Reversing and exploiting an Apple firmware update

K. Chen
Georgia Institute of Technology

Abstract

The security posture of a computer can be adversely affected by poorly-designed devices on its USB bus. Many modern embedded devices permit firmware to be upgraded in the field and the use of low-cost microcontrollers in these devices can make it difficult to perform the mathematical operations needed to verify a cryptographic signature. The security of many of these upgrade mechanisms is very much in question. For a concrete example, we describe how to tamper with a firmware upgrade to the Apple Aluminum Keyboard. We describe how an attacker can subvert an off-the-shelf keyboard by embedding into the firmware malicious code which allows a rootkit to survive a clean re-installation of the host operating system.

1 Introduction

In 2005, the Defense Science Board of the Department of Defense expressed concerns about the migration of microelectronics foundries from the United States to foreign countries and its impact on the security of microchips and microelectronic components delivered to the government and military and used in critical infrastructure [3]. If an adversary is able to gain access to a microelectronic component during the design phase, then a clandestine modification will corrupt every unit manufactured and the confidentiality, integrity or availability of any system using such a component can be compromised. Moreover, given the complexity of modern systems, such a modification can be deeply embedded into a system and difficult to detect and attribute.

In [16], King et al. implemented a general-purpose malicious processor by modifying a VHDL implementation of the Leon3 open source SPARC processor. In order for unprivileged software to access privileged memory locations, the MMU was modified so that when a particular magic sequence was observed on the data bus, all protection checking was disabled. By reserving lines in the instruction cache and data cache and using processor debugging hardware, a shadow mode mechanism was created to allow malicious code to be bootstrapped via a UDP packet. These malicious modifications were implemented using only a 0.13% increase in logic gates and allowed the implementation of malicious services such as root privilege escalation, a login backdoor and the theft of passwords from individuals using the system.

Vertical disaggregation in the semiconductor industry has continued since the 1990's and fabless IC production is now common. The significantly lower costs of capital and lower costs of operation offshore and the infeasibility of having low-volume manufacturing within the United States only increases the possibility of this kind of threat being realized.

Threats from malicious hardware are not limited to microelectronic components and recently, consumers have suffered security compromises simply through the purchase and use of off-the-shelf consumer electronics. In 2006, Apple shipped a number of iPod music/video players pre-installed with a Windows virus called RavMonE.exe [4] and TomTom shipped a number of its GO 910 GPS navigation devices [6] infected with Windows malware. In a promotion in Japan in 2006, McDonald's distributed 10,000 MP3 players pre-installed with a password-stealing trojan [5].

In 2007, Seagate shipped a number of its Maxtor Personal Storage 3200 hard drives with a trojan called Virus.Win32.AutoRun.ah which was capable of disabling anti-virus software and was designed to steal login credentials to World of Warcraft and a number of online Chinese games [7]. During the 2007 holiday season, BestBuy sold a number of digital picture frames preinstalled with a Windows virus under its house brand Insignia [8]. In 2008, Hewlett-Packard shipped a number of infected USB keys with its ProLiant servers [9] and Asus shipped infected Eee Box mini-computers [10].

Although it is believed in all of these cases that com-

puters used in manufacturing or testing were inadvertently infected and that malware was able to "hitch" a ride, these episodes illustrate the feasibility of an attacker subverting the supply chain.

Recently, counterfeit Cisco networking equipment originating from China has been discovered in the United States. The counterfeit equipment was often found due to duplicate MAC addresses, duplicate serial numbers and having a higher failure rate than genuine equipment. In 2008, a PowerPoint presentation of an Office of Management and Budget briefing given by the FBI regarding the problem of counterfeit Cisco equipment was inadvertently leaked onto the internet [18]. The presentation described a number of rings located in the United States that were actively selling counterfeit Cisco gear on the auction site eBay and to Cisco Gold/Silver partners through whom equipment ultimately ended up at numerous government agencies and defense contractors. The presentation asks rhetorically if the motive behind the counterfeit gear is for profit, or if it is state-sponsored with the intention to "cause immediate or premature system failure during usage," "gain access to otherwise secure systems," or "weaken cryptographic systems."

Due to intense pressures to reduce time-to-market, a number of consumer hardware products can have their firmware field-upgraded, and often firmware upgrades are released to fix bugs and problems that are discovered after the product has shipped. In fact, there is an official USB device class specification for doing firmware upgrades over USB [2] although we are not certain how widespread its use is. The problem is that an attacker may also decide to upgrade the firmware on a device with their own code for malicious purposes by taking advantage of weaknesses in the upgrade mechanism.

