INTERCEPTING PASSWORDS TO ESCALATE PRIVILEGES ON OS X

MATHEW WEEKS
@SCRIPTJUNKIE

INTERCEPTING PASSWORDS TO ESCALATE PRIVILEGES ON OS X

A few weeks ago, a lot of attention was paid to Dropbox for "hacking" macs. Dropbox asked for your admin password when it was installed, then used that root access to enable privileges later even if you try to disable them. Despite the internet's indignation and Dropbox's impoliteness, Dropbox wasn't exploiting any vulnerability or breaking any security barrier; of course, once you've given your admin password to something, it can do whatever it wants with it.

If you were surprised by that, just wait, because there is a real security issue; Dropbox was being nice, on OS X, software doesn't need to ask you for your password, it can simply take it the next time you make an administrative change:

(video: https://youtu.be/Fssjt2KXuxk)

This is a follow up to the previous post, The Security Pretend Game. That post described how all major desktop OS's now run user programs with limited privileges by default, and have users escalate privileges when required from within that desktop to run certain programs. While this might be enough to prevent you from accidentally breaking something important, these "security" measures do nothing to stop hackers; any of the supposedly unprivileged programs running in the same logon session have sufficient privileges to detect, intercept, and alter or hijack the privilege escalation process.

Last time I intercepted passwords from command line tools, this time I did it from the standard OS X GUI. The source code is at https://github.com/scriptjunkie/kcap.

OS X has attempted to prevent this by blocking unprivileged process injection/debugging of the program which runs the password prompt, preventing unprivileged code from detecting the keystrokes via keystroke logging functions, and preventing programs from drawing on top of or over the password prompt window. These restrictions are easy to circumvent, however.

This program simply uses screen captures and programmatically generated key and mouse events to locally and graphically man-in-the-middle an OS X password prompt to escalate privileges.

OS X makes this interception easy because, in contrast with Windows' UAC prompt, unprivileged processes are allowed to interact with and spoof input to the password request window. Here's how it works:

1. It polls the screen to detect a system authentication prompt. Once the user triggers one,

2. It takes a screenshot of the prompt.

3. It closes the real authentication prompt.

4. It creates a cloned version and displays that.

5. Then when the user enters a password it saves the password,

6. Re-opens the original password prompt,

7. Moves it offscreen,

8. And types in the password to the original password prompt.

9. Then it moves the orignal prompt back to its original position

10. And hits enter, sending the typed password on to the original prompt.

This may sound like a lot, but it only results in a brief flash in the UI that is hard to see.

The peanut gallery might point out that there is an empty dock item that is created when you run this, and it's possible to see a few pixels in the corner briefly show up when you run the attack. This is only because I was lazy and have never programmed an OS X GUI program before, so I just spent a couple hours and did it in Java. If you write a native program you can make both of those go away.

I tested it on a Macbook Air at the default resolution 1440x900 against most of the password prompts from the system preferences application. It probably will get the pixel calculations wrong on other systems so you'll need to test and possibly modify some of the offsets if you want to run it elsewhere.

If Apple "fixes" this attack vector by denying unprivileged programs the right to interact with that window, it doesn't matter; unprivileged programs could simply modify the shortcut that starts the system settings application and have it start a cloned version instead that does the same thing except also recording your password.

Common wisdom would have you believe when you type in your password you are only granting root privileges to one command at one point in time. In reality, you are granting root privileges to any hacker who has ever run any code in any process at any previous time in your account and decided they wanted escalated privileges.

Having users escalate privileges from an unprivileged desktop will never be a real or effective security barrier. Instead, use separate accounts for admin tasks and day-to-day tasks. Don't browse the internet or check email when logged in to your admin account and don't enter your admin password when logged into your day-to-day account.