

IT'S IMPOSSIBLE TO PROVE YOUR LAPTOP HASN'T BEEN HACKED. I SPENT TWO YEARS FINDING OUT.

Micah Lee

April 28 2018, 11:00 a.m.

Photo: Micah Lee

Digital security specialists like me get some version of this question all the time: “I think my laptop may have been infected with malware. Can you check?”

We dread this sort of query because modern computer exploits are as complex, clever, and hard to reason about as modern computers – particularly if someone has the ability to physically access your device, as is routinely the case with laptops, especially when traveling. So while it’s definitely possible to detect certain types of tampering, it isn’t always trivial. And even in controlled environments, it’s impossible to give a laptop a clean bill of health with full confidence – it’s always possible that it was tampered with in a way you did not think to check.

The issue of tampering is particularly relevant for human rights workers, activists, journalists, and software developers, all of whom hold sensitive data sought by powerful potential attackers. People in these vocations are often keenly aware of the security of their laptops while traveling – after all, laptops store critical secrets like communication with sources, lists of contacts, password databases, and encryption keys used to vouch for source code you write, or to give you access to remote servers.

Join Our Newsletter
Original reporting. Fearless journalism. Delivered to you.
I'm in →

How safe is it to leave your laptop in your hotel room while you're attending sessions at a conference? If you come back to find your laptop in a different position than where you thought you left it, can you still trust it? Did someone tamper with it, did a hotel housekeeper simply straighten up the items you left on your desk, or did you misremember where you left it?

These questions typically can't be answered with total confidence because clever tampering can be so hard to detect. But I hoped I could get a sense of the risks with a carefully controlled experiment. For the last two years, I have carried a "honeypot" laptop with me every time I've traveled; this computer was intended to attract (and then detect) tampering. If any hackers, state-sponsored or otherwise, wanted to hack me by physically messing with my computer, I wanted to not only catch them in the act, but also gather technical evidence that I could use to learn how their attack worked and, hopefully, who the attacker was.

While traveling by air, I checked this laptop in my luggage to make it easily accessible to border agents, both domestic and foreign, to tamper with if they chose to. When staying in hotels, I left the laptop sitting on the desk in my room while I was away during the day, to make sure that any malicious housekeepers with permission to enter my room, or anyone else who broke into my room, was free to tamper with it if they chose to. I also put a bunch of hacker stickers all over it, hoping that this would make it a more enticing target.

Over the duration of this experiment, I traveled to Europe three times and domestically in the United States five times (including once to Puerto Rico). I found eight different notices from the Transportation Security Administration informing me that my baggage had been searched. I have no way of knowing how many times it had been searched by other authorities who weren't kind enough to leave me a note.

I never caught anyone tampering with this laptop. But the absence of any evidence of tampering – and my obsessive thoughts about the various ways an attacker could have evaded by detection – serve to underline how fraught the process of computer forensics can be. If someone who makes their living securing computers thinks they could have missed a computer infection, what hope is there for the average computer user?

At the end of my experiment, I thought through all of the things that could have gone wrong. Perhaps someone did tamper with my honeypot laptop, and my methodology for detecting this wasn't thorough enough to notice. Or maybe potential attackers noticed that the laptop I carried with me and used at the conferences I was attending was different than the one I left in my room, and decided against tampering with it in case it was a trap.

But the most likely reason I didn't catch any attackers is that no one tried to tamper with my laptop. Hacking a target's laptop by physically tampering with it while they're traveling probably happens only rarely because it's so expensive – it may require travel, physical surveillance, breaking and entering, and the risk of getting caught or breaking the laptop is high. Compare this to cheap forms of hacking like email phishing: You can target thousands of people at once from the comfort of your office, and the risk of getting caught is much lower.

Still, I believe actively checking devices for tampering is worthwhile. You'll never catch an attacker in the act if you never look for evidence of their attacks. And just looking for evidence, even if you don't find any, increases costs for attackers: If they want to be sure you won't notice, they're going to have to get more creative. I believe it's useful to explain the technology and the methodology I came up with to detect tampering and share what I learned from the experience. Doing so gives a taste of just how many ways there are to tamper with a laptop.

Photos of the honeypot laptop on the desk in several different hotel rooms: Micah Lee

Evil Maid Attacks

If you don't [use full disk encryption](#) on your laptop, anyone who gains physical access to it, even for just a few minutes, can access all of your data and even implant malware on your computer to spy on you in the future. It doesn't matter how good your password is because without encryption, the attacker can simply unscrew the case on

your laptop, remove your hard disk, and access it from another computer.

