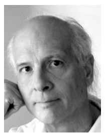
Musings

RIK FARROW



Rik is the editor of *;login:*.
rik@usenix.org

Once a year, the local Mac users group asks me to talk to them about Mac OS X security. And every year, I search for something exciting and recent, something that I can use to motivate Mac users into being concerned about their security. In April of 2012, I had lots of fodder for a good talk.

Mac users tend to be complacent about their security. They know that Windows users are the predominant target of malware. During the LEET workshop, Tudor Dumitraş of Symantec Research Labs shared information from their Worldwide Intelligence Network Environment, a large dataset collected from millions of Windows hosts. Tudor had little to say about Macs; the amount of malware collected was small, as were the number of Mac users with antivirus and intrusion protection installed.

But does this mean that Macs are inherently more secure? Or is it simply a case of there being many fewer targets, and thus the much greater focus on Windows systems?

The Numbers

Four years ago, Adam O'Donnell wrote an article for *IEEE Security and Privacy* [1] in which he outlined a game theory approach for why Macs have not been targeted by malware writers. Game theory uses simple mathematical equations to calculate the best strategies for players, in this case, malware writers and Mac users, and the associated payoffs. In 2008, O'Donnell calculated that the Macs would have to make up 16.6% of the total number of client systems before they would become an attractive target for malware writers. With Mac adoption at about 11%, Mac users should have been safe for a while longer [2].

The arrival of the Flashback trojan, which infected around 700,000 Macs over a short period of time in April 2012, was certainly a rude awakening [3]. Flashback (which Kaspersky calls Flashfake) used a vulnerability in Apple's version of Java (other versions of Java had been patched in January 2012) to install a trojan downloader. The attack used social engineering to get people to run the Java applet, by posing as a method for updating Adobe Flash player—itself a notorious source of infection in 2011. If the user starts the applet, which could be part of a Web page or an email, the applet installs the trojan, executes it, and the trojan contacts its C&C (command and control) server—in other words, exactly like a Windows bot.

Apple responded quickly (for Apple) with a patch to its Java implementation, Within a week, Apple produced two more patches, actually disabling the execution of Java besides updating their JVM. Their first patch would also remove Flashback, which I thought was a nice touch.

But what happened to Apple's "more secure than Windows" veil? Newer Windows versions (since Vista) have included security features designed to prevent exploits from succeeding (NX, DEP, and ASLR). If you read my summary of Tudor's LEET presentation (in this issue) or watch his presentation online, you will learn that these defensive measures have done little to stem the flood of malware. Just as O'Donnell implies, the platforms with the greatest numbers also attract the most viruses and attacks.

A Closer Look

So why was Flashback so successful? I too had hopes that UNIX-based operating systems, such as Mac OS X and Linux, would be less vulnerable to attacks like this. But this turned out to be the usual self-deception, where one has faith in something without there being anything but an illusion of security.

Flashback relies on the Java bug to escape the JVM's sandbox. It then downloads and installs the executable and arranges for it to execute every time the user logs in. As the user can write to his own directories, adding a file to ~/Library/StartupItems, or to ~/Library/LaunchAgents/ [4] is something malware can easily do once a user has begun executing the malware's code. Flashback and SabPub [5], another Mac trojan, one used for spying rather than acting in a botnet, both add entries in the StartupItems directory. Launchctl can easily work like cron, and arrange for execution of a program on user login, routinely (every *n* seconds) after the user has logged in, or at set times. Neither approach requires administrator privilege.

As for locking these methods down, I don't think you can prevent them. ~/Library/StartupItems/ is a directory (folder if you like) owned by the user, and changing the owner of that directory to root won't stop an attacker working as the owner of the parent directory. I didn't see a way to configure launchd [6] that prevents ordinary users from taking advantage of this useful (and yet dangerous) tool. At least in Linux, you can control which users get to use `crontab -e` with /etc/cron.allow or /etc/cron.deny. But Linux users will soon be getting something like launchd in the form of upstart [7], a replacement for the init daemon, which will not only facilitate faster booting but also eventually replace cron. We will have to see if upstart can be locked down so only root or sudoers can use it.

Mac and *nix users still have one advantage over Windows users when it comes to client security. If Windows users opt to include Administrator privileges with their user account, this privilege can be used by exploits as well. For example, a Windows user with Administrator privilege can starting sniffing network interfaces without being required to authenticate.

Mac and Linux users, on the other hand, get prompted to enter their password, or the root password in Linux, when updating or installing software, before they can perform privileged tasks. Sniffing the network, for example, running tcpdump requires that the user work as root. While requiring that the user provide a password, a form of authentication, is nice, I am sure that many users do this without considering or understanding, the implications of doing so.

