

iSeeYou: Disabling the MacBook Webcam Indicator LED

Matthew Brocker
Johns Hopkins University

Stephen Checkoway
Johns Hopkins University

Abstract—The ubiquitous webcam indicator LED is an important privacy feature which provides a visual cue that the camera is turned on. We describe how to disable the LED on a class of Apple internal iSight webcams used in some versions of MacBook laptops and iMac desktops. This enables video to be captured without any visual indication to the user and can be accomplished entirely in user space by an unprivileged (non-root) application.

The same technique that allows us to disable the LED, namely reprogramming the firmware that runs on the iSight, enables a virtual machine escape whereby malware running inside a virtual machine reprograms the camera to act as a USB Human Interface Device (HID) keyboard which executes code in the host operating system.

We build two proofs-of-concept: (1) an OS X application, *iSeeYou*, which demonstrates capturing video with the LED disabled; and (2) a virtual machine escape that launches *Terminal.app* and runs shell commands. To defend against these and related threats, we build an OS X kernel extension, *iSightDefender*, which prohibits the modification of the iSight’s firmware from user space.

I. INTRODUCTION

At the beginning of the 2008 school year, the Lower Merion School District provided a MacBook laptop to each enrolled student. These laptops came pre-loaded with the *LANrev* remote administration tool (RAT) which allowed school district officials to, among other things, capture images from the MacBooks’ built-in iSight webcam. During the following 18 months, officials captured more than 30 thousand images from these webcams [5, 6]. The first indication that images were being captured was every time the software took a picture, the green indicator LED would briefly illuminate [5, 6, 40]. Some teachers were so concerned by this they they covered the lens of the webcams on their own laptops [6]. Here, the indicator LED worked exactly as it was supposed to and alerted the users that they were being photographed.

The possibility that a webcam could be capturing pictures without the LED illuminating has led to suggestions that owners should tape over the webcam [41] as well as products designed to cover the camera such as stickers [10] and the amusingly named *iPatch* [53].

This incident illustrates the dangers of passive sensors like cameras and microphones. Unlike active input devices like keyboards and mice that require user actions to provide input, a passive sensor requires no action on the part of the user to capture input. Indeed, a user is typically unaware that input is being captured at all unless specific mechanisms are built into

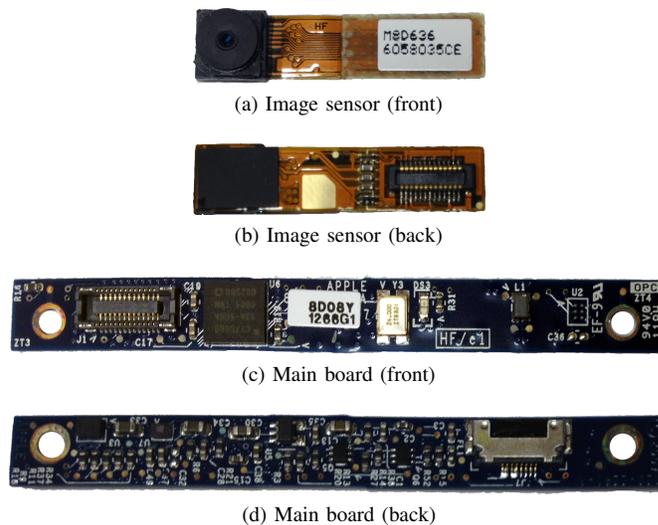


Fig. 1: The iSight from a 2008 MacBook we studied.

the technology to indicate that the sensor is currently in use. Such mechanisms include camera-use indicator LEDs, shutter sounds on cell phone cameras, and GPS-use indicator icons on mobile devices and laptops.

In the past few years, the ever-expanding set of sensors present in commodity laptops and smart phones has prompted the security and privacy community to begin researching ways to detect and limit the undesired use of sensors [20, 22, 24, 25, 29]. At the same time, researchers have demonstrated attacks exploiting the presence of sensors such as a clickjacking attacks against Adobe Flash to gain access to the camera and microphone [23] from a malicious web page and exfiltrating audio from microphones in modern automobiles [11]. (See Section II for more examples.)

Our results in this paper demonstrate that, at least in some cases, people have been correct to worry about malware covertly capturing images and video. We show a vulnerability in the iSight webcam that affects a particular range of Apple computers—including the MacBooks given to the students in the Lower Merion School District—that can be exploited to turn on the camera and capture images and video without the indicator illuminating.

At a high level, our investigation of the iSight revealed that it is designed around a microprocessor and a separate image sensor with an indicator LED sitting between them such

that whenever the image sensor is transmitting images to the microcontroller, a hardware interlock illuminates the LED. We show how to reprogram the microcontroller with arbitrary, new firmware. This in turn enables us to reconfigure the image sensor, allowing us to bypass the hardware interlock and disable the LED. We also show a new method of performing a virtual machine escape based on our ability to reprogram the microcontroller.

Specifically, our technical contributions in this paper are five-fold:

- 1) We describe the architecture of the Apple internal iSight webcam found in previous generation Apple products including the iMac G5 and early Intel-based iMacs, MacBooks, and MacBook Pros until roughly 2008 (Section III).
- 2) We demonstrate how to bypass the hardware interlock that the iSight uses to turn on the indicator LED whenever the camera is capturing images or video (Section IV) and provide a proof-of-concept user space application, *iSeeYou*, to do so (Section V).
- 3) We demonstrate how to use the capability developed to bypass the hardware interlock to achieve a virtual machine escape (Section VI).
- 4) We develop an OS X kernel extension, *iSightDefender*, to defend against these attacks (Section VII).
- 5) We sketch the design space for building a secure camera module (Section VIII).

The ability to bypass the interlock raises serious privacy concerns and the technical means by which we accomplish it raises additional security concerns which we discuss in Section IX.

Threat model. To mount our main attack where we capture video without any external indication to the victim, we assume that an attacker is able to run native code on the victim’s computer as an unprivileged user. Further, we assume the code is unencumbered by defenses such as Apple’s *App Sandbox* [2] which is used for applications downloaded from the Mac App Store but by little else. This assumption is quite mild and would typically be satisfied by malware such as RATs.

For the virtual machine escape, we assume the attacker has code running locally in the virtual machine and with whatever privileges the guest OS requires to communicate with USB devices. We also assume that the virtual machine monitor has exposed the iSight device to the virtual machine. This second assumption is quite strong as virtual machine monitors typically do not expose USB devices to the guest OS unless the user specifically configures it to do so, for example to use video conferencing software.

Generality of results. We stress that our main result—disabling the iSight LED—only applies to the first generation internal iSight webcams and we make no claims of security or insecurity of later models, including the most recent (renamed) FaceTime cameras. The virtual machine escape described in

Section VI likely holds for other USB devices that use the Cypress EZ-USB chip used in the iSight, but we have not yet tested other devices.

II. RELATED WORK

General purpose computers contain a variety of processors designed for performing specialized tasks other than general-purpose computation. Examples include graphics processing units (GPUs) which produce video output; processors in network interface controllers (NICs) which perform network packet processing; microcontrollers in peripherals such as keyboards, mice, and webcams; microcontrollers in laptop batteries; and, in some systems, baseboard management controllers (BMCs) which enables out-of-band system management independent of the host computer’s CPU.

Security researchers have only recently begun examining these additional processors and the firmware that runs on them. In many cases, the designers of these systems appear not to have appreciated the security implications of their interfaces and implementations.

Perhaps the most well-studied processor apart from the CPU is the GPU. Vasiliadis et al. [54] demonstrate using the GPU to harden malware against detection by using the GPU to implement unpacking and runtime polymorphism. Ladakis et al. [31] use the GPU’s direct memory access (DMA) capability to monitor the system’s keyboard buffer to build a keylogger. Beyond GPU malware itself, researchers have used the GPU to accelerate malware detection [30] and intrusion detection systems [46].