Disk encryption does a great job of protecting your data in case you lose your laptop or someone steals it from you. When this person tries accessing your data, they should be completely locked out, so long as the passphrase you use to unlock your laptop is **strong enough** that they can't guess it.

But there is a sneaky class of attack, called “evil maid” attacks, that disk encryption alone cannot protect against. Evil maid attacks work like this: An attacker (such as a malicious hotel housekeeper, for example) gains temporary access to your encrypted laptop. Although they can't decrypt your data, they can spend a few minutes tampering with your laptop and then leave it exactly where they found it. When you come back and type in your credentials, *now* you have been hacked.

Exactly how an evil maid attack would work against your laptop depends on many factors: the type of computer you use, what operating system you use, which disk encryption software you use, and the configuration of firmware used to boot your computer, firmware which I'll call “BIOS,” although it can also go by acronyms like EFI and UEFI. Some computers have considerably better technology to prevent evil maid attacks than others – for example, attackers have to do more advanced tampering to hack a Windows laptop encrypted with BitLocker than they do to hack a Mac laptop encrypted with FileVault (**as of now**, anyway) or a Linux laptop encrypted with LUKS.

The honeypot laptop I used, with red boxes around the hard disk and the SPI flash chip that stores the BIOS firmware.
Photo: Micah Lee

Here are the main ways that an attacker could physically tamper with your laptop:

An attacker could modify data on your hard disk. “Full disk encryption,” the term used to refer generically to systems like FileVault, really ought to be called “nearly full disk encryption” because, except in a few specific circumstances, there's always a small part of a computer's disk that isn't encrypted.

When you power on your laptop, before your disk has been unlocked, your computer

loads a program from this unencrypted part of your disk; it then runs the program, and the program asks you to enter your passphrase. The program converts your passphrase into an encryption key and tries to use it to unlock the disk. If you typed the correct passphrase, the disk unlocks, and the rest of the operating system (which is stored in the encrypted part of the disk) boots up. If you don't know the right passphrase, there is no way to unlock the disk.

But since the program that asks for your passphrase isn't encrypted, it's possible for an attacker that physically has your laptop to replace it with a malicious version that looks exactly the same to the user, but that takes extra steps. For example, after you successfully unlock your disk, it might copy malware onto it that, after the computer finishes booting up, automatically runs in the background, spying on what you're doing.

Computers that support "secure boot" or "verified boot," such as Chromebooks and Windows laptops with BitLocker, aren't vulnerable to this. The BIOS can detect if the unencrypted part of your disk has been tampered with, and if it has, it will refuse to boot. MacBooks and laptops that run Linux could potentially be attacked in this way.

An attacker could replace your BIOS firmware with malicious firmware. When you power on your computer, the very first program that your computer runs is your BIOS firmware. The job of this program is to initialize all of your hardware – your memory, disks, Wi-Fi adapter, video card, USB ports, and everything else – and then ultimately boot an operating system, typically the one stored on your hard disk.

When you format your disk and install a new operating system on your computer, your BIOS firmware doesn't change. This is because this program isn't stored on your hard disk at all. Instead, it's stored in a small chip on your computer's motherboard called an SPI flash chip.

This is why BIOS malware is so stealthy – you can't get rid of it by formatting your hard disk, and it can spy on you across operating systems, such as if you boot to a [Tails](#) USB stick.

SPI flash chips have eight pins, including one for providing the chip with power, one for reading data, and one for writing data to the chip. This means that it's possible for an attacker to power off your laptop, open up the case, and attach their own wires to the SPI flash chip pins in order to power it on, and then read and write data to it (the chip itself has no way of telling the difference between this, or just being part of the normal computer). Using this technique, an attacker with physical access to your laptop can replace your BIOS firmware with whatever malware they want.

The Italian spyware firm Hacking Team was caught [selling such BIOS malware](#) to its customers (the company's clients include [foreign governments with troubling human rights records](#)). This specific firmware made sure that Windows was always infected with malware. If you're a target of a Hacking Team customer, even formatting your disk and re-installing Windows would not remove the malware. As soon as you reboot, the malicious BIOS firmware would re-infect the freshly installed Windows with the same malware again.

Trying to dump BIOS firmware directly from the SPI flash chip by wiring it to a BeagleBone Black, a small and cheap external computer. Photo: Micah Lee

An attacker could do other things to your hardware. Tampering with unencrypted data on your hard disk, or replacing your BIOS firmware with malware, are the most straightforward types of evil maid attacks, but the list of other potential attacks is only limited by the attacker's creativity and budget.