In some cases, the owner or operator of the embedded system makes deliberate efforts to tamper with the system without the sanction of the manufacturer. For example, there is an entire industry which sells aftermarket performance products for various brands of automobiles that increase performance by changing the air-fuel mixture, removing the top speed governor, increasing rev-limit, etc. often at the expense of fuel economy and increased emissions. These software modifications are applied to a vehicle through its OBD II connector or by removing the ECU from the vehicle and sending it in for bench-programming. Other examples include firmware for the Linksys WRT54G, jailbreaking/unlocking software for the Apple iPhone and third-party firmware for lower-end digital cameras which unlock RAW shooting and other advanced features typically found only on higher-end models. Television pirates have reprogrammed the firmware of cable and satellite receivers and attacked smartcards for the purpose of signal theft.

2 Prior Work

Researchers have recently become interested in attacking insecure software update and installation mechanisms. Package managers for Linux and BSD operating systems are typically run as superuser in order to modify system software. They differ in whether cryptographic signatures are on the root metadata, are embedded within the packages themselves or on detached package metadata. In [15], Cappos et al. investigated popular package managers such as Yum, Apt, YaST, ports, etc. and discovered weaknesses in every one they looked at. If an adversary has control of a mirror or is performing a man-in-the-middle attack via poisoning ARP or DNS or spoofing DHCP, a variety of attacks can be performed. Clients can be continuously served outdated repository metadata in order to prevent security updates or served outdated, but legitimately signed packages with known vulnerabilities ("metadata replay"). Yum can be subverted by rewriting package metadata to cause additional packages to be installed or by returning a `repomd.xml` file of unlimited size to fill up the disk of a client and cause denial of service.

An attacker can of course setup their own mirror for popular Linux distributions such as Ubuntu, Debian, Fedora, CentOS and openSUSE and perform these kinds of attacks. Over the years, numerous sites serving open-source software have been compromised with the 2003 compromise of the GNU project's ftp server being among the most serious [1]. Just recently in August 2008, an unknown attacker compromised a number of servers at Red Hat, including a machine used to sign Fedora packages. Red Hat claimed however that the package signing key itself was not compromised, but did issue a new signing key [11].

For Windows and Mac OS X operating systems, in [12] Amato developed an open-source toolkit called EvilGrade for the exploitation of popular software products which perform insecure updates. An attacker performing a man-in-the-middle attack can then exploit a vulnerable application by injecting a fake update. Applications which fail to verify updates and have modules in the toolkit include iTunes, Winamp, WinZip and the Sun Java plugin. The toolkit even includes a module for performing malicious updates to machines running the Mac OS X operating system.

However, the idea of tampering with software updates is by no means new. In the past, before the widespread penetration of the internet, patches were delivered on magnetic or cartridge tape and shipped through the mail or sent by courier. K. Mitnick has said that long ago he successfully stole the source code to a DECNET/E protocol sniffer from a small company called Polar Systems using a fake update. Mitnick took a legitimate up-

date tape for the VMS operating system from Digital Equipment Corporation, added code to install a backdoor into the login program, repackaged the box and shrink-wrapped it with a counterfeit shipping label. He then obtained a UPS uniform, delivered the fake update tape to the firm in person and waited for the update to be installed [17].

In high-security environments such as government installations and defense contractors, we have often heard stories of the USB ports of computers being disabled by filling them in with epoxy glue. On some computers with poorly-designed USB devices, we do not believe that this is sufficient to protect against rogue USB devices. For example, we performed most of the work in this paper on a 4th-generation Apple Macbook Pro laptop computer (with model identifier `MacbookPro4,1`) and even without plugging any devices into its external USB ports, the computer already has four devices on its USB bus: the bluetooth device, the internal iSight webcam located above the LCD screen, the integrated keyboard/trackpad and the infrared receiver.

In this paper, we discuss a practical attack that can be performed today which blends the threats of malicious hardware and malicious software updates.

3 Keyboards

As the primary point of data entry in a computer, keyboards and typewriters before them have long been of interest to attackers. In the 1980's, numerous news sources reported that typewriters in the U.S. Embassy in Moscow were discovered to have devices planted inside of them which sent out signals encoding keystroke information using the power cord at television frequencies. Apparently, other countries with embassies in Moscow discovered similar devices in the 1970's.

In 1999, the FBI obtained warrants to enter the office of Nicodemo S. Scarfo and Frank Paolercio in New Jersey and encountered an encrypted file. In order to decrypt this file, the FBI later returned to the office and covertly installed a keystroke logger on Scarfo's computer in order to capture his PGP encryption passphrase. The FBI obtained the passphrase and is alleged to have obtained evidence of illegal gambling and loansharking. In 2001, the DEA conducted an investigation of an MDMA manufacturing operation. They received authorization to enter the Escondido, CA office of Mark Forrester and Dennis Alba to install a keylogger onto their computer in order to obtain passphrases for PGP and their accounts on the encrypted mail service Hushmail.com.