Duflot and Perez [17] demonstrate exploiting a NIC to achieve arbitrary code execution. In follow up work, Duflot et al. [18] build a NIC malware detection framework.

Miller [36] demonstrates how to communicate with Apple laptop batteries using the System Management Bus, authenticate to the battery to “unseal” it, and change both configuration values and firmware. This enables overcharging the battery resulting in overheating and, potentially, leading to a fire.

Tereshkin and Wojtczuk [52] introduce the concept of a “Ring –3” rootkit which runs on Intel’s Active Management Technology (AMT) hardware which has a processor independent of the host CPU with a separate interface to the NIC and DMA access to main memory.

In a very similar vein, Farmer [21] discusses weaknesses and vulnerabilities in the Intelligent Platform Management Interface (IPMI)—the standard interface to the baseboard management controller (BMC). Like AMT, a BMC has direct access to the host system but its operation is completely independent making exploits both extremely powerful and difficult to detect. Moore [39] builds on this work to produce a penetration tester’s guide for examining IPMI and BMCs.

A webcam is just a particular type of sensor attached to a computing device. Others include microphones, accelerometers, and GPS sensors. Our work joins an emerging line or research on the security and privacy implications of such sensors. For example, Schlegel et al. [50] show how to use a mobile phone’s microphone to extract credit card

numbers and PINs from spoken and tone-based interfaces. Marquardt et al. [34], Miluzzo et al. [38], Owusu et al. [43] use smartphone accelerometers to extract information about key presses. Checkoway et al. [11] extract audio and GPS coordinates from automobiles. Templeman et al. [51] use smartphone cameras to covertly take pictures which are then used to create 3D models of physical spaces.

Our virtual machine escape (Section VI) is not the first to emulate a USB Human Interface Device (HID) such as a mouse or keyboard. Wang and Stavrou [56] among other things, use a compromised smart phone to act as a USB HID keyboard and send key presses to the host system. Kennedy and Kelley [28] use a small microcontroller to interact with the Windows Powershell. Pisani et al. [45] similarly describe having USB devices pose as HID keyboards to control the computer. Elkins [19] adds a RF receiver for remote controlling a fake HID keyboard.

III. INTERNAL ISIGHT ARCHITECTURE

This section describes the architecture of the internal iSight webcam in sufficient detail to understand how the multi-step attack described in Section IV works. Readers who are already familiar with the iSight or the Cypress EZ-USB or who are not interested in the low-level details of the device are encouraged to skip directly to Section IV and use this section and Figure 2, in particular, as a reference as needed.

The internal iSight consists of a Cypress CY7C68013A EZ-USB FX2LP, a Micron MT9V112 CMOS digital image sensor, a 16 byte configuration EEPROM, and an indicator LED (see Figure 1). A block diagram is given in Figure 2.

A. Cypress EZ-USB

The host computer interacts with the iSight entirely through a USB connection to the Cypress EZ-USB. The EZ-USB is responsible for handling all USB requests and sending replies including video data.

The EZ-USB has an internal Intel 8051-compatible microcontroller core and 16 kB of on-chip RAM accessible as both code and data “main” memory¹ but lacks persistent storage [13]. In general, the firmware for the 8051 core can be located in one of three locations: (1) external memory such as flash or EPROM attached to the EZ-USB address/data bus; (2) an I²C EEPROM; or (3) loaded from USB. The iSight loads its firmware at boot from the host computer over USB (see Section IV-B).

B. Micron digital image sensor

The Micron digital image sensor is a low-power system-on-a-chip (SOC) capable of producing an image in several formats. The image sensor is configured by the I²C interface which can read from and write to several hundred configuration

¹The standard 8051 is a Harvard architecture which has separate code and data memory differentiated by hardware signals. In the configuration used by the iSight, the signals are combined effectively giving a single main memory address space.

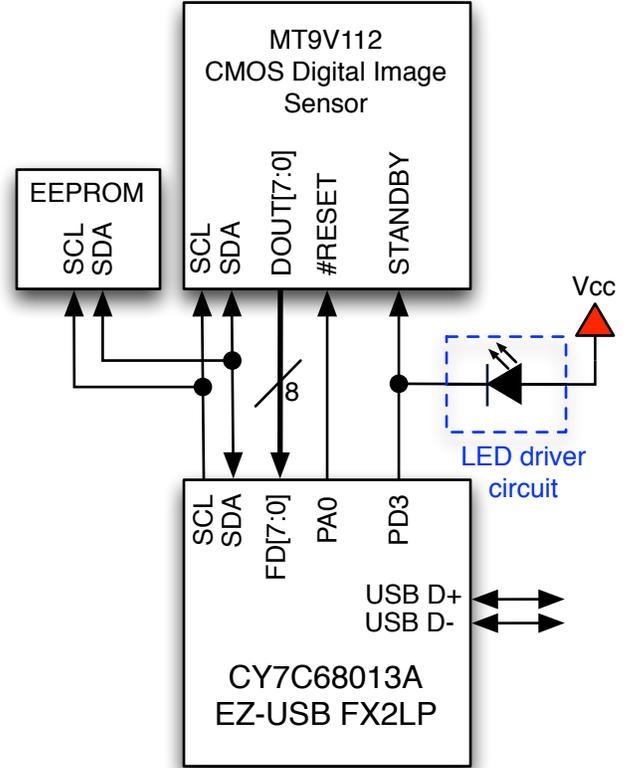


Fig. 2: Internal iSight architecture block diagram consisting of a Cypress EZ-USB, a Micron digital image sensor, a 16 byte configuration EEPROM, and an indicator LED.

registers [35]. In addition to the I²C interface, several hardware signals influence the operation of sensor.

The most important signals from our perspective are the active-low #RESET and active-high STANDBY signals. The corresponding hardware pins are connected directly to the EZ-USB’s general purpose I/O (GPIO) pins. As shown in Figure 2, #RESET is connected to pin 0 of GPIO port A and STANDBY is connected to pin 3 of GPIO port D. The other connection between the image sensor and the EZ-USB shown in Figure 2 DOUT [7 : 0] → FD [7 : 0] is an 8 bit unidirectional bus which transfers pixel data to the EZ-USB’s FIFO interface. Other, less important, control signals are omitted from the diagram.

The #RESET signal performs a hardware reset, resetting all configuration registers to their default value. The STANDBY signal controls controls output enable and power down functions. That is, when STANDBY is asserted, the image sensor stops producing data on DOUT [7 : 0] which enters the high impedance state as well as allowing the image sensor to transition to a low power state.

C. Configuration EEPROM

The first byte of the 16 byte EEPROM controls whether the EZ-USB loads its firmware from USB or from the EEPROM itself. When set to load firmware from USB, as the iSight does, the EEPROM contains the USB vendor ID (VID), product

TABLE I: Relation between the PD3 GPIO, the STANDBY signal, and the LED.

PD3	STANDBY	LED
High	Asserted	Off
Low	Deasserted	On

ID (PID), device release number, and a configuration byte for the initial device enumeration. Once the EZ-USB has enumerated using the VID, PID, and release values, software on the host computer can load the firmware. The iSight initially enumerates with vendor ID 0x05ac (Apple, Inc.) and product ID 0x8300 (Built-in iSight (no firmware loaded)).

D. Indicator LED

Since the purpose of the indicator LED is to illuminate whenever the camera is capturing video, a LED driver circuit is connected directly to the STANDBY input of the image sensor (see Figure 2). In this way, whenever PD3 is high—that is, STANDBY is asserted—the LED is off and whenever PD3 is low—so STANDBY is deasserted and the image sensor is producing output—the LED is on. Since the LED is controlled by the same output that controls STANDBY, there is no danger that firmware on the EZ-USB could deassert STANDBY and turn the LED off (see Table I). However, as we demonstrate in Section IV, we can bypass the STANDBY signal such that changing PD3 allows us to control the LED without affecting the operation of the image sensor.