Here are a few examples:

- An attacker could potentially figure out a way of spying on your computer use by replacing firmware on other components of your computer besides your BIOS, like your processor, video card, network card, or hard disk.
- An attacker could install a hardware keylogger (they would plug your internal keyboard into the keylogger, then plug the keylogger into the motherboard) with the intention of stealing your laptop later, but with a record of your disk passphrase and your other keystrokes.
- An attacker could completely replace your laptop with a different laptop of the same model – they could even put your real laptop case with all your stickers and scratches on the fake laptop, so that it looks exactly the same. When you type your passphrase into this one though, it could send that passphrase over the internet to the attacker, who could then use it to unlock your real disk.

When I decided to start this honeypot laptop project, I realized early on that I couldn't possibly detect every form of tampering. Because tampering with the data on the hard disk or the BIOS firmware are the simplest and cheapest types of evil maid attacks to conduct, and because attackers have limited resources and prefer low-hanging fruit when it's available, I decided to limit my detection to these two components. But who knows? It's possible that my honeypot laptop has a malicious component in it that I never checked for.

Re-assembling the computer. Photo: Micah Lee

Methodology

In February 2016, shortly before I was planning to fly to Spain for the [Internet Freedom Festival](#), I bought a Lenovo IdeaPad S210 Touch for about \$700 to use as my honeypot laptop.

Here was the plan. Before each trip, I would:

- Update all of the software on my honeypot laptop. (I wanted potential attackers to see that I'm using up-to-date bootloader software stored on the small unencrypted part of my disk, to believe that I actively use this computer.)
- Power off the laptop, and don't power it on again until the trip was over. (Just by powering on the computer, I risk slightly modifying data in my BIOS and hard disk.)
- Remove the hard disk from the laptop, attach it to an external USB enclosure, and plug it into another computer, taking care not to modify any data on it. From there, I could make a record of the state of the disk.
- Attach a [BeagleBone Black](#), or BBB, a tiny \$50 computer that's great for hardware hacking, to the SPI flash chip on the motherboard and use it to dump the BIOS firmware, saving an exact copy of the data stored on the chip.
- Re-assemble the computer.

Then, during the trip, I would:

- Put the honeypot laptop in my checked luggage.
- Leave the honeypot laptop unattended in my hotel room.

Once I returned home from a trip, I would:

- Remove the hard disk from the laptop, plug it into another computer, and record the state of the disk again. If even a single bit of data on the disk has changed, I could detect it.
- Attach the BBB to the SPI flash chip and dump the BIOS firmware again. I could then compare the firmware image I took from before my trip with the image I took after

it, to detect if it was tampered with.

Along the way, I planned to document everything: I'd take photos of my laptop in my luggage, in hotels, and of the cards that TSA leaves informing me that they searched my luggage; I'd keep a log of the state of my hard disk and BIOS before and after each trip; and I'd keep a journal that includes all technical hurdles I ran into.

This is *mostly* how it went down, but I did run into a few snags. Bear with me, in the following sections I venture much deeper into the technical weeds than I already have.

Checking the Hard Disk for Tampering

Before I proceed, there a few concepts I need to explain.

- Computer drives, whether disk- or flash- based, are organized into separate “partitions.” For example, if you install Linux with disk encryption, your drive will likely have two partitions: a small (often less than 1 gigabyte) unencrypted partition called “/boot” – this is where the program that asks for your encryption passphrase is stored, and where evil maids might put their malware – and the rest of the disk will contain a large encrypted partition. After you unlock the encrypted partition with the right passphrase, it will likely contain two other partitions inside, a “/” or “root” partition that holds the rest of the files on the computer, and a “swap” partition that’s only used when the computer is running low on memory. (I refer to both flash- and disk- based storage drives as “disks” and “hard disks.”)
- There is a small amount of disk space at the beginning of every hard disk (which I call the disk header) that’s reserved for a bootloader program. When you power on a computer and boot from your hard disk, you run this program. In Linux, this program simply starts running another program called “grub” that’s stored in your unencrypted “/boot” partition. Grub is responsible for actually booting your operating system. In order to detect evil maid attacks, it’s important to make sure the disk header was not tampered with.
- In cryptography, a “hash function” is a one-way function that takes an input of any size and outputs a fixed-size result called a “hash” or “checksum.” For example, with SHA256, the hash function I use in this project, whether your input is 5 bytes long (the size of the word “hello”) or 512 gigabytes long (the size of a hard disk), the output will always be 32 bytes long. The same input will always lead to the same output, but if you change the input in *any way*, even if only a single byte is different, the content of the output will be entirely different (although the length will be the same). You can use checksums to detect tampering.