Free and commercial software keystroke loggers are widely available. logKext is a free and open source keystroke-logging kernel extension for Mac OS X. Given the ease of using `SetWindowsHookEx()` in the Microsoft Win32 API, there are literally thousands of keystroke loggers for Microsoft Windows.

Hardware keystroke loggers for both PS/2 and USB keyboards are also widely available commercially and typically are devices that sit inline between the terminating plug of the keyboard and the port on the host computer. Parasite devices that sit inside the keyboard are also commercially available, but require more effort to install. Keystroke-logging mini-PCI cards that can be installed into laptops are also commercially available. In general, it is difficult to extract the captured data and so in [19], Blaze et al. built a hardware logger that perturbed the inter-keystroke delay and used the delays as a covert channel and for interactive protocols such as SSH and VNC, they were successful in extracting keystroke data without having to physically access the logger. Researchers have even reported success at the recovery of keystrokes from recordings of acoustic emanations from somebody typing at a keyboard [20].

4 The Apple Aluminum Keyboard

In August 2007, Apple introduced new wired and wireless keyboards to accompany its redesigned iMac desktop computer products. The keyboards have low-profile keys and come in a thin, aluminum enclosure. The wired keyboard terminates in a USB A male connector and connects to a desktop computer through an available USB port. The wireless keyboard uses Bluetooth 2.0. The wired keyboard which has a model number of A1243 and an Apple part number of MB110LL/A. It is a compound USB device in that the device is actually a hub with a keyboard plugged into it and an available USB port on each side of the keyboard. The keyboard we examined has a vendor id of `0x05ac` and a product id of `0x0220`, and its integrated hub has a product id of `0x1006`. This keyboard is widely deployed and until March 2009, came standard with all new iMac and Mac Pro desktop computers from Apple, although a customer had the option to pay additional money to get the wireless keyboard in lieu of the wired keyboard. The keyboard can also be purchased separately from a desktop computer for $49.00 USD directly from Apple and from other retailers.

In March 2009, Apple introduced another wired keyboard with model number A1242 and an Apple part number of MB869LL/A. The layout of the keys on this keyboard is identical to the Bluetooth keyboard and does not have a numeric keypad. This keyboard has a vendor id of `0x05ac` and a product id of `0x021d`, and its integrated hub has a product id of `0x1005`. Both of the wired keyboards can be used without difficulty on almost any modern machine via the USB human interface device class.

In this paper, we examine the wired keyboard with the

numeric keypad introduced in 2007. We were unable to find sales figures for the three models of keyboards currently available for sale by Apple, but we believe that it is the most widely deployed keyboard out of the three due to its cost and length of time on the market.

By examining a disassembled keyboard, we learned that internally there is a Cypress CY7C63923 microcontroller in a 48-pin SSOP package surface mounted to the circuit board of the keyboard. The CY7C63923 is an 8-bit microcontroller with a Harvard architecture, 256 bytes of RAM and 8 kilobytes of flash. The chip belongs to Cypress Semiconductor's enCoRe II family of chips designed primarily for ease of use in low-speed USB devices. The chip can support one low-speed USB device address with three endpoints: the required control endpoint and two additional endpoints. Upon plugging the keyboard into a computer, we learned that two IN interrupt endpoints are configured with addresses of `0x81` and `0x82`. They both have a `bInterval` value of 10. Endpoint `0x81` has a `wMaxPacketSize` of 8 and endpoint `0x82` has a `wMaxPacketSize` of 1.

On the circuit board adjacent to the microcontroller, there is a Microchip 25LC040A serial EEPROM in an 8-pin SOIC package. It is a 4-kilobit EEPROM with an SPI interface. Although adjacent to the microcontroller, by following circuit traces on the board we determined that the EEPROM is actually connected to the Cypress CY7C65630 USB 2.0 hub controller in a 56-pin QFN package on the board. A high-level schematic of the keyboard is in Figure 1. We traced the circuit paths for the LED on the keyboard underneath the CAPS LOCK key. The anode is connected to V_{DD} and the cathode is connected through resister R7 to pin 8 of the microcontroller, which is P2.7. This means that the LED is active-low on this pin.

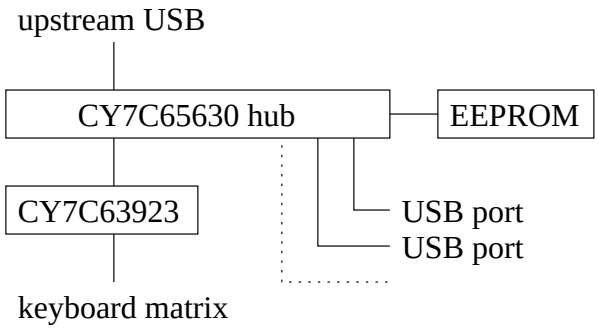

Figure 1: A high-level schematic of the keyboard.