As a proof of concept, consider the dialog box shown in Figure 1. I began my Mac security talk in 2011 with this illustration, which I left on the screen for 15 minutes, while I waited to begin. I then asked members of the audience if they noticed anything unusual about the dialog box. I asked the people who I knew did Mac consulting if it alarmed them. Eventually they noticed the poor English, but not that "Cancel," customarily found in such dialog boxes, had been replaced with "Abort." This example was taken from actual Mac malware.

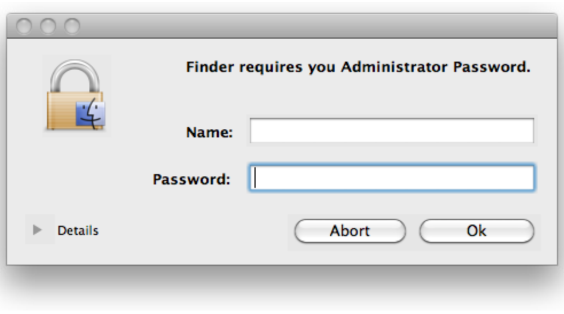


Figure 1: A dialog box found on Mac OS X systems that is requesting an administrator's password

The SabPub [5] Mac malware I mentioned earlier is a backdoor that is being used in directed attacks. That is, Mac users are singled out, targeted with email that uses social engineering to get them to execute and install the malware, often just by clicking on a link or opening a Microsoft Office Word or Excel file [8].

While I was working on my Mac user group presentation [9], I asked many security friends what they thought about Mac security, knowing that most of them used Macs (laptops, desktops, and/or iPhones). Two people told me that they knew that there is an active, but underground, market in Mac-specific malware, but neither would go on the record. I know these people are not just pulling my leg. So if you are using a Mac, secure in the knowledge that you are safe from targeted attacks, I suggest you adjust your belief system.

Defense

I also wondered what this group of security geeks thought about using antivirus for Macs. Most do not use AV on their Macs. One salient reason is that the market for Mac AV is so small that there is little focus on it by AV vendors. Another person, in a position to know, said that an AV product developed for Linux had sold only six copies before it was discontinued. I am not saying this to suggest that Linux or Macs are immune to malware, just that current AV is not going to be much help.

Currently, iOS has stronger security than Mac OS X. Apple has published a 20-page document [10] describing iOS security features that sound very good to me. They are not perfect, but actually enforcing the use of ASLR on built-in and purchased apps, the use of a sandbox, the use of signed "entitlements" for access to resources, all appear to be evidence of good security design.

Some versions of Linux include SELinux. I've been using SELinux for years and have taught classes about it for the past three years. I recently had to check if my new CentOS 6.2 install had it enabled, as I have had no problems with SELinux at all. SELinux helps sandbox services as well as applications that run with privileges, like set-user-ID programs. With the sandbox command, you can run Web browsers in isolated environments that disappear when you exit. What you cannot easily do is write policy that you can trust to be complete for your own applications. Writing policy is just too complex.

Speaking of Web browsers, it is important to keep in mind that any extension you install in your browser becomes part of the browser, its "chrome." While extensions have some limitations, the software you have added has access to everything you download, just as the default install of the browser does. It's this feature that makes it possible for NoScript to block JavaScript, and Ghostery to tell you about Web bugs, ad networks, and other "invisible" stuff added to Web pages and used to track you. It also makes it possible for malware that gets installed as an extension to modify what you see and to collect the information you enter when using your browser. The article by Franzi Roesner in this issue explains how their extension, which controls third-party tracking, takes advantage of rewriting the pages you view.

Your Web browser, and the ability of your mailtool to parse the same set of protocols, is what has made client systems so vulnerable to attacks. If we were still using plain old text, the Mac attacks, and most Windows attacks, would not be possible. Not that I expect anyone to give up the advances we've seen in pretty displays over the past 17 years, but it has made us vulnerable. Go back to text-only days, and the most comparable attack would be the inclusion of vi or emacs macros at the beginning of a text file. Today, it is common for image protocol parsers, such as JPEG and PNG, to include exploitable vulnerabilities. I suggest reading the summary of Ulfar Erlingsson's LEET talk, or the associated paper [11], if you want to learn how Google is working at reducing the risk associated with the many protocols we use today.

The Lineup

Franzi Roesner and her co-authors start us off with research into just what your browser can do for (or to) you. Roesner has researched the various ways your browsing habits can be tracked, using an assortment of cute technology, such as single-pixel iFrames. While many browsers have an option to block third-party trackers, these measures are ineffective against tracking by social media sites. Ever wondered how Facebook "Like" or Google "+1" buttons know who you are? Roesner et al. don't just explain how these trackers work, they also present an extension that fits into Firefox and Chrome that neuters these buttons—until you want to use them. Roesner's paper was presented at NSDI '12, and a summary of her presentation appears in this issue as well.