When I started this project, I decided to dual-boot Windows 10 and [Debian](#), a popular Linux distribution, on my honeypot laptop – that is, I installed both operating systems in different partitions on the same disk, and when I powered on the computer, I got to choose which to boot to. But due to various time-consuming and annoying issues related to Windows updates, I eventually chose to abandon Windows altogether and just run Debian on my honeypot laptop, which made my job of detecting hard disk tampering simpler.

Before each trip, I removed the disk from the honeypot laptop, plugged it into a USB enclosure, and then plugged the USB disk enclosure into a different computer. The first snag I ran into was that when I plugged in the USB disk, my computer automatically tried to mount the partitions. This isn’t good – just by mounting the partitions, I risked modifying the data. So I changed the settings on my computer (which was running

Linux) to disable automatically mounting external drives by following [these instructions](#).

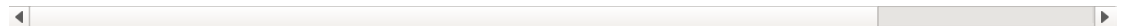
Hard disk from the honeypot laptop, removed and plugged into a cheap external USB enclosure that can be plugged into a separate computer. Photo: Micah Lee

Once the USB disk enclosure was attached to a computer other than the honeypot, I could generate checksums of all of the disk's partitions, as well as the disk header, using a tool called [sha256sum](#). In order to take a checksum of just the disk header, I used a tool called [dd](#) to copy the disk header into a file, and then used sha256sum to take a checksum of that file.

When I returned home from my trip, I repeated the same process to generate a new set of checksums. Finally, I compared the checksums from before my trip with the checksums from after my trip. If the checksums were not the same, then the data on the disk must have changed – this would be evidence of tampering. If I discovered this, I could then start looking into exactly what data had changed to discover how the tampering worked.

For example, in March 2017, before flying to Amsterdam to attend a [Tor Project](#) meeting with developers, volunteers, and advocates for the open-source anonymity network, these were the checksums I generated:

```
4040239f4f0a2090c3ca15216b6e42522c4c3cd291f2c78f3c9e815f25be8295 c
ed6e8a3438e55d2aeae4ae691823c4005f7b5df0b62d856bd72d54fa00d886bb /
db3d92ed1cfa8621e5673da32100d9117a3835c06a613cf9ac0f2f90de404d17 €
cbeb585b6fa39a8425f57fa095ac17353a583bccd93532d65d9274da628a4c72 1
```



Ten days later, after I had returned from my trip, I generated another set of checksums. They all turned out to be exactly the same, which allowed me to confirm that the data on my hard disk had not changed at all.

Checking the BIOS for Tampering

When I started this project, I intended to dump BIOS firmware images externally using a BeagleBone Black (explained previously) and a software tool called [flashrom](#), which is used for reading and writing data that's stored on physical chips on circuit boards, roughly following the [instructions outlined here](#). But I quickly hit a snag, one that I didn't have the tools or knowledge of electronics to easily resolve.

The SPI flash chip that holds my honeypot laptop's BIOS firmware has a pin for power and another pin for ground. With the laptop powered off, I connected the power and ground pins to my BBB, and then powered on the BBB.

I had hoped that the BBB would provide power to the SPI flash chip, allowing me to read and write directly to that chip. But instead the BBB immediately powered off. It turns out that, the way this specific laptop was wired, the power for the SPI chip was not isolated from the rest of the system. In order to provide power to that chip, I also needed to provide power to rest of the components of the motherboard, and that takes more watts than my BBB was able to handle. This was annoying because one of the reasons I chose a Lenovo computer as my honeypot laptop is because I have had success doing this exact process on other Lenovo computers in the past, dumping the BIOS firmware by connecting a BBB to the SPI flash chip.

So I decided to change strategies. Instead of dumping the BIOS firmware by connecting wires directly from the SPI flash chip, I would instead use a piece of software called [chipsec](#), running on the honeypot laptop itself, to dump the firmware. However, this strategy has a few downsides compared to directly connecting to the chip:

- In order to run chipsec, I needed to first power on the honeypot laptop and boot to an operating system. It turns out that this process, booting up the computer, slightly modifies the data stored in the BIOS firmware, which makes it more difficult to check for tampering.
- It's impossible to get a complete BIOS dump from within an operating system, but you can get most of it.
- When I dump BIOS firmware using chipsec, it may be possible for sophisticated BIOS malware to lie to chipsec, which could be used to prevent detection. (I have never heard of BIOS malware that actually does this, though.)