5 Apple's Firmware Update

Version 1.0 of the Aluminum Keyboard Update from Apple was released on April 8, 2008 and is available online [13]. Apple released the update to address complaints from users about keys repeating unexpectedly while typing and other issues. The resulting download file is named `AlKybdFirmwareUpdate.dmg`. From the disk image file, a flat package file named `AlKybdFirmwareUpdate.pkg` can be obtained. When running this installer package, an application called "Aluminum Keyboard Firmware Update" is created in the `/Applications/Utilities` directory. In Mac OS X, applications are actually stored in a directory structure with ".app" appended to the name of the application. In the directory `Contents/Resources` within the application's directory, two files called `HIDFirmwareUpdaterTool` and `kbd_0x0069_0x0220.irrxfw` are of interest.

5.1 Reversing the Firmware Update

The magic number for the application `Aluminum Keyboard Firmware Update` is `0xCAFEBABE` which indicates that it is a fat/universal binary. For convenience, we primarily examined the Intel x86 portion of the binary in this Cocoa application. Examining the `CustomerKB.nib` file from the English localization, the NSButton push button in the lower right-hand corner of the user interface had a target outlet set to the `MyMainController` class and its action was set to `doUpdate:`.

The `applicationDidFinishLaunching:` delegate method starting at address `0x00004fe7` runs after the application has launched and been initialized, but prior to the first event. The method calls a number of routines. It calls a routine which consults the file `SystemVersion.plist` in `/System/Library/CoreServices/` to determine whether the running operating system version is at least 10.5.2. It finds the keyboard to update by calling the routine starting at `0x000035e0` which uses the I/O Kit library to search for devices on the USB bus with a vendor ID of `0x0220` and product IDs of `0x222`, `0x221`, `0x220`, and `0x228`, in that order. However, we examined a number of Apple Aluminum keyboards "in the wild" and only observed keyboards having a product id of `0x220`.

The application checks a number of properties of the keyboard and checks the validity of the firmware image file `kbd_0x0069_0x0220.irrxfw` in the bundle. The firmware validity checking routine is called `CRC32:` and is the 75 byte routine starting at `0x00003005`. Despite the name, this routine does not do CRC32 at all and in fact, it simply just adds up the bytes of the firmware image file and the application verifies that the sum is `0x252ed7`.

An `otool` disassembly of the first 64 bytes of

Filename	Size	SHA-1
AlKybdFirmwareUpdate.dmg	1,568,432	8c914be94e31a1f2543bd590d7239aebc1ebb0c0
AlKybdFirmwareUpdate.pkg	1,483,229	7e1e75a4d937f6ba44f97a7bfc72e3a04fc9a1be
HIDFirmwareUpdaterTool	76,480	3d564d6cb3bd73121876a2f9a0b9b85ca032a3fb
kbd_0x0069_0x0220.irrxfw	18,253	cf2b7ac6d4575b8f57ba5562ab1d94ff337736f4

Figure 2: Files of interest from Apple, Inc.

```
00004df4   pushl   %ebp
00004df5   movl    %esp,%ebp
00004df7   pushl   %ebx
00004df8   subl    $0x24,%esp
00004dfb   movl    0x08(%ebp),%ebx
00004dfe   movl    0x000080f0,%eax
00004e03   movl    %ebx,(%esp)
00004e06   movl    %eax,0x04(%esp)
00004e0a   calll   0x000090e0
00004e0f   testb   %al,%al
00004e11   jne     0x00004e2f
00004e13   movl    $0x00000015,0x10(%ebp)
00004e1a   movl    0x00008040,%eax
00004e1f   movl    %ebx,0x08(%ebp)
00004e22   movl    %eax,0x0c(%ebp)
00004e25   addl    $0x24,%esp
00004e28   popl    %ebx
00004e29   leave
00004e2a   jmpl    0x000090e0
00004e2f   movl    0x000080f4,%eax
```

Figure 3: First 64 bytes of `doUpdate:`

```
00004c7a   movl    0x08(%ebp),%eax
00004c7d   cmpl    $0x69,0x50(%eax)
00004c81   jne     0x00004cb3
00004c83   movl    0x00008040,%eax
00004c88   movl    0x08(%ebp),%edx
00004c8b   movl    $0x00000011,0x08(%esp)
00004c93   movl    %eax,0x04(%esp)
00004c97   movl    %edx,(%esp)
00004c9a   calll   0x000090e0
00004c9f   movl    0x00008044,%eax
00004ca4   movl    %eax,0x04(%esp)
00004ca8   movl    0x08(%ebp),%eax
00004cab   movl    %eax,(%esp)
00004cae   calll   0x000090e0
00004cb3   movl    0x08(%ebp),%edx
00004cb6   cmpl    $0x69,0x50(%edx)
00004cba   jbel    0x00004d56
```

Figure 4: Part of the version checking code

`doUpdate:` is shown in Figure 3. When the user presses the update push button, the instruction at `0x00004e0a` causes a 239 byte routine located from `0x00003850` to `0x0000393e` to be called[1] which checks whether the machine doing the update is plugged into a wall outlet.[2] Requiring that the machine doing the update be plugged into a wall outlet is understandable since losing power during a firmware upgrade can result in a damaged keyboard, but the reader may disable this check by changing the `jne` at `0x00004e11` to `jmp`, i.e. changing the `0x75` to `0xEB`.

