Don't forget your hardware

Hard Hacks

Now that your networks are secure and you've convinced your users to secure their passwords and software, it's time to turn your attention to your hardware, to keep it from being attacked. *By Kurt Seifried*

This month, it's back to the physical world – specifically, hacking systems via hardware. In security, sys admins usually focus on remote attacks (because anyone on the Internet can hit you) and local, shell-level attacks (because you can't trust all users – or trust them to secure their passwords and so forth properly). Rarely do administrators pay much attention to physical attacks because, face it, someone with physical access and time can bypass virtually all security measures or simply modify the system so that, for example, after it boots up, it records your passwords to decrypt the hard drive. However, a number of physical attacks can take only seconds – meaning anyone who's had even a minimal amount of access to the system because, say, you turned around to talk to someone or deal with someone who bumped into you (I know, it sounds a bit like a spy movie).

Auto-Mounting Filesystems

One common method of attack on all peripheral connection standards that support storage devices (USB, eSATA, FireWire, Thunderbolt, etc.) is through the filesystem. By default, most versions of Linux now mount filesystems automatically on any devices plugged in, which is handy – you can stick a USB thumb drive in, it appears on your desktop, and things just work. The downside is that, with automount enabled, an attacker can use any supported filesystem (and there are many) to attack the system, as reflected by the number of filesystem-related attacks that have occurred on Linux systems. To limit the number of supported filesystems, you can *blacklist* various kernel modules. To get a list of modules, simply list the contents of `/lib/modules/$version/kernel/fs`, then simply add them to `/etc/modprobe.conf` or `/etc/modprobe.d/blacklist.conf` (or whatever file your distribution uses; it should be similar). Of course, you need to leave whatever filesystems you actually use (e.g., ext4) enabled.

FireWire and Thunderbolt

These interfaces are old, but still good. The basic issue is that Firefox and Thunderbolt are designed to be extremely fast and flexible (e.g., they do quite a bit more than just transfer simple data). Thunderbolt, for example, can be used to send a video signal to drive monitors, and FireWire devices can talk directly to other FireWire devices. In theory, you could plug an external FireWire hard drive into a FireWire-capable camera and just dump the data straight to it without using a computer. Thunderbolt and FireWire both provide DMA (Direct Memory Access) to devices, which is great for speed and not so great for security because you can write to virtually any location in memory, and also to any of the PCI devices that use memory mapping. In other words, a malicious Thunderbolt [1] or FireWire device can write directly to memory, bypassing the operating system entirely. An attacker could, for

KURT SEIFRIED

Kurt Seifried is an Information Security Consultant specializing in Linux and networks since 1996. He often wonders how it is that technology works on a large scale but often fails on a small scale.

example, overwrite userspace application memory or even overwrite kernel memory, making it fairly trivial to do anything they want to the system in a pretty much undetectable manner.

The good news is that most of us aren't actually using FireWire, and Thunderbolt is still in its infancy, so my advice would be to disable FireWire and Thunderbolt in the BIOS if you can. You might also consider disabling the ports physically; however, this will of course damage the system, so I would not recommend it unless you have no other choice. Note that several commercial programs use FireWire DMA attacks to recover passwords from computers with FireWire (e.g., on Mac OS X, it can gobble up all the passwords in the keychain storage in memory), so these attacks are used quite a bit. In other words, if you're logged in and your encryption keys, authentication keys, and so on are in use (e.g., stored in memory), you are vulnerable.

USB Hacking

If you have a general purpose computer, it's safe to say it has a USB port (be it a desktop, laptop, netbook, tablet, phone, camera – you name it). The state of DMA with respect to USB on Linux is a little more complicated. USB drivers can support DMA, but they don't have to. However, USB 3.0 DMA will become more common because it helps adds significantly to performance. Another interesting and unique avenue of attack for USB is the HID (Human Interface Device – typically things like mice, keyboards, and game controllers). The real magic of these devices is that they can send control signals to the system, the most interesting device being a keyboard, which would allow you to run applications, for example. To make things especially interesting, it turns out that you can fit a USB hub and HID device into a very small space [2]-[5], like the space in a computer mouse or a large thumb drive. Then, you could send it to someone as a prize or drop it near the office of your target and wait for them to plug it in. Once plugged in, the device can send keystrokes and mouse commands to the machine, opening a command prompt and establishing a reverse bind shell so you can get remote access to the system, for example. Or, if combined with a storage device, it can open that device and install any software stored inside. Additionally, these devices can be programmed to either wait or send the attack at intervals. So, if you plug a device into a computer that is off or suspended, it will still work once the user comes back (and almost nobody checks the back of their computer to see if anything was plugged in while they were at lunch).

Securing USB

Unlike FireWire and Thunderbolt, however, chances are you can't live without USB (not if you want to use a keyboard and mouse, anyway). So, what to do? All USB devices have a vendor and product ID that you can use to identify them. Why not just whitelist all the USB devices you have by product and vendor ID so when an attacker plugs theirs in, it doesn't work? The problem is that the vendor and product ID can be spoofed easily. Sadly, the device simply sends an arbitrary number when queried and, as such, provides no real security. For once, I have some good news: You can configure your system easily to disable any new USB devices from being plugged in and working by simply editing the `authorized_default` setting for each USB controller:

```
for i in /sys/bus/usb/devices/usb*/➥
   authorized_default; do echo 0 > ➥
   $i; done
```

Another bit of good news: On all Linux systems I tested, it appears that, even when you disable new USB devices from working, then plug one in and re-enable new USB devices, the one you just plugged in won't work. So, even if an attacker plugs a device in and leaves it – waiting for you to re-enable USB so you can plug a thumb drive in, for example – you should be safe (but please test this before relying on it!). Basically, you can create a program or script to run that shell command, disable all USB devices and only enable them quickly when you need to plug a new device in.

Bluetooth

Don't forget Bluetooth. Not only does it let you plug in external peripherals, it allows you to do so wirelessly. The designers weren't completely ignorant of security; Bluetooth devices now use a reasonably secure cryptographic protocol. However, older devices do not, so if you're using older devices, especially something like a keyboard, it wouldn't be that hard for an attacker to modify a Bluetooth device to send the same device serial number and other information that your device does. The easiest way to deal with this is to disable Bluetooth, or, if you have to use it, ensure that it is configured to require authorization for a device to connect and that all your devices are new enough to support secure connections.

Conclusion

Physical security still matters, especially when a physical attack might last less than a few seconds (plug the USB device in, wait for it to be recognized and do its work, then unplug it). Because you don't want to go around with a large tube of glue physically disabling all the ports in your machines, you're going to have to ensure that your systems, both at the BIOS level and operating system level, are configured securely. ■■■

INFO

[1] Hacking Macs through the Thunderbolt interface: *http://www.breaknenter.org/2012/02/adventures-with-daisy-in-thunderbolt-dma-land-hacking-macs-through-the-thunderbolt-interface/*

[2] Don't stick that in there – HID: *https://www.securepla.net/dont-stick-that-in-there-hid-human-interface-device/*

[3] Hacking the HID: From Zero to Pwned in 10 seconds: *http://hak5.org/episodes/hak5-1006*

[4] USB-Rubber-Ducky/Firmware: *https://github.com/hak5darren/USB-Rubber-Ducky/tree/master/Firmware*

[5] Weaponizing the Teensy: *http://initiate6.blogspot.ca/2011/09/in-progress-this-post-will-be-updated.html*