Keith Winstein and Hari Balakrishnan write about Mosh, a mobile shell application that is actually designed for mobility. SSH uses TCP, and your connections die if your source IP address changes. Also, TCP handles bufferbloat and high latency connections poorly. Mosh leverages SSH for its ability to set up and authenticate a connection, and then takes over, using AES encryption for your data and UDP for transmission. Mosh maintains virtual screens at both the client and the server, and provides local echo to clients using a conservative approach, but one that makes high latency connections much more pleasant to use. Keith presented this paper at USENIX Annual Tech in Boston, and that summary will appear in the October 2012 issue of *;login:*.

Mohammed Mannan and Paul van Oorschot examine the use of password tools for mobile devices. They evaluate their own tool, ObPwd, and show how it can provide strong passwords while simplifying the entering of passwords for mobile users.

ObPwd also works for desktop users, allowing users to share the same mechanism and secret token (an image or file) on both types of systems.

Rick Forno takes us in a different direction. Rick has been a long-time security professional, a Washington D.C. observer, and is now the director of the cybersecurity program at the University of Maryland, Baltimore Campus. Rick presents the history of failed security policies that have led us to this moment in US history and demonstrates how these proposals have remained essentially the same since 1996. He then makes three suggestions of his own for improving the stance of US security policy.

Dan Geer heads in yet another direction, with a different look at policy. Dan lets me read the text of speeches he makes, and when I read one of his latest, I contacted him immediately and asked him to write for this issue. What caught my attention this time was Dan's concept of the role of the Internet rejectionist, and how important this is for creating an infrastructure that is robust enough to withstand and quickly recover from failures or large-scale attacks using the Internet.

Matei Zaharia and co-authors, winners of the NSDI '12 Best Paper award, present Spark, an interface for distributed data modeling. Spark borrows some concepts from DryadLINQ, but is open source, and already in use outside the lab where it was developed. Spark presents a rich language for interaction with large datasets via Hadoop and HDFS, but also introduces the concept of *resilient distributed datasets* (RDDs). While using Spark, you can persist the result of filtering or manipulating large datasets in memory and speed up applications on Hadoop by as much as 40 times. RDDs are stored in memory and not replicated—if they are lost, a graph of transformations is used to recover the data. You can find out more about Spark on its Web site, http://www.spark-project.org, and by reading the paper presented at NSDI '12 and the summary in this issue.

Doug Hughes presents the second part of the story he began in the June 2012 issue. In that issue, Doug hinted at a near disaster involving two racks of drives storing 640 TB that started failing. Doug explains how the combinations of failures managed to overcome a level of redundancy they had felt was surely sufficient. Imagine having the loss of vast amounts of data hanging over you, not for days, but for longer than a week. Doug also provides lessons learned, not just in debugging but also in the design of redundant storage.

David Blank-Edelman almost departs from teaching us about Perl to explain XPath. XPath provides a common syntax for addressing portions of XML documents. XML documents can be viewed as trees, with a single root and many children, and XPath allows you to efficiently specify and extract elements of those trees. In the end, David shows us how to use XPath with Perl libraries.

Dave Josephsen continues his series on tools, written by Mathias Kettner, that work with Nagios. In this column, Dave waxes enthusiastic over Livestatus, a tool that presents Nagio's structures via a UNIX socket, using a simple API. Dave explains how Livestatus is superior to the other commonly used methods of extracting status information from Nagios, as well as demonstrating the use of Livestatus.

Dave Beazley explains how the import statement works in Python. A lot is happening behind the scenes when you import library modules and third-party extensions, and Dave explains how Python searches for these libraries. He also explains how you can vary the searchpath, how the searchpath varies for different versions of Python, and what different types of files may be found using import.

Robert Ferrell is excited about yet-to-be written, but oh so necessary, apps. You might think that with hundreds of thousands of apps already written, there could only be tiny niches left, but Robert's imagination takes us to places few dare to go.

We have a nice batch of book reviews in this issue. Elizabeth Zwicky starts with a book on analyzing big data, covers two books on Agile programming, then a book on building free software communities. She finishes up this set of five reviews with a book on domain modeling. Peter Gutmann tackles a legal tome that covers electronic signatures. The author has left few stones unturned: while not exactly beginning with the Stone Age, he does go back to the Magna Carta. Sam Stover writes an enthusiastic review of Metasploit, a book on the penetration testing framework. Metasploit itself is vast, but Sam tells us that this book carefully relates both how Metasploit fits into pen testing and also how Metasploit is used. Finally, Mark Lamourine reviews three books by Allen Downey, designed to teach students (and the self-taught) how to think like scientists. Downey uses examples in Python to teach problem solving, use of statistics, and computation modeling.

Finally, we have two reports, for NSDI '12 and LEET '12. There were many interesting papers at NSDI this year, and LEET combined academic and commercial research for a very interesting workshop, one that has been one of my personal favorites.