If the reader's keyboard has already had the firmware update applied to it, the update program will display a dialog box saying that the firmware is already up-to-date and refuse to apply the update. A disassembly of some of the code which performs the check is shown in Figure 4. The version checking can be bypassed by making the first conditional short jump unconditional, i.e. changing `0x75` at `0x00004c81` to `0xEB`, making the second conditional near jump unconditional, i.e. changing `0x0f8696000000` at `0x00004cba` to `0xe99700000090`, and then removing the conditional jump at `0x00004820`, i.e. changing `0x740e` to `0x9090`.

5.2 Obfuscation

If the application is satisfied that a keyboard that needs to be updated is attached and the user presses the update button, then it invokes the `HIDFirmwareUpdaterTool` with the arguments `-parse kbd_0x0069_0x0220.irrxfw` to first check that the firmware image can be parsed and then invokes it with the arguments `-progress -pid 0x220 kbd_0x0069_0x0220.irrxfw`. The second invocation does the heavy lifting of actually updating the firmware of the keyboard. The firmware image file `kbd_0x0069_0x0220.irrxfw` is "encrypted" and the core of the decryption routine is displayed in Figure 5. Let A denote the following 83 byte sequence:

```
31 1c ef 62 df a7 43 23 78 92 22 6a
38 12 14 a4 65 02 2b 00 9c 00 57 5e
10 85 50 73 d0 b1 17 2b 49 ac 49 c4
33 21 b4 48 23 8c 27 98 12 34 80 00
48 ff b4 8f 04 2e 24 2d 92 c7 82 e2
a6 a5 20 20 98 11 84 26 b7 cc 28 f3
e6 98 38 23 dc ba 28 44 42 39 44
```

```
00004086   movzbl  %al,%edx
00004089   incl    %ecx
0000408a   movzbl  0x00006020(%edx),%eax
00004091   notl    %eax
00004093   xorb    (%esi),%al
00004095   xorb    %al,0xffffff55(%ebp,%edx)
0000409c   cmpb    %cl,%bl
0000409e   movl    %ecx,%eax
000040a0   ja      0x00004086
```

Figure 5: Decryption code

and let $B = B_0 B_1 \cdots B_{52}$ denote the following 53 byte sequence:

```
12 14 a4 65 02 2b 00 9c 00 57 5e 10
85 50 73 d0 b1 17 2b 49 ac 49 c4 33
21 b4 48 23 8c 27 98 12 34 80 00 48
ff b4 8f 04 2e 24 2d 92 c7 82 e2 a6
a5 20 20 98 11
```

The decryption routine reads the firmware file in 83 byte chunks with the ith chunk XOR-ed with the 1's complement of A and then each byte XOR-ed with $B_{i+16 \bmod 53}$ to produce the "plaintext." So the first 83 bytes of `kbd_0x0069_0x0220.irrxfw` are XOR-ed with the complement of A and then each byte is XOR-ed with `0x17`. The next 83 bytes are XOR-ed with the complement of A and then each byte is XOR-ed with `0x2b`, and so forth. The sum of each byte in the plaintext is then computed and verified to be `0x1057f8`.

5.3 Bypassing the obfuscation

We did not make an attempt to completely understand the algorithm used to obfuscate the firmware image, as it turns out that the tool `HIDFirmwareUpdaterTool` sends "cleartext" over the USB bus to the keyboard's bootloader. The unobfuscated firmware file can be easily obtained from memory.

```
$ gdb -q HIDFirmwareUpdaterTool
(gdb) b *0x4abc
Breakpoint 1 at 0x4abc
(gdb) r -progress -pid 0x220
      kbd_0x0069_0x0220.irrxfw
Breakpoint 1, 0x00004abc in ?? ()
(gdb) dump binary memory dump.bin
      0x61ec 0x89ec
```

The block size in the microcontroller's flash memory appears to be 64 bytes and by examining the file `dump.bin` 35 bytes at a time, the pattern becomes readily apparent. The first 8820 bytes of `dump.bin` are what is relevant. The firmware image is sent over the USB bus 32 bytes at a time and for every grouping of 35 bytes in `dump.bin`, the first 3 bytes indicate where the following 32 bytes are written into flash. The first 2 bytes encode the block number and the third byte encodes whether the 32 bytes are in the top half or the bottom half of the block.