In order to use chipsec, I set up a USB stick with the operating system [Ubuntu](#) (another popular Linux distribution). With the hard disk removed from the honeypot laptop, I plugged in my Ubuntu USB stick, powered on the laptop, and booted to Ubuntu. I put a copy of chipsec on an SD memory card, which I also plugged into the laptop. From there, I was able to run a specific chipsec command to dump the BIOS firmware and save it to the SD card, which I could then inspect on my other computer.

Dumping the BIOS firmware using chipsec.

Here is the [VirusTotal report](#) from the first BIOS firmware image that I dumped using chipsec from my honeypot laptop.

Once I successfully managed to dump the BIOS firmware, I came up with this plan:

- Before each trip, I would remove the hard disk from the laptop, boot to the Ubuntu USB stick, and dump the BIOS firmware, making sure to save a copy of it on my other computer.
- After I return from the trip, I would repeat the process, dumping a fresh BIOS firmware image.
- Then I would generate checksums of the BIOS firmware images from before and after my trip. If the checksums were exactly the same, I could confirm that my BIOS was not tampered with.

Of course, it wasn't this simple. It turns out, *every time* I booted my honeypot laptop to an Ubuntu live USB stick and dumped the BIOS firmware, that firmware image had a different checksum than the previous one. In order to investigate what was going on, I used a program called [UEFITool](#). This is a graphical program that lets you load BIOS images, view and edit what data is stored inside, and extract data into separate files.

Inspecting BIOS firmware using UEFITool.

For this specific laptop, each BIOS firmware image is exactly 4 megabytes. Some of that space is used to store the actual programs that make up the BIOS (an evil maid attacker would replace these programs with malicious versions), and some of it is used to store other data, such as saved BIOS settings.

Looking at two BIOS firmware images that had different checksums, I was able to use UEFITool to extract the same components from both, and then generate new checksums for those individual components to see if they matched. I discovered that there was only one small part of the firmware images that differed, and that part did not include any programs. It turns out, each time I powered on the honeypot laptop and opened the boot menu to tell it to boot from my Ubuntu USB stick, it saved information related to booting to a USB stick in that section of the firmware, and this information was slightly different each time, which caused the firmware images to always have different checksums.

So I amended my plan for detecting tampering in the BIOS firmware. To compare firmware images from before and after my trip, I would have to open each image in UEFITool, extract all of the components except for the one that I knew changed, generate checksums for those components, and then compare *those* checksums to make sure they matched.

What I Learned

Traveling with a honeypot laptop was a lot of work. It required spending a few hours both before and after each trip if I hoped to actually catch an evil maid attacker in the act. So after two years without catching anyone, I have decided to retire the project.

A tool exists today, that didn't exist when I started the project, that makes it possible to catch evil maid attackers in the act in a different way. [Haven](#) is an Android app,

designed to run on a spare phone that you leave in your hotel room while you're away, perhaps sitting on top of your laptop. It uses all of its sensors – microphone, motion detector, light detector, and cameras – to monitor the room for changes, logs everything it notices, and can send Signal notifications to the phone you carry with you when it detects a change. Haven isn't perfect – there are plenty of false positives – but it gets better all the time and is still likely to catch anyone attempting to tamper with the laptop that's sitting under the phone Haven is running on.

I was able to do this entire project with 100% free and open source software, thanks to projects like Debian and Ubuntu and tools like dd, sha256sum, flashrom, chipsec, and UEFITool. Other than the honeypot laptop itself, you can buy all of the hardware tools I used, like screwdrivers, a USB enclosure, and a BeagleBone Black, for less than \$100.

Top photo: Honeypot laptop in the luggage.

 We depend on the support of readers like you to help keep our nonprofit newsroom strong and independent. [Join Us](#) →

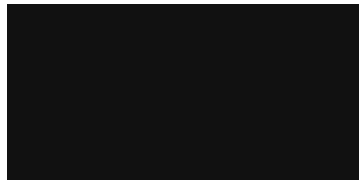
RELATED



Edward Snowden's New App Uses Your Smartphone to Physically Guard Your Laptop



How to Protect Yourself Against Spearphishing: A Comic Explanation



How to Use Signal Without Giving Out Your Phone Number



How Right-Wing Extremists Stalk, Dox, and Harass Their Enemies