IV. DISABLING THE INDICATOR LED

Disabling the indicator LED on the iSight entails two requirements. First, as described in Section III, the indicator LED is directly connected to the STANDBY pin on the image sensor. In order to disable the LED, we need to keep STANDBY asserted. Since asserting STANDBY will disable the image sensor output, we need to configure the image sensor to ignore STANDBY before we assert this signal. Second, we need a way to modify the firmware on the EZ-USB to in order to configure the image sensor appropriately as well as keep STANDBY asserted whenever we want the LED to stay off.

A. Bypassing the STANDBY signal

The Micron image sensor has a 16 bit configuration register, RESET (which is distinct from the #RESET power on reset signal). RESET is addressable from the I²C interface at address 0x0D in register page 0 [35]. The most significant 8 bits control hardware clocks and how bad frames should be handled which are of no interest to us and can be left as 0. The least significant 8 bits have the following functionality as described in the image sensor data sheet [35, Table 13]:

- Bit 7. Prevent STANDBY from affecting entry to or exit from the low-power state if set.
- Bit 6. Prevent STANDBY from contributing to output enable control if set.
- Bit 5. Reset the SOC (but not the sensor) if set.

- Bit 4. Disable pixel data output if set.
- Bit 3. Chip enable. Normal operation if set, no sensor readout otherwise.
- Bit 2. Software standby if set, otherwise normal operation.
- Bit 1. Restart reading an image frame.
- Bit 0. Reset the sensor to its default state if set, normal operation otherwise.

Bits 0, 1, and 5 are of no interest and can be set to 0 but the remaining 5 bits enable us to bypass the STANDBY signal while still maintaining normal operation. This includes entering a (software) standby state and disabling output when appropriate.

When the iSight is first powered up (or, more precisely, when #RESET becomes deasserted), the RESET register has value 0x0008; that is, normal operation and STANDBY affects the low-power state and output enable. If RESET is set to 0x00c8, then the camera has normal operation but STANDBY is effectively bypassed. When it becomes desirable for the camera to enter the standby state, RESET can be set to 0x00d4 which disables output and enters the software standby state.

With RESET set to either 0x00c8 or 0x00d4, the hardware STANDBY signal is ignored. This enables the use of the EZ-USB PD3 output to control the LED independent of the standby state of the image sensor.

B. Programming the EZ-USB

When the iSight is first powered, it checks the configuration EEPROM and then waits for programming over USB (see Section III-C). The *AppleUSBVideoSupport* I/O Kit driver matches the vendor ID (VID) and product ID (PID). The driver loads and the `AppleUSBCamera::start()` function downloads the camera’s firmware (stored in the `gTheFirmware` array) to the EZ-USB using a series of vendor-specific USB “Firmware Load” device requests [13, Section 3.8]. The camera will then reenumerate and function as a webcam.

One approach to change the firmware on the camera is to modify the *AppleUSBVideoSupport* driver to contain different firmware. A second approach would be to provide a new driver that matches the VID/PID and provides a higher probe score [3]. The new driver would run at system start up instead of Apple’s driver and download the new firmware to the camera. These approaches have two major drawbacks. The first drawback is that they rely on programming the iSight when it is in its unprogrammed state which only happens when the camera is first powered by the USB bus. The second drawback is that root access is required in order to modify the existing driver or load a new driver.

A third approach overcomes both drawbacks by letting the iSight be programmed with the legitimate firmware when it is first powered. Once the firmware has been loaded onto the camera, it can be reprogrammed at any time using “Firmware Load” vendor-specific USB device requests. Furthermore, it can be reprogrammed from any user space process.

V. PROOF OF CONCEPT

The discussion in Section IV shows that, in principle, it is possible to modify the legitimate firmware to disable the LED. In this section, we describe the proof-of-concept application, *iSeeYou* we created which reprograms the iSight to add the capability to enable or disable the LED using a new vendor-specific USB device request.

A. Modifying the firmware

Although one could reimplement the camera functionality, we opted to create new firmware by appending new binary code to the legitimate firmware and patching it to call our new code. The first step is to extract the legitimate firmware from the *AppleUSBVideoSupport* driver.²

The firmware consists of an 8 byte header followed by a sequence of triples: a 2 byte size, a 2 byte address, and size-bytes of data. This format corresponds exactly to the “C2 Load” format of the EEPROM for loading firmware directly from the EEPROM [13, Table 3-6]. Each triple specifies the data that should be written to the EZ-USB’s main memory at a given address. By stripping off the header and the final triple,³ we can construct the “raw” firmware image. The raw firmware can then be analyzed, for example, by using IDA Pro.

The raw firmware is structured similarly to sample code provided in the Cypress EZ-USB FX2LP Development Kit [14] including a hardware initialization function and USB events that are serviced by a main loop based on state bits set by interrupt handlers.

To the legitimate firmware, we add two bits of state, “is the sensor in software standby or running” and “is the LED enabled or disabled,” as well as four new functions, `reset_sensor`, `enter_standby`, `exit_standby`, and `handle_led_control`.

When the LED is enabled, the behavior of the camera is indistinguishable from the normal behavior. That is, when the camera is in its standby state the LED is off and when the camera is in its running state, the LED is on.

The legitimate firmware contains a function to reset and configure the image sensor. This is called both from the hardware initialization function and the handler for the USB set interface request. It begins by deasserting the `STANDBY` signal and asserting the `#RESET`. After a short spin loop, it deasserts `#RESET` and, depending on the function argument, deasserts `STANDBY`. It then proceeds to configure the image sensor. We patch the firmware to call `reset_sensor` instead of this configuration function in both locations. The `reset_sensor` function reimplements the reset functionality but adds a call to the function which writes to the I²C bus to program the `RESET` register to bypass the `STANDBY` signal (see Section IV-A). At this point, if the LED has been disabled or the argument indicates that it should enter the standby

state, the `STANDBY` signal is asserted to turn off the LED which will have momentarily illuminated during the reset sequence. Otherwise, the sensor is left running and the LED is enabled so `STANDBY` remains deasserted and the LED stays on. Finally, the `reset_sensor` function jumps into the middle of the configuration function, just past the `#RESET` and `STANDBY` manipulating code, in order to perform the rest of the configuration.

The `enter_standby` and `exit_standby` functions update the bit of state which records if the image sensor is running or in standby. Then, based on whether the LED is enabled or not, they deassert (resp. assert) `STANDBY` as needed to turn the LED on (resp. off). Finally, these functions use I²C to program the `RESET` register to enter or exit software standby. Each location in the legitimate firmware which sets the state of the `STANDBY` signal is patched to call its new, corresponding standby function instead.

The final function, `handle_led_control` is responsible for handling a new vendor-specific USB device request. The main loop in the legitimate firmware which handles USB device request “setup” packets is patched to instead call `handle_led_control`. If the `bRequest` field of the request does not match the new vendor-specific value, then it jumps to the legitimate handler. Otherwise, based on the `wValue` field of the request, the LED is enabled or disabled. As with the other functions, the LED is then turned on if it has been enabled and the image sensor is running. Otherwise, it is turned off.

B. Demonstration application: iSeeYou

iSeeYou is a simple, native OS X application; see Figure 3. When *iSeeYou* starts, it checks for the presence of a built-in iSight using the appropriate vendor and product IDs. If the iSight is found, *iSeeYou* initiates the reprogramming process using the modified firmware described above. Once the camera has been reprogrammed and has reenumerated, the start/stop button begins/ends capturing and displaying video. The LED Enable/LED Disable control sends USB device requests with the new vendor-specific value to enable/disable the indicator LED while video is being captured. Finally, when the user quits *iSeeYou*, the camera is reprogrammed with the legitimate firmware.