5.4 I/O Kit API

In Mac OS X, the I/O Kit API is a collection of frameworks for device driver development and communication with hardware. `HIDFirmwareUpdaterTool` accesses the keyboard over the USB bus using services from I/O Kit. The tool creates a device-matching dictionary to find the Apple Aluminum keyboard in the I/O Registry, which is a memory-resident, tree data structure[3] that maintains the configuration of devices in the system. When hardware is added or removed from the system, the I/O Registry is automatically updated to reflect the change in hardware configuration. The I/O Registry on a Mac OS X system can be examined using the developer tool "I/O Registry Explorer" or `ioreg` from the command line.

In order to first communicate with the I/O Kit, it is necessary to obtain the I/O Kit master port, a Mach port. `HIDFirmwareUpdaterTool` uses the `IOMasterPort()` function and passes the resulting port to functions that require a port argument, but in recent versions of Mac OS X, the constant `kIOMasterPortDefault` defined in `IOKitLib.h` can instead be passed. At the end, the port is released using `mach_port_deallocate`.

Then, `IOServiceMatching()` with an argument of "IOUSBDevice" is used to create a device-matching dictionary. This is a broad search of the I/O Registry for all USB devices. The search is narrowed by taking the resulting dictionary and adding rules requiring that the vendor ID match `0x05ac` and the product ID match `0x0220` and passing the resulting dictionary to `IOServiceGetMatchingService()` to obtain the registered IOService object. This function is called in lieu of the API call which returns an iterator over the set of matching devices. The program expects at most one matching device and simply accepts the first matching device on the USB bus. In the unlikely event that a computer has multiple Apple Aluminum keyboards attached to it, then only one of the attached keyboards will have its firmware updated.

Then `IORegistryEntryCreateCFProperty()` is used to query the `bcdDevice` property of the keyboard, which is a binary-coded decimal representation of the current firmware version of the keyboard. The keyboards we have observed have a `bcdDevice` value of `0x67` prior to the firmware update and a value of

0x69 afterwards. HIDFirmwareUpdaterTool will refuse to update a keyboard if it detects that its version is above 0x68. This version checking can be disabled by removing the near jump if above instruction at 0x00003373, i.e. changing 0x0f873b0a0000 to 0x909090909090.

5.5 Bootloader operation

The keyboard does not have an interrupt OUT endpoint, and so the control endpoint has to be used to enter the bootloader mode. After running the routine at 0x000020c3, the bootloader of the microcontroller is running and the keyboard is no longer running. By examining the IOUSBDevRequest passed to IOUSBDeviceClass::deviceDeviceRequest() in this routine, and after consulting the USB standard, we determined that an HID-specific request is being used to put the keyboard into bootloader mode. In particular, the bmRequestType was 0x21 and the bRequest was 0x09 indicating that a Set_Report request is sent to the keyboard. The wValue was 0x030a which means that the request is a feature report with a report ID of 0x0a. The wIndex was 0x0000 which means that the request was being sent to interface zero and the pData was simply 0x0a. The bootloader specifies in its device descriptor that it has a bDeviceClass of 0xFF and has a product ID of 0x0228. In order to see the firmware upgrade data go to the keyboard, we looked at the data being written over the USB bus in 0x00002d69.

The 260 byte routine starting at 0x00002d69 handles the bulk of the writing and reading to the keyboard over the USB bus. Each call results in a packet of data being sent to the keyboard followed by a read to get the response. The indirect call at 0x00002dcf calls interfaceWritePipe() in IOUSBInterfaceClass which sends over the contents of the 64 byte buffer starting at 0x0000a7c0, and the indirect call at 0x00002e07 calls interfaceReadPipe() in IOUSBInterfaceClass which reads the response into the 64 byte buffer starting at 0x0000a760. The code which prepares the first 64 byte buffer transmitted to the keyboard is shown in 6. The first 64 byte packet sent to the keyboard is

```
ff 38 00 01 02 03 04 05 06 07 00 00
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 53 00 00
00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00
```

