by Limor “Ladyada” Fried (ladyada@alum.mit.edu) and Phillip Torrone (fill@2600.com)

“Preventing IoT Device Attacks”

Attack surface reduction” is a security principle that you can use to guide your choices when designing an IoT product or service. The attack surface of a hardware or software environment is all of the different points where an unauthorized user can try to insert or extract data. Keeping the attack surface as small as possible is an essential but necessary security measure. Since devices like the ESP8266 and others have come along, anyone can be an IoT device developer for about $5.

With IoT, there are at least two attack surfaces. The thing itself, say an Internet-connected temperature sensor, and the service - whether Google Cloud, Microsoft Azure, or Amazon AWS, etc. Since web service security has been discussed a ton in 2600 Magazine and other publications, let’s go over device security, from the easiest first.

These “ten things” are not everything you’ll have to worry about, but it’s a good start, and if you do these, you’re ahead of 99 percent of IoT vendors.

#1 Require login and password. This is number one because it’s the bare minimum. Don’t have an open, network-accessible interface to your IoT device. You may think “oh nobody is going to guess the URL or the port number” but that’s the first thing attacks probe. Even if it’s on an intranet, require some authentication!

#2 Don’t have default logins and passwords. We mentioned this before, but it bears repeating because it’s so common! Make sure your device has a unique, unguessable password by default.

#3 Two-factor authentication. In addition to a username and password, maybe have an SMS or time-based second factor. 2FA will protect you even if the password is sniffed or stolen. 2FA is free and pretty easy to implement these days - you no longer have to distribute a physical token, since most everyone has a mobile phone.

#4 Require TLS/SSL. Whenever your users or devices connect to the Internet, whether over Wi-Fi or cellular, use the latest available version of TLS, sometimes called SSL or HTTPS. TLS will encrypt all data transmitting between the device and the service, protecting both. TLS will significantly reduce your risk of sniffing. A few years ago, microcontrollers were older and smaller and couldn’t effectively run a TLS stack. Nowadays, there’s no excuse to skip it.

#4.5 Authenticate Host Certificates. TLS is not just data encryption; it’s also server authentication. So, if you’re using TLS, make sure your device is checking the fingerprint or certificate chain of the server. We’ve seen some TLS implementations where it’s possible to skip this, which makes man-in-the-middle attacks possible.

#5 Turn off any unused services. If you have an embedded Linux or RTOS for your device, make sure no services are running. File sharing, remote login, mail servers, etc. These days, most services are not enabled by default, but check anyway. Sometimes these are left on during development and are forgotten when the firmware is released.

#5.5 Don’t accept any inbound connections. If you can, don’t allow any way for outside parties to connect into the device. If you have a debugging port left open, that’s just another attackable surface.

#6 Require physical access for important configurations. We’ve seen some Wi-Fi cameras that can be controlled over the Internet, but if you want to change the access point password, you need to plug it into a computer and change the setting over USB. This reduces the surface that can be attacked by automated scripts.

#7 Individualized/Revocable Authentication Keys. For your device to connect to the service, chances are it has some authentication key or password. Make sure that you have a unique key or password for each device - even if the user never sees these, you shouldn’t reuse them. You’ll
also need to have a way to revoke/re-instantiate keys if they’re lost, corrupted, or stolen.

#8 Data Paranoia. Even though you may only be shuffling data from your IoT device to your IoT service, don’t trust that the data is well-formatted. This is often forgotten in a rush to complete and ship firmware, but you should assume that attackers will try to send corrupted or malformed chunks of data to both sides of the connection to corrupt memory. Clean up and vet data thoroughly; this will also keep your device running smoothly if the network connection is flaky.

#8.5 Updatable Firmware. Bootloaders are the best, and it’s a good idea to have one on your device. Many are write-only so that the deployed firmware can’t be read. Being able to update firmware will help customers recover the device if it gets bricked, hacked, or if there’s a critical security update. We like USB bootloaders the best, or ones where you insert an SD card with a file. Having updatable firmware increases your attack surface a bit because it opens another access point into your device, but we think that if someone has physical access, they could connect a JTAG programmer to erase and reprogram it anyways.

#9 Secure storage for authentication keys. Embedded Linux devices have a regular filesystem, and microcontrollers often store their code in flash memory, so even if your hard-code authentication keys in flash or EEPROM, it can be read out. Yes, even if you have a chip that has firmware-readback turned off, it’s possible to glitch chips into revealing their secrets. Your microcontroller memory should not be considered secure storage! Instead, you may want to consider using a secure element chip. These chips are designed to withstand common decapping and glitching attacks and can be programmed with the private key at your factory. Then, it never leaves the secure chip. Instead of having the key sit in microcontroller memory where it could be read out, data that needs to be authenticated or encrypted is sent back and forth through I2C. It’s a little extra cost, but it is an excellent way to keep the secrets in a lock-box.

#10 Over the air updates. This one is a little tricky. Not having OTA is risky because then there’s no way to send important security updates. On the other hand, having OTA is dangerous because it allows an attacker to take over the device completely. We think OTA is a good idea, but you need to combine it with the prior rules - firmware must be transmitted over an authenticated, encrypted connection. Having firmware be signed with public-key cryptography (so the private key is not stored on the device) is a common idea, but be aware that private keys can leak out, so that should not be the only way you verify the firmware is valid.

We’ve seen more than one company accidentally “brick” their devices with a mistaken OTA - some even required a physical recall - so if you do have OTA updates, make sure you always have a way for physical-access-rollback.

For both your IoT device and service - if it has a web interface, it should be protected against standard hacking techniques like remote code execution, path traversal, cross-site request forgery, and SQL injection. There are scanning services you can run against the website as well as on the code itself to find egregious errors.

Good night and good luck.