VI. VIRTUAL MACHINE ESCAPE

The reprogrammability of the iSight firmware can be exploited to effect a virtual machine escape whereby malware running in a guest operating system is able to escape the confines of the virtual machine and influence the host operating system.

One method is to reprogram the iSight from inside the virtual machine to act as a USB Human Interface Device (HID) such as a mouse or keyboard. Once the iSight reenumerates, it would send mouse movements and clicks or key presses which the host operating system would then interpret as actions from the user.

To demonstrate the feasibility of a virtual machine escape from a VirtualBox virtual machine, we implemented a USB

²There are several open source tools to perform this task, e.g., iSight Firmware Tools [7], several of which include binary patching to fix bugs in the USB interface descriptors.

³The final triple stores a single 0x00 byte to address 0xE600 which takes the Intel 8051 core out of reset so that it can begin executing instructions.



Fig. 3: *iSeeYou* running on a white MacBook “Core 2 Duo” capturing video from the internal iSight with the LED (the black dot to the right of the square camera at the top, center of the display bezel) unilluminated.

HID keyboard which, once loaded, performs the following actions in order:

- 1) send a “host key” press;
- 2) send command-space to open *Spotlight*;
- 3) send the key presses for “Terminal.app” one at a time;
- 4) wait a few seconds, send a return key press, and wait a few more seconds for the *Terminal* to open;
- 5) send the key presses for a shell command followed by a return key press;
- 6) disconnect from the USB bus and modify its USB device descriptor to use the product ID 0x8300—the PID for the iSight in its unprogrammed state; and
- 7) reenumerate.

The VirtualBox host key, which defaults to the left command key on a Mac host, releases keyboard ownership, causing the rest of the key presses to go to the host operating system rather than to the guest operating system [42, Chapter 1].

Figure 4 shows an iSight that has been reprogrammed from inside a VirtualBox virtual machine sending key presses to *Spotlight*, instructing it to open *Terminal.app*.

When a new keyboard is first plugged into the computer, the *Keyboard Setup Assistant* asks the user to press several keys in order to determine the keyboard layout. This behavior appears to be controlled by the vendor ID, product ID, and device release number. By using the appropriate values for an

Apple USB keyboard, the assistant does not appear and there is no visual indication that the operating system believes a new keyboard has connected.

The shell command entered into the *Terminal* is unconstrained and could, for example, use *curl* to download arbitrary, new code and run it.

After the iSight has finished typing commands, it reenumerates as an unprogrammed iSight which causes the *AppleUSB-VideoSupport* driver to reprogram it with the legitimate iSight firmware, removing evidence that the iSight was the infection vector.

Although we use the iSight to escape from the virtual machine, in theory, any EZ-USB device which is accessible from inside the virtual machine can be reprogrammed to behave as a HID keyboard.

The one major limitation is that the USB device must be connected to virtual machine before the attack is possible. By default, virtual machine monitors do *not* provide this connection for most devices and thus malware would need to coerce the user into establishing the connection.

Even with the device connected to the virtual machine, there is no feedback to the firmware that the attack is proceeding as planned. All it can do is send key presses in response the USB polling. If the user is sitting in front of the computer, the key presses sent by the iSight may be apparent and the

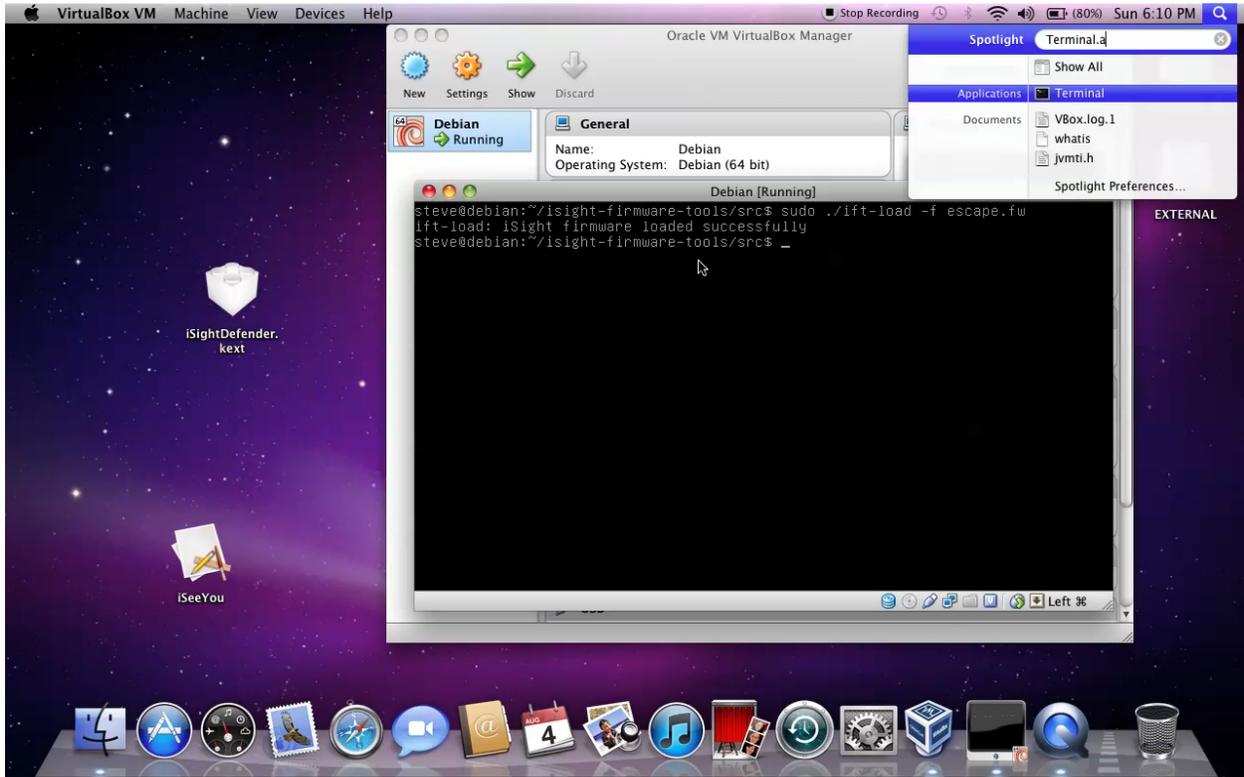


Fig. 4: Virtual machine escape. The iSight has been reprogrammed to act as a USB keyboard from inside a VirtualBox virtual machine and it is sending key presses to the host operating system. It is in the middle of entering “Terminal.app” into *Spotlight*.

user can interfere by performing an action such as typing or clicking the mouse. One way to partially compensate is to decrease the USB polling interval by changing the USB endpoint descriptors in the firmware allowing the iSight to send key presses more quickly.

Each operating system has its own policy which governs a process’s ability to send USB device requests. On Linux, this is controlled by *udev*. In Figure 4, we used *sudo* to bypass the access controls. Alternatively, the appropriate permissions could be granted to the user. One of these steps is required even though the host operating system, OS X, imposes no restrictions on the use of USB device requests. Since each guest operating system controls access to the USB device once it has been connected to the virtual machine, to perform an escape, malware must first acquire sufficient privileges in the guest operating system to reprogram the camera—a potentially nontrivial feat.

VII. DEFENSES

There are several approaches one can envision to defend the iSight against the attacks described in the previous sections. One can change (1) the hardware, (2) the firmware on the EZ-USB (unfortunately this is not effective, see below), or (3) the software on the host system. See Table II for an overview of possible defenses and their efficacy.