and by the asl_log at 0x00005094, we ascertain that this call has the effect of instructing the bootloader to

```
00004c66   movl    0x000060a0,%esi
00004c6c   cmpb    $0x00,0x0000607d
00004c73   movb    $0xff,(%esi)
00004c76   movb    $0x38,0x01(%esi)
00004c7a   movb    $0x00,0x02(%esi)
00004c7e   movb    $0x01,0x03(%esi)
00004c82   movb    $0x02,0x04(%esi)
00004c86   movb    $0x03,0x05(%esi)
00004c8a   movb    $0x04,0x06(%esi)
00004c8e   movb    $0x05,0x07(%esi)
00004c92   movb    $0x06,0x08(%esi)
00004c96   movb    $0x07,0x09(%esi)
00004c9a   je      0x00004cc1
00004c9c   xorl    %edx,%edx
00004c9e   movl    %esi,%eax
00004ca0   leal    0x2d(%esi),%ecx
00004ca3   addb    (%eax),%dl
00004ca5   incl    %eax
00004ca6   cmpl    %ecx,%eax
00004ca8   jne     0x00004ca3
00004caa   movb    %dl,0x2d(%esi)
00004cad   movl    0x000060a0,%edx
00004cb3   movl    %esi,%eax
00004cb5   addl    $0x12,%edx
00004cb8   movb    $0x00,0x2e(%eax)
00004cbc   incl    %eax
00004cbd   cmpl    %eax,%edx
00004cbf   jne     0x00004cb8
```

Figure 6: USB packet setup routine

Return value	Reason for error
0x00	Device did not respond error
0x08	Flash protection error
0x10	Communication checksum error
0x80	Invalid command error

Figure 7: Error codes

go into bootloader mode. Observe the simple checksum calculation done from 0x00004c9e to 0x00004caa. For the example above, 0xff + 0x38 + 0x01 + 0x02 + $\cdots$ + 0x07 = 0x153 which is 0x53 mod 0x100.

The first two bytes of the USB packet is used to issue commands to the bootloader. We have already seen that ff 38 corresponds to entering the bootload mode, and by the asl_log calls in 0x00004ce0 and 0x00004dd9 respectively, we determined that ff 3a commands the bootloader to verify flash memory and ff 3b exits the bootloader. The command ff 39 means to write to flash memory. The return values from 0x00002d69 can be interpreted by examining the code from 0x00004b63 to 0x00004b8b. See Figure 7.

There is also a final checksum at the very end, which is computed as a sum of all the firmware blocks mod 0x10000. In this case, the data from blocks 0x02 to 0x4b are summed and the checksum is 0x4e41b which is 0xe41b mod 0x10000. This is stored in the last flash write packet in big endian format.

6 Exploitation

6.1 A benign exploit

For ethical reasons, the firmware modification we describe is benign. The firmware is modified so that the LED under the CAPS LOCK key of the keyboard will flash momentarily when the keyboard is first plugged into a system. However, malicious payloads can be developed by individuals with mal-intent.

Since the LED is active-low on pin P2.7 which corresponds to register 0x02 on the microcontroller, we searched the unobfuscated firmware image for instructions of the form MOV reg[0x02], expr which start with the opcodes 0x62 0x02. We found the sequence 0x62 0x02 0x80 in block 0x0c which did in fact turn out to be the instruction MOV reg[0x02], 0x80. The final checksum for the entire firmware image was 0x4e41b. By replacing 0x80 by 0x00, the new checksum is 0x4e39b and so 0xe41b in the last block has to be replaced by 0xe39b.

As a proof-of-concept, the following edited gdb session performs the changes mentioned above and demonstrates code execution on an Apple Aluminum keyboard.

```
$ gdb -q HIDFirmwareUpdaterTool
(gdb) tb *0x226a
Breakpoint 1 at 0x226a
(gdb) r -progress -pid 0x220
      kbd_0x0069_0x0220.irrxfw
HIDFirmwareUpdaterTool version 1.6.0
#1##2##3#
Breakpoint 1, 0x0000226a in ?? ()
(gdb) set {char}0x64b9 = 0x00
(gdb) set {short}0x845e = 0x9be3
(gdb) c
```

6.2 Rootkit persistence

Any malicious code embedded into the firmware would be immune to the typical rootkit detection methods which examine the integrity of the filesystem, check for hooks or direct kernel object manipulation, or detect hardware and/or timing discrepancies due to virtualization in the case of a virtual-machine based rootkit. Such code could also completely bypass the remote attestation of a Trusted Platform Module, if one were present in the computer. As far as everybody is concerned, our code is simply the user typing commands at the keyboard.

When the operating system enumerates the keyboard, a rootkit can use a custom control endpoint signal to indicate to malicious firmware on the keyboard that the attacker's rootkit is still operational on the host and that no action is necessary. However, if the keyboard does not receive such a signal, it can send malicious commands to the host computer to re-establish control to the attacker. It would obviously be ideal for this to occur after a certain period of inactivity by the legitimate user in the hopes that he/she is not using the computer to witness any unauthorized activity.