In 2006, I decided that I needed to go out into the world and point out just how badly the various security mechanisms that had been created, starting with MULTICS, were working. The point of my "Security Is Broken" talk [12] was not to beat a dead horse, but to encourage researchers and developers to try some different approaches. I ended my talk by suggesting that capabilities seemed like a fruitful direction, even though it had been explored in the past.

Today, we have capability-like features in iOS [10], as well as Robert Watson's Capsicum [13], now part of FreeBSD 9, and perhaps soon to appear in other OSes. The difference between Capsicum and approaches like SELinux is that the security policy of an application appears within the source of the application, instead of being stored and managed separately. Using Capsicum forces the developer to consider first what resources are needed, then to arrange for those capabilities to be available to an application running with reduced privileges, the set of allowed capabilities.

This still doesn't deal with the basic problem we have today in many of our applications. Web browsers operate with our own user privileges these days, and that allows reading and writing anything that we can. Even a browser or email tool that is effectively sandboxed can still be used to pass a Word document to an exploitable version of Office, and to silently do other things we don't want. We are not used to working in the type of compartmentalized systems that will work best with a capabilities-based system. Similar compartmentalized workstations were tried in the late '80s and were never easy to use.

Consider the LAMP model: Apache, MySQL, and PHP all running over Linux. That's an enormous software stack, all based on having a flexible Web server that can be remotely changed. Wordpress, as an awful example, recommends that you allow Apache write permission to all of its files, for purposes of administration (installing modules and updates). You can download PHP shell from SourceForge, something I imagine gets installed in lots of writeable Web server directories.

As I prepared for my Mac users group talk, I could tell from my email exchanges with the organizers just how disturbed they were. Someone had written software that had *silently* infected 700,000 of Macs—perhaps their very own Macs. The security paradigm we have inherited from MULTICS, that users need to be isolated from one another, does not fit the security problems we face today. Today, via our Web browsers, we execute code and interpret complex files that are provided to us by third parties that are usually unknown to us, in our own user context. So it is no surprise to me that malware still gets installed, when the very systems we use does that by design.

References

[1] Adam O'Donnell, "When Malware Attacks (Anything but Windows)," *IEEE Security and Privacy*, 2008: http://www.securitymetrics.org/content/attach/Metricon3.0/j3attAO.pdf.

[2] Andy Greenberg, "Cybercrime Game Theory: Why Apple's Malware Grace Period Ended Early," *Forbes*, April 20, 2012: http://www.forbes.com/sites/andygreenberg/2012/04/20/cybercrime-game-theory-why-apples-malware-grace-period-ended-early/.

[3] Igor Soumenkov, "Flashfake Mac OS X Botnet Confirmed," Kaspersky Lab, April 6, 2012: http://www.securelist.com/en/blog/208193441/Flashfake_Mac_OS_X_botnet_confirmed.

[4] David Lanier, "Cron and launchctl on Mac OS X 10.5 Leopard": http://www.davidlanier.com/story/cron-and-launchctl-on-mac-os-x-105-leopard.

[5] Costin Raiu, "SabPub Mac OS X Backdoor: Java Exploits, Targeted Attacks and Possible APT Link": http://www.securelist.com/en/blog/208193467/SabPub_Mac_OS_X_Backdoor_Java_Exploits_Targeted_Attacks_and_Possible_APT_link.

[6] Mac OS X Developer Library, launchd.plist(5): http://developer.apple.com/library/mac/#documentation/Darwin/Reference/ManPages/man5/launchd.plist.5.html#//apple_ref/doc/man/5/launchd.plist.

[7] "What Is Upstart?" *Linux*, Jan. 6, 2011: http://www.linux-magazine.com/Online/News/What-is-Upstart; http://upstart.ubuntu.com/.

[8] Dennis Fisher, "MacControl Trojan Being Used in Targeted Attacks Against OS X Users," *ThreatPost*, March 28, 2012: http://threatpost.com/en_us/blogs/maccontrol-trojan-being-used-targeted-attacks-against-os-x-users-032812.

[9] Rik Farrow, "Malware: No One Is Safe (Mac malware presentation for Mac users): http://www.rikfarrow.com/malware2012.pdf.

[10] *iOS Security*, May 2012: http://images.apple.com/ipad/business/docs/iOS_Security_May12.pdf.

[11] Mike Samuel and Ulfar Erlingsson, "Parse to Prevent Pwnage": https://www.usenix.org/system/files/conference/leet12/samuel.pdf.

[12] Rik Farrow, "Security Is Broken": http://video.google.com/videoplay?docid=1762847950860111011.

[13] Robert Watson et al., "Capsicum, Practical Capabilities for UNIX": http://www.usenix.org/event/sec10/tech/full_papers/Watson.pdf.