The most comprehensive defense would be to change the hardware used in the iSight. See Section VIII for several

TABLE II: Overview of possible defenses.

Defense	Deployable	User	Root
Change hardware	No	Yes	Yes
Change firmware	Yes	No	No
<i>App Sandbox</i>	Yes	Some	No
<i>iSightDefender</i>	Yes	Yes	No

A “Yes” in the Deployable column indicates that the defense could be deployed to existing computers. A “Yes” in the User (resp. Root) column indicates that the defense would prevent an unprivileged (resp. root) process from reprogramming the iSight. A “Some” indicates that some reprogramming attempts would be prevented but others allowed.

secure hardware designs. Of course, changing the hardware is not a deployable solution for existing devices.

If the hardware must remain the same, then if the firmware on the camera could be changed to disallow future reprogramming, then the camera would be secure against our attacks. Unfortunately, the “Firmware Load” USB device request used to reprogram the 8051 core is handled entirely by the EZ-USB device itself and cannot be blocked or handled by the 8051 itself [13, Section 3.8]. Thus no matter how one programs the device’s firmware, it can be reprogrammed by an attacker who can send basic USB messages to the camera.

Apple deploys sandboxing technology called the *App Sandbox*⁴ [2] which can prevent applications inside the sandbox from accessing the iSight. Specifically, the `com.apple.security.device.camera` entitlement enables an application to capture still images and video from cameras, including the internal iSight. The `com.apple.security.device.usb` entitlement enables applications to access USB devices.

Any *App Sandbox*-protected application lacking the `usb` entitlement would be prohibited from reprogramming the iSight and thus prohibited from disabling the indicator LED. Although an application with the `usb` entitlement but lacking the `camera` entitlement would be prohibited from using the high-level APIs for accessing the camera, such as the QTKit API [4], it could easily reprogram the camera to not appear as a USB video class (UVC) device and instead transfer the frames of video using a custom protocol.

The major drawback to using the *App Sandbox* to protect the camera is that applications need to opt into the protection, something malware is unlikely to do. Worse, the *App Sandbox* has, at times, been broken allowing applications to escape from the restrictions [12, 37].

Perhaps the best way to defend against reprogramming the iSight without changing the hardware is to modify the operating system to prevent particular USB device requests from being sent to the camera. We have built such a defense structured as an OS X kernel extension called *iSightDefender*.

When iSight is powered for the first time, it enumerates with vendor ID `0x05ac` and product ID `0x8300` and is programmed with the legitimate firmware via the *AppleUSBVideoSupport* kernel extension as described in Sections III-C and IV-B. When it reenumerates with product ID `0x8501` the kernel matches and loads the normal drivers as well as *iSightDefender*.

I/O Kit kernel drivers are written in a subset of C++ and each USB device is represented by an object of class `IOUSBDevice` which is responsible for communicating with the hardware by sending messages to objects in lower layers of the USB stack. When *iSightDefender* is started, it overwrites the C++ virtual method table of its “provider” `IOUSBDevice` to point to the virtual method table of a subclass of `IOUSBDevice`.⁵ The subclass overrides the four `DeviceRequest` member functions. The overridden implementations check if the device request is for the “Firmware Load” vendor-specific request and, if so, log the attempt in the system log and block the request.

iSightDefender is able to block all user space reprogramming attempts,⁶ including those mounted from within a virtual

machine. The latter requires some care as the normal drivers that match against the `IOUSBDevice` are unloaded and the virtual machine monitor’s own driver is loaded in their place.

Using *iSightDefender* raises the bar for attackers by requiring the attacker to have root privileges in order to reprogram the iSight. In some sense, this is the strongest possible software-based defense. Since malware running as root would have the ability to replace or modify kernel code, any defense implemented in the kernel can, theoretically, be bypassed. Despite this limitation, we believe it is a step in the right direction and encourage its use.

iSightDefender, and its source code, is freely available.⁷

VIII. SECURE CAMERA DESIGNS

When designing a secure camera, there are two main considerations. First, for sensors such as cameras and microphones, an indicator that the sensor is recording is essential to prevent surreptitious recording. (Although laptop microphones do not, in general, have indicators, it is common for stand alone USB microphones; see [27] for an example.) For the highest level of assurance that the indicator cannot be bypassed, the indicator should be controlled completely by hardware.

Second, as with any peripheral connected to the computer, a vulnerability in the software running on the peripheral or the ability to reprogram the software enables an attacker to leverage all of the capabilities of the peripheral. Section II contains numerous examples of this. The virtual machine escape in Section VI is another example where an attacker leverages the USB connection and the ability of the EZ-USB to mimic any USB device to the host computer. Apple’s most recent FaceTime cameras in its 2013 MacBook Air model eschews USB 2.0. Instead, the camera is connected to the host computer over PCIe [33]. Vulnerabilities in the camera would potentially enable an attacker to have DMA access to the host system. This is a significantly stronger capability than USB access.

A. Secure indicators

Laptop cameras are typically constructed by pairing a CMOS image sensor-on-a-chip (e.g., the Micron MT9V112 found in the iSight or the Toshiba TCM8230MB(A)) with a separate microcontroller that handles communication with the host computer (e.g., the EZ-USB FX2LP found in the older MacBooks or the Vimicro VC0358 [55] found in more recent MacBook Pros [26]). There are, of course, many possible combinations of image sensors and microcontrollers one could use.

Image sensors-on-a-chip tend to have a number of common features that can be used to build a secure indicator.

- 1) Separate power connection for CMOS sensor itself. For example, `VAAPIX` on the MT9V112 powers its pixel array and `PVDD` on the TCM8230MB(A) powers its photo diode. A GPIO pin on the microcontroller can be connected to both the LED driver circuit and the

⁴Formerly codenamed *Seatbelt*.

⁵There seems to be no supported mechanism for interposing on USB device requests. The authors appreciate the irony of using virtual table hijacking — a common hacker technique — for defending against attack.

⁶In fact, *iSightDefender* worked so well that one author spent more than an hour attempting to diagnose (nonexistent) problems with *iSeeYou* before noticing the tell-tale lines in the system log indicating that *iSightDefender* had been loaded by a computer restart and it was blocking reprogramming requests.

⁷<https://github.com/stevecheckoway/iSightDefender>

CMOS sensor power supply circuit. Whenever images are to be captured, the microcontroller sets its GPIO pin appropriately, power is supplied to the sensor and the LED turns on.

- 2) #RESET pins. The LED driver circuit can be connected to the #RESET pin and a GPIO pin on the microcontroller. The microcontroller would hold the image sensor in reset whenever it was not capturing images. Compared to the power connection for CMOS sensor, holding the entire sensor-on-a-chip in reset means that before images could be captured, the sensor would need to be reconfigured. Reconfiguring typically means sending a few dozen bytes over an I²C or SPI bus. This introduces a slight delay.
- 3) Output clocks and synchronization signals. Image sensors typically latch outputs on one edge of an output clock signal and image consumers are expected to read the data on the other edge of the clock. In addition, there are signals used to indicate which part of the image the latched data represents. For example, the MT9V112 has FRAME_VALID and LINE_VALID signals indicating when it's outputting a frame or a line within the frame, respectively, whereas the TCM8230MB(A) has VD and HD for vertical and horizontal synchronization. These pins can also be used to control the LED by adding some simple hardware that drives the LED if it has seen one of these signals change in the past few milliseconds.

Depending on the specifics of the image sensor output signal, a retriggerable, monostable multivibrator can be used to drive the LED as long as its input changes sufficiently often. The multivibrator's output pulse width needs to be set appropriately such that it is triggered frequently enough to continuously drive the LED while images are being recorded.

Some care must be taken when using these output signals. The exact meanings of the signals can frequently be changed by configuring the sensor. This is analogous to the situation with the iSight where we changed the meaning of the STANDBY signal.