As an example, the keyboard could send the following keystrokes: COMMAND-SPACE, followed by terminal and RETURN, then followed by exec /bin/sh 0</dev/tcp/127.0.0.1/4444 1>&0 2>&0 and RETURN, where 127.0.0.1 is of course replaced by the IP address of the attacker's machine. In Mac OS X, COMMAND-SPACE activates Spotlight and terminal is typed into the Spotlight search box to launch the terminal application. Then exec is used to send a shell back to the attacker [14]. The above command just sends a shell back to port 4444 on localhost. The firewall in the Mac OS X Leopard operating system is by default not enabled, and in any case, does not block outgoing connections. In the event that the user uses an outbound firewall like Little Snitch, an extra RETURN at the end of the above sequence of keystrokes will select the default option of allowing the outbound TCP connection from Terminal.app. This would allow a rootkit to persist even if the user has

performed a clean re-installation of Mac OS X on their computer.

6.3 Denial of service

It is very easy to brick a keyboard by interrupting the bootloader during firmware re-programming. However, a keyboard bricked in this way can generally be unbricked by reflashing to `0x69` firmware. If the bootloader still runs, then `HIDFirmwareUpdaterTool` can be invoked with the arguments `-progress -pid 0x228 kbd_0x0069_0x0220.irrxfw`. We did not investigate whether the bootloader itself could be overwritten.

7 Conclusion

Reverse-engineering Apple's keyboard firmware update was a fairly simple exercise. Apple could have attempted to obfuscate the binary as they do in the well-known segments with the `0x8` flags set on the LC_SEGMENT load commands in certain Apple binaries (indicating AES encryption) such as Finder.app, Dock.app, etc. which are designed to hinder the piracy of the Mac OS X operating system. Apple could have issued a `PT_DENY_ATTACH ptrace()` request as they do in iTunes as an anti-debugging measure. However, such methods can be bypassed without much difficulty.

For a device as simple in design as a keyboard, it is hard to imagine why a firmware update mechanism is even required. Many other devices have firmware update mechanisms that we believe can also be exploited by attackers for malicious purposes.

References

[1] http://www.cert.org/advisories/CA-2003-21.html, 2003.

[2] http://www.usb.org/developers/devclass_docs/DFU_1.1.pdf, 2004.

[3] http://www.acq.osd.mil/dsb/reports/2005-02-HPMS_Report_Final.pdf, 2005.

[4] http://www.apple.com/support/windowsvirus/, 2006.

[5] http://www.mcd-holdings.co.jp/news/2006/release-061013.html, 2006.

[6] http://www.tomtom.com/news/category.php?ID=2&NID=349&Language=1, 2007.

[7] http://www.seagate.com/www/en-us/support/downloads/personal_storage/ps3200-sw, 2007.

[8] http://web.archive.org/web/20080211095252/www.insignia-products.com/news.aspx?showarticle=13, 2008.

[9] http://www.auscert.org.au/render.html?it=9077, 2008.

[10] http://www.asus.co.jp/news_show.aspx?id=12964, 2008.

[11] http://www.redhat.com/archives/fedora-announce-list/2008-August/msg00012.html, 2008.

[12] http://www.infobyte.com.ar/, 2008.

[13] http://support.apple.com/downloads/Aluminum_Keyboard_Firmware_Update_1_0, 2008.

[14] http://labs.neohapsis.com/2008/04/17/connect-back-shell-literally/, 2008.

[15] CAPPOS, J., SAMUEL, J., BAKER, S., AND HARTMAN, J. H. A look in the mirror: Attacks on package managers. In *Proceedings of the 15th ACM Conference on Computer and Communications Security* (Oct. 2008).

[16] KING, S. T., TUCEK, J., COZZIE, A., GRIER, C., JIANG, W., AND ZHOU, Y. Designing and implementing malicious hardware. In *Proceedings of the First USENIX Workshop on Large-Scale Exploits and Emergent Threats* (Apr. 2008).

[17] PARKER, T., SACHS, M., SHAW, E., STROZ, E., AND DEVOST, M. G. *Cyber Adversary Characterization: Auditing the Hacker Mind*. Syngress, 2004.

[18] ROLDAN, R., MIYAMOTO, I., AND LEON, T. FBI Criminal Investigation: Cisco Routers, 2008.

[19] SHAH, G., MOLINA, A., AND BLAZE, M. Keyboards and covert channels. In *Proceedings of the 15th USENIX Security Symposium* (Aug. 2006).

[20] ZHUANG, L., ZHOU, F., AND TYGAR, J. D. Keyboard acoustic emanations revisited. In *Proceedings of the 12th ACM Conference on Computer and Communications Security* (Nov. 2005).

Notes

[1] This cannot be seen from the assembly listing due to message passing in Objective C.

[2] Curiously, Apple's developer documentation says that `IOPMCopyBatteryInfo()` is unsupported on all Intel CPU based systems and yet, Apple uses it in this routine anyway.

[3] Well, it is almost a tree. A RAID disk controller is an example of an exception.