started, opening the actual mp3 with iTunes and then executing the malicious code. Any of the other described privilege escalation vulnerabilities described here can then be used by the script to install malware into the system. Imagine, then the malware would send the disguised installer via mail to all addresses in the user's Address Book. This would be a self-replicating trojan written in AppleScript.

3.8 Personal Filesharing Denial of Service

Personal File Sharing is very useful to transfer files between Macs. Therefore, many people leave it enabled in the **System Preferences**. Unfortunately, a guest account is enabled with it, that lets anyone from the LAN and the internet connect without authentication. The guest user cannot do much, but he can write to the Drop Box. There is no limitation in the amount of data written to the guest Drop Box and there is no way to disable the guest user in the System Preferences. An attacker could write data to the Drop Box unnoticed until the victim's hard disk is full. In the worst case, this could crash the victim's Mac, since swapping fails when the Startup Disk is full.

The guest account can be disabled by editing
`/Library/Preferences/com.apple.AppleFileServer.plist` and changing

```
<key>guestAccess</key>  
<true/>
```

to

```
<key>guestAccess</key>  
<false/>
```

and then restarting the Apple File Server by typing:

```
sudo killall -HUP AppleFileServer
```

The guest account should be disabled by default and one should be able to activate it in System Preferences.

3.9 Clear Text Passwords in Swap File

Apple's Security Framework does not use `mlock()` or equivalent to prevent passwords to be swapped to disk. Therefore it is likely, that user passwords and other passwords from the Keychain will be written to the swap file in clear text. You can verify this on your own Mac by typing:

```
sudo strings /var/vm/swapfile0 |grep -A 4 -i longname
```

One may argue, that this is not such a big deal, because one needs root privileges to be able to read the passwords. Yet, there are two scenarios in which this can become a major problem. First, when an attacker has root or physical access as described in section 3.5. Second, it makes

OS X unusable for multiple users, because it is trivial for every admin user to read the other users' passwords and therefore to decrypt the FileVault.

In the future version Tiger, one will be able to activate swap encryption in the System Preferences. A way to enable encrypted swap in Panther is described in [8].

4 What to do once you got root

There exist at least two rootkits for Mac OS X, *osxrk* [3] and *WeaponX* [1]. The latter was written because the author of *WeaponX* felt that *osxrk* did not meet the standards of the Mac platform. *WeaponX* hides itself from *kextstat*, *netstat*, *utmp* and *wtmp*, and the source code is available as a nice Xcode project. I do not count *Opener* as a rootkit because it is merely a shell script without the possibility to hide itself.

5 Conclusion

Mac OS X is not a Unix system where you want other people have shell accounts. All the described vulnerabilities can be fixed by the user if he wants to have a more secure system, but it is the default setting that counts. It is also very likely that more vulnerabilities will be discovered in the near future.

In order to further evaluate the security situation in Mac OS X, it will be necessary to take a closer look at Apple's **Admin Framework** by taking advantage of the open nature of the Objective-C language and the Mach kernel. The security implications of Mach Injection also need to be explored further. One could imagine malware that works entirely with user privileges that patches the system in an evil way.

Despite all these vulnerabilities, Mac OS X is still much more secure than Windows, because in Windows you are practically always root. We can only hope that Apple can manage this balancing act between ease-of-use, closed-source, and UNIX security in the long run and that OS X eventually deserves to be called secure again.

References

- [1] Neil Archibald. *WeaponX Rootkit for Mac OS X*. <http://neil.slampt.net/>.
- [2] CocoaDev. *MethodSwizzling*. <http://www.cocoa-dev.com/index.pl?MethodSwizzling>.
- [3] g@pper. *osxrk Rootkit for Mac OS X*. <http://packetstormsecurity.org.pk/UNIX/penetration/rootkits/osxrk-0.2.1.tbz>.
- [4] Angelo Laub. *System Preferences Root Exploit*. <http://annulator.de/3sec2rewt.sh>.
- [5] Unsanity LLC. *Application Enhancer (APE)*. <http://www.unsanity.com/haxies/ape/>.
- [6] The Zaius Project. *Zaius free MacOS X patching*. <http://zaius.sourceforge.net/>.
- [7] Jonathan Rentzsch. *mach_inject*. http://rentzsch.com/mach_inject.
- [8] Andreas Schwarz. *Encrypted Swap on Mac OS X 10.3 (Panther)*. <http://andreas-s.net/osx-encrypted-swap.html>.