An all-in-one design where the image sensor is integrated with the microcontroller which communicates to the host computer is likely to have fewer options for a secure design. A dedicated output pin which could drive an indicator LED would suffice. However, hardware designers are typically loathe to dedicate pins to specific functions, instead a variety of functions tend to be multiplexed over a single pin.

It is likely that, even in this case, there would be a separate power connection for the CMOS sensor. As with the two-chip design above, the LED driver circuit and a power supply circuit could be driven by a GPIO.

B. Secure firmware

Although using one of the secure indicator designs described above will ensure the LED will turn on when the camera turns on, it does nothing to protect against reprogramming attacks.

For this, we make four concrete recommendations which, taken together, can secure the firmware on the camera. These

apply more generally to any peripheral or embedded system connected to a host computer.

- 1) Store the firmware in nonvolatile storage on the camera module. Most commercial off-the-self (COTS) microcontrollers contain some amount of nonvolatile storage, such as flash memory, to hold firmware.⁸ By programming the firmware at the factory, one avoids the possibility that the legitimate firmware will be replaced by an attacker on the host system before being downloaded to the microcontroller.

Depending on the specific requirements of the system, the factory programming could be the complete firmware or a secure loader designed to load cryptographically signed firmware from the host (see below).

- 2) Use a microcontroller which can block unwanted firmware reprogramming attempts. It is essential that trusted code running on the microcontroller is able to block reprogramming attempts for illegitimate firmware.
- 3) Firmware updates, if necessary, should be cryptographically signed and the signature verified before applying the update. This requires both nonvolatile storage for the code to verify the signature and a microcontroller which can block reprogramming attempts. Since microcontrollers are typically resource constrained devices, choosing an appropriate signature scheme which can be implemented within the constraints is important. Scheme selection is outside the scope of this paper but we note that recent microcontrollers have started to contain specialized crypto instructions which can reduce code size and increase efficiency. For example, Rohde et al. [49] use specialized AES instructions in some Atmel ATxmega microcontrollers to implement the Merkle signature scheme.
- 4) Require root/administrator privileges to send reprogramming requests. Strictly as a matter of defense in depth, software running on the host system should restrict reprogramming attempts. Thus, even if the hardware and firmware defenses prove inadequate, this added layer of protection can still defend against some attacks.

Adding this sort of restriction typically involves a device-specific kernel module (our *iSightDefender* is an example). This may be more difficult for plug and play devices expected to conform to standard protocols and interact with generic drivers such as USB video class (UVC) or USB human interface device (HID) class devices.

The inability of the EZ-USB to block reprogramming attempts indicates that this widely-used microcontroller is inappropriate for use in any system where security is a consideration.

⁸Microcontrollers without nonvolatile storage can be paired with external nonvolatile storage, such as flash or an EEPROM, to the same effect.

IX. DISCUSSION

Although some webcams, such as the Logitech QuickCam Pro 9000, come with an explicit “LED control” that can disable the LED [57], such controls are not the norm and, in fact, are a very bad idea from both a security and a privacy standpoint. Giving the user the ability to disable a privacy feature is tantamount to giving malware the same capability.

A particularly unsavory element of the hacker culture, “ratters,” install malware bundled with remote administration tools (RATs) on victims’ computers. There are several popular RATs, including Blackshades and DarkComet, which come with a variety of “features” such as keyloggers, the ability to install additional malware, and the ability to record video and sound using the webcam. Rats are often installed with the goal of spying on women.

RATs and the ratters who use them have recently come under public scrutiny after Miss Teen USA Cassidy Wolf’s webcam was used by ratter Jared Abrahams to capture her naked images without her knowledge [15]. Abrahams arrest and guilty plea came on the heels of an *ars technica* exposé on ratters [1].

A commonly asked question on forums devoted to ridding, such as the *Hack Forums* “Remote Administrator Tools” forum, is how can one disable the webcam’s LED. In one representative thread, started by forum user “Phisher Cat” [44], the poster asks

So as all of you know, newer laptops have a light when a laptop webcam turns on, and so this scares the slave.

Is it theoretically possible for a RAT to disable this light?

disturbingly referring to his victim as “the slave,” as is common in this subcommunity. The first response by “Jabaar” notes that “[p]eople have been trying to figure this out for a very long time. The light won’t be able to be disabled as it is built into the hardware.” Others agree: “Capital Steez” writes that there is “no way to disable it,”⁹ and “FBI™” concurs “there [i]s no way to do” it. Still others suggest using social engineering in an attempt to convince the victim that the LED is normal, for example, “Orochimarū” writes, “You can’t physically turn it off but you **can** use social engineering to fool them. Maybe send an error or warning msgbox that says ‘Camera is now updating, please do not disturb’ or something.” There are many such threads on Hack Forums alone, all expressing similar sentiments: disabling LEDs is a capability the ratters *really* want to have but do not think is possible.

Unfortunately, the implications of surreptitiously capturing video do not end with privacy violations like school officials and ratters spying on people. As part of the general trend of growing frustration with passwords as an authentication mechanism, some companies are moving to biometric identification; in particular, using facial recognition on video taken with

⁹With what appears to be a complete lack of empathy for his own hapless victims, “Capital Steez” further comments, “If I was using my laptop at night and that light randomly turned on I would freak the fuck out” [44]. Indeed.

webcams. For example, BioID is software-as-a-service which provides biometric identification to online service providers using a webcam [8]. Luxand’s *FaceSDK* is a cross-platform software development kit that uses the webcam to identify the user [32].

In principle, this sort of facial recognition is trivially defeated by providing the expected picture or video to the software performing the authentication. Malware that can capture video of the victim can replay that video to authenticate to the given service. This is not a new attack. The Android *Face Unlock* system was defeated shortly after being released by holding a picture of the face in front of the camera [9]. Duc and Minh [16] describes weaknesses of several facial recognition authentication systems in the presence of pictures.

By disabling the indicator LED before capturing video, the victims have no way of knowing that their accounts may be compromised.

Although the ability to disable the LED can lead to serious privacy and security problems, there are at least two legitimate use cases. The first is that some people really do not want the LED on while they are recording. We do not find this to be a compelling use as the benefit does not seem to outweigh the potential cost; however, others may value this more than we do.

The second use case is significantly more compelling: laptop recovery. For example, the OS X version of *Adeona* software captures pictures using the laptop’s internal iSight to aid in recovery of a laptop that has been stolen by taking a picture of the thief [47, 48]. The *LANrev* software used in the Lower Merion School District incident discussed in the introduction had a similar “Theft Track” feature which is how the school officials were able to obtain pictures of students. For this use, one does not want the thief to know he is being observed.

X. RESPONSIBLE DISCLOSURE

The authors followed responsible disclosure practices by disclosing the LED disabling vulnerability to Apple product security team on July 16, 2013 and the virtual machine escape on August 1, 2013. The disclosures included the source code for *iSeeYou* and the virtual machine escape as well as directions for mounting both attacks. Apple employees followed up several times but did not inform us of any possible mitigation plans. The *iSightDefender* code was also provided to Apple and is now publicly available.¹⁰

XI. CONCLUSIONS AND FUTURE WORK

We have shown that being able to reprogram the iSight from user space is a powerful capability. Coupled with the hardware design flaw that allows the indicator LED hardware interlocks to be bypassed, malware is able to covertly capture video, either for spying purposes or as part of a broader scheme to break facial recognition authentication. Although the *iSightDefender* defense described in Section VII raises the barrier for malware, including RATs, to take control of the

¹⁰<https://github.com/stevecheckoway/iSightDefender>

camera without being detected by requiring root privileges, the correct way to prevent disabling the LED is a hardware solution.

In this paper, we have examined only a single generation of webcams produced by a single manufacturer. In future work, we plan to expand the scope of our investigation to include newer Apple webcams (such as their most recent high-definition FaceTime cameras) as well as webcams installed in other popular laptop brands.

The virtual machine escape described in Section VI demonstrates the danger that reprogrammable peripheral devices such as keyboards and mice pose. We plan to undertake a much broader examination of these devices in an attempt to understand the security implications of connecting one device to a computer which can, under attacker control, pretend to be a wide range of devices. One particularly promising direction is to study how drivers react to malformed or malicious responses from devices. In the worst case, a user space program could reprogram a peripheral device which in turn exploits a poorly written driver to inject code into the kernel.

REFERENCES

- [1] N. Anderson, "Meet the men who spy on women through their webcams," *ars technica*, Mar. 2013. Online: <http://arstechnica.com/tech-policy/2013/03/rat-breeders-meet-the-men-who-spy-on-women-through-their-webcams/>
- [2] *App Sandbox Design Guide*, Apple Inc., Mar. 2013. Online: <http://developer.apple.com/library/mac/#documentation/Security/Conceptual/AppSandboxDesignGuide/AboutAppSandbox/AboutAppSandbox.html>
- [3] *I/O Kit Fundamentals: Driver and Device Matching*, Apple Inc., May 2007. Online: <https://developer.apple.com/library/mac/#documentation/devicedrivers/conceptual/IOKitFundamentals/Matching/Matching.html>
- [4] *QTKit Framework Reference*, Apple Inc., Feb. 2009. Online: http://developer.apple.com/library/mac/#documentation/QuickTime/Reference/QTKit/QTKit/objckit/_index.html
- [5] Ballard Spahr, "Lower Merion School District forensics analysis: Initial LANrev system findings," May 2010, redacted. Online: <http://www.scribd.com/doc/30891576/LMSD-Initial-LANrev-System-findings>
- [6] —, "Report of independent investigation: Regarding remote monitoring of student laptop computers by the Lower Merion School District," May 2010. Online: http://www.social-engineer.org/resources/100503_ballard_spahr_report.pdf
- [7] E. Bersac. iSight Firmware Tools. Oct. 2009. Online: <https://launchpad.net/isight-firmware-tools>
- [8] BioID, Inc. The easy, secure way to log in and manage online identities and accounts. 2013. Last accessed: 2013-08-06. Online: <http://mybioid.com/index.php?id=67>
- [9] M. Brian, "Android 4.0 Face Unlock feature defeated using a photo," *The Next Web*, Nov. 2011. Online: <http://thenextweb.com/google/2011/11/11/android-4-0-face-unlock-feature-defeated-using-a-photo-video/>
- [10] camJAMR.com. 2012. Last accessed: 2013-08-07. Online: <http://store.camjamr.com/shop-now/camjamr-webcam-covers.html>
- [11] S. Checkoway, D. McCoy, D. Anderson, B. Kantor, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *Proceedings of USENIX Security 2011*, D. Wagner, Ed. USENIX, Aug. 2011. Online: <http://www.autosec.org/pubs/cars-usenixsec2011.pdf>
- [12] CoreLabs, Core Security Technologies. Apple OS X Sandbox predefined profiles bypass. Nov. 2011. Online: <http://www.coresecurity.com/content/apple-osx-sandbox-bypass>
- [13] *EZ-USB[®] Technical Reference Manual*, Cypress Semiconductor Corporation, 2011. Online: <http://www.cypress.com/?docID=27095&d1m=1>
- [14] Cypress Semiconductor Corporation. CY3684 EZ-USB FX2LP Development Kit. 2013. Online: <http://www.cypress.com/?rID=14321>
- [15] A. Dobuzinskis, "California man agrees to plead guilty to extortion of Miss Teen USA," *Reuters*, Oct. 2013. Online: <http://www.reuters.com/article/2013/10/31/us-usa-missteen-extortion-idUSBRE99U1G520131031>
- [16] N. M. Duc and B. Q. Minh, "Your face is NOT your password: Face authentication bypassing Lenovo – Asus – Toshiba," Presented at BlackHat Briefings, Jul. 2009. Online: <https://www.blackhat.com/html/bh-usa-09/bh-us-09-main.html>
- [17] L. Dufлот and Y.-A. Perez, "Can you still trust your network card?" Presented at CanSecWest 2010, Mar. 2010. Online: <http://www.ssi.gouv.fr/IMG/pdf/csw-trustnetworkcard.pdf>
- [18] L. Dufлот, Y.-A. Perez, and B. Morin, "What if you can't trust your network card?" in *Proceedings of RAID 2011*, R. Sommer, D. Balzarotti, and G. Maier, Eds.

- Springer, Sep. 2011, pp. 378–397. Online: <http://www.ssi.gouv.fr/IMG/pdf/paper.pdf>
- [19] M. Elkins, “Hacking with hardware: Introducing the universal RF USB keyboard emulation device: URFUKED,” Presented at DefCon 18, Aug. 2010. Online: <http://www.youtube.com/watch?v=EayD3V77dI4>
- [20] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, “TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones,” in *Proceedings of OSDI 2010*, R. Arpaci-Dusseau and B. Chen, Eds. USENIX, Oct. 2010. Online: http://static.usenix.org/events/osdi10/tech/full_papers/Enck.pdf
- [21] D. Farmer, “IPMI: Freight train to hell,” Jan. 2013. Online: <http://fish2.com/ipmi/itrain.pdf>
- [22] C. Gibler, J. Crussell, J. Erickson, and H. Chen, “AndroidLeaks: Automatically detecting potential privacy leaks in Android applications on a large scale,” in *Trust and Trustworthy Computing*, ser. Lecture Notes in Computer Science, S. Katzenbeisser, E. Weippl, L. J. Camp, M. Volkamer, M. Reiter, and X. Zhang, Eds. Springer Berlin Heidelberg, 2012, vol. 7344, pp. 291–307.
- [23] J. Grossman. Clickjacking: Web pages can see and hear you. Oct. 2008. Online: <http://jeremiahgrossman.blogspot.com/2008/10/clickjacking-web-pages-can-see-and-hear.html>
- [24] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall, “‘These aren’t the droids you’re looking for’: Retrofitting Android to protect data from imperious applications,” in *Proceedings of CCS 2011*, G. Danezis and V. Shmatikov, Eds. ACM Press, Oct. 2011. Online: <https://research.microsoft.com/pubs/149596/AppFence.pdf>
- [25] J. Howell and S. Schechter, “What you see is what they get: Protecting users from unwanted use of microphones, cameras, and other sensors,” in *Proceedings of W2SP 2010*, C. Jackson, Ed. IEEE Computer Society, May 2010. Online: <https://research.microsoft.com/pubs/131132/devices-camera-ready.pdf>
- [26] iFixit. MacBook Pro Retina Display teardown. 2013. Online: <http://www.ifixit.com/Teardown/MacBook+Pro+Retina+Display+Teardown/9493>
- [27] S. T. Inc. Meteor Mic - USB Studio Microphone. 2013. Online: <http://www.samsontech.com/samson/products/microphones/usb-microphones/meteormic/>
- [28] D. Kennedy and J. Kelley, “PowerShell omfg...” Presented at DefCon 18, Aug. 2010. Online: http://www.youtube.com/watch?v=eWoAGxh0a_Q
- [29] J. Kim, Y. Yoon, K. Yi, and J. Shin, “ScanDal: Stack analyzer for detecting privacy leaks in android applications,” in *Proceedings of MOST 2013*, H. Chen and L. Koved, Eds. IEEE Computer Society, May 2013. Online: <http://www.mostconf.org/2012/papers/26.pdf>
- [30] N. S. Kovach, “Accelerating malware detection via a graphics processing unit,” Master’s thesis, Air Force Institute of Technology, Sep. 2010.
- [31] E. Ladakis, L. Koromilas, G. Vasiliadis, M. Polychronakis, and S. Ioannidis, “You can type, but you can’t hide: A stealthy GPU-based keylogger,” in *Proceedings of EuroSec 2013*, T. Holz and S. Ioannidis, Eds. ACM, Apr. 2013. Online: <http://www.cs.columbia.edu/~mikepo/papers/gpukeylogger.eurosec13.pdf>
- [32] Luxland, Inc. Detect and recognize faces with Luxand FaceSDK. 2013. Last accessed: 2013-08-06. Online: http://luxand.com/facesdk/index_c.php
- [33] Mactaris. Webcam Settings 2.0 will support FaceTime HD camera on MacBook Air 2013. Jul. 2013. Online: <http://mactaris.blogspot.com/2013/07/webcam-settings-20-will-support.html>
- [34] P. Marquardt, A. Verma, H. Carter, and P. Traynor, “(sp)iPhone: Decoding vibrations from nearby keyboards using mobile phone accelerometers,” in *Proceedings of CCS 2011*, G. Danezis and V. Shmatikov, Eds. ACM Press, Oct. 2011. Online: <http://www.cc.gatech.edu/~traynor/papers/traynor-ccs11.pdf>
- [35] *1/6-Inch SOC VGA CMOS Digital Image Sensor: MT9V11212ASTC*, Micron Technology, Inc., 2005. Online: <http://download.micron.com/pdf/datasheets/imaging/MT9V112.pdf>
- [36] C. Miller, “Battery firmware hacking: Inside the innards of a smart battery,” Presented at Black Hat Briefings, Aug. 2011. Online: http://media.blackhat.com/bh-us-11/Miller/BH_US_11_Miller_Battery_Firmware_Public_WP.pdf
- [37] —, “Owning the fanboys: Hacking Mac OS X,” Presented at Black Hat Japan Briefings, Oct. 2008. Online: <https://www.blackhat.com/presentations/bh-jp-08/bh-jp-08-Miller/BlackHat-Japan-08-Miller-Hacking-OSX.pdf>
- [38] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and

- R. R. Choudhury, "TapPrints: Your finger taps have fingerprints," in *Proceedings of MobiSys 2012*, S. Seshan and L. Zhong, Eds. ACM Press, Jun. 2012. Online: <http://synrg.ee.duke.edu/papers/tapprints-final.pdf>
- [39] H. Moore. A penetration tester's guide to IPMI and BMCs. Jul. 2013. Online: <https://community.rapid7.com/community/metasploit/blog/2013/07/02/a-penetration-testers-guide-to-ipmi>
- [40] M. T. Moore, "Pa. school district's webcam surveillance focus of suit," *USA Today*, May 2010. Online: http://usatoday30.usatoday.com/tech/news/surveillance/2010-05-02-school-spy_N.htm
- [41] A. O'Donnell. How to secure your webcam in less than 2 seconds. Mar. 2011. Online: <http://netsecurity.about.com/b/2011/03/25/how-to-secure-your-webcam-in-less-than-2-seconds.htm>
- [42] *Oracle VM VirtualBox*®, Oracle Corporation, 2013. Online: <http://www.virtualbox.org/manual/>
- [43] E. Owusu, J. Han, S. Das, A. Perrig, and J. Zhang, "ACcessory: Password inference using accelerometers on smartphones," in *Proceedings of HotMobile 2012*, R. K. Balan, Ed. ACM Press, Feb. 2012. Online: <http://www.hotmobile.org/2012/papers/HotMobile12-final42.pdf>
- [44] Phisher Cat. Webcam light scaring slaves. . . . Jul. 2013. Registration required. Last accessed 2013-08-06. Online: <http://www.hackforums.net/showthread.php?tid=3660650>
- [45] J. Pisani, P. Carugati, and R. Rushing, "USB-HID hacker interface design," Presented at BlackHat Briefings, Jul. 2010. Online: <https://media.blackhat.com/bh-us-10/presentations/Rushing/BlackHat-USA-2010-Rushing-USB-HID-slides.pdf>
- [46] R. Riedmüller, M. M. Seeger, H. Baier, C. Busch, and S. D. Wolthusen, "Constraints on autonomous use of standard GPU components for asynchronous observations and intrusion detection," in *Proceedings of IWSCN 2010*, A. Brunstrom and S. J. Knapskog, Eds. IEEE Computer Society, May 2010. Online: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5497999
- [47] T. Ristenpart, G. Maganis, A. Krishnamurthy, and T. Kohno, "Privacy-preserving location tracking of lost or stolen devices: Cryptographic techniques and replacing trusted third parties with DHTs," in *Proceedings of USENIX Security 2008*, P. Van Oorschot, Ed. USENIX, Jul. 2008. Online: <http://adeona.cs.washington.edu/papers/adeona-usenixsecurity08.pdf>
- [48] ——. Adeona. Oct. 2009. Online: <http://adeona.cs.washington.edu/index.html>
- [49] S. Rohde, T. Eisenbarth, E. Dahmen, J. Buchmann, and C. Paar, "Fast hash-based signatures on constrained devices," in *Proceedings of CARDIS 2008*, G. Grimaud and F. ois Xavier Standaert, Eds. Springer, Sep. 2008. Online: <https://www.hgi.rub.de/hgi/publikationen/fast-hash-based-signatures-constrained-devices/>
- [50] R. Schlegel, K. Zhang, X. Zhou, M. Intwala, A. Kapadia, and X. Wang, "Soundcomber: A stealthy and context-aware sound trojan for smartphones," in *Proceedings of NDSS 2011*, A. Perrig, Ed. Internet Society, Feb. 2011. Online: <http://www.cs.indiana.edu/~kapadia/papers/soundcomber-ndss11.pdf>
- [51] R. Templeman, Z. Rahman, D. Crandall, and A. Kapadia, "PlaceRaider: Virtual theft in physical spaces with smartphones," in *Proceedings of NDSS 2013*, P. Ning, Ed. Internet Society, Feb. 2013. Online: http://www.internetsociety.org/sites/default/files/02_2_0.pdf
- [52] A. Tereshkin and R. Wojtczuk, "Introducing ring –3 rootkits," Presented at Black Hat Briefings, Jul. 2009. Online: <http://www.blackhat.com/presentations/bh-usa-09/TERESHKIN/BHUSA09-Tereshkin-Ring3Rootkit-SLIDES.pdf>
- [53] The iPatch. 2011. Last accessed: 2013-08-07. Online: <http://www.theipatch.com>
- [54] G. Vasiliadis, M. Polychronakis, and S. Ioannidis, "GPU-assisted malware," in *Proceedings of MALWARE 2010*, J.-Y. Marion, N. Rathaas, and C. Zhou, Eds. IEEE Computer Society, Oct. 2010, pp. 1–6. Online: <http://dcs.ics.forth.gr/Activities/papers/gpumalware.malware10.pdf>
- [55] Vimicro, *VC0358 USB 2.0 Camera Processor*, Sep. 2012. Online: http://www.vimicro.com/english/product/pdf/Vimicro_VC0358_PB_V1.1.pdf
- [56] Z. Wang and A. Stavrou, "Exploiting smart-phone USB connectivity for fun and profit," in *Proceedings of ACSAC 2010*, P. Traynor and K. Butler, Eds. ACM Press, Dec. 2010, pp. 357–366. Online: <http://dl.acm.org/citation.cfm?id=1920314>
- [57] ZTech, "QuickCam Pro 9000 LED Control," Post on the Logitech Forums, Feb. 2008, last accessed: 2013-08-06. Online: <http://forums.logitech.com/t5/Webcams/QuickCam-Pro-9000-LED-Control/td-p/186301>