SI5/M2II - IoT Security Lab
September 26th, 2022
Yves Roudier - UCA / Polytech Nice Sophia

This lab aims at giving you a hands-on experience with IoT Security and Privacy concerns and technologies. We will be looking at several vulnerability databases and ways to find vulnerable IoT systems. We will see how to retrieve information from a vulnerable firmware. We will finally experiment with QR-codes.

1) IoT vulnerabilities

You will be looking at diverse databases that compile IoT vulnerabilities and compare the information that they contain.

1.1) National Vulnerability Database

The National Vulnerability Database (or NVD) is a well-known repository for vulnerabilities discovered in software and firmware, as discovered by security experts. For instance, a quick search may point you to CVE-2019-11061 (https://nvd.nist.gov/vuln/detail/CVE-2019-11061), one such vulnerability on Smart Home IoT systems. This page does not only explain how the vulnerability works, but it also its severity and points to patches and or corrective actions.

You will find the full listing of vulnerabilities at https://nvd.nist.gov/vuln/full-listing , and interesting statistics on this site, like for instance the number of vulnerabilities involving the "IoT" keyword in their description over the years (https://nvd.nist.gov/vuln/search/statistics?form_type=Basic&results_type=statistics&query=IoT&search_type=all).

1.2) Google Dorks Database

The Google search engine periodically indexes a vast amount of the Internet web pages and web services. By doing so, it also records many services available to everyone, voluntarily or because of a misconfiguration or careless deployment, typically with standard passwords or no security. Dorks are searches performed using the many metadata keywords offered by the Google search engine.

For instance, you may search for cameras feeds available over the web over port 8081 using the following dork: *intitle:"cam" inurl:8081.* Beware! Accessing some of these webcams may be offensive or illegal (meteo webcams are your best bet if you want to check the result as they should normally be accessible to the general public).

Similarly, the following dork: *intitle:"LaserJet" "Device status" "Supplies summary"* should return online HP Laserjet accessible online.

You can experiment with other dorks you can find through … the Google search engine, of course!

1.3) Shodan

Go to the Shodan website (https://www.shodan.io) and create an account. After connecting to the website, have a look through the different interfaces offered. The main one is the search bar, which allows to sift through the numerous data collected through crawling the web and all the services supporting online applications and IoT systems. Shodan especially collects open ports and responses obtained from these services.

As an example, you might search for servers supporting the MQ Telemetry Protocol (MQTT), a protocol used by many IoT services. Now try to search for *port:1883 "MQTT"* in order to locate the default MQTT broker port that does not use the TLS secure transport. Have a look at how many servers could be found by Shodan.

2) Firmware Analysis: Initial Reconnaissance

In this section, we will reproduce the initial reconnaissance part of a firmware analysis process / pentesting over an IoT device (artifact), the TP-Link HS100 Smart Power Plug. It is very important to gather as much information as possible on the target system.

This plug has been analyzed in several different studies :

- A firmware and protocol analysis at the following blog page: https://www.softscheck.com/en/reverse-engineering-tp-link-hs110/
- A client and wireshark dissector for the protocol are provided here: https://github.com/softScheck/tplink-smartplug by Lubomir Stroetman, the author of this study
- The more protocol oriented study discussed during the lecture: https://troopers.de/downloads/troopers17/TR17_fgont_-iot_tp_link_hacking.pdf
- A more advanced python client for the plug was published here: https://python-kasa.readthedocs.io/en/latest/_modules/kasa/smartdevice.html
- Even more libraries have been developed as people are trying to integrate the plug into IoT frameworks.

At the same time, TP-Link has been releasing new versions of the firmware, that have suppressed the most blatant security flaws and that have finally even introduced automatic updates, which will force devices that can openly communicate with the Internet to update their firmware when TP-Link decides so, without any notice, and with side effects for some people (see for instance: https://www.reddit.com/r/TPLinkKasa/comments/eiiqz5/firmware_update_causes_hs100_smart_plug_to_toggle/). This will sometimes be breaking the integration with existing packages (see for instance: https://plugins.octoprint.org/plugins/tplinksmartplug/, https://github.com/softScheck/tplink-smartplug/issues/81 or https://github.com/plasticrake/homebridge-tplink-smarthome/issues/154). In addition, firmware updates sometimes fail (https://lightrun.com/answers/softscheck-tplink-smartplug-firmware-versions-).

We will be using Kali Linux and its tools to have a look at some of the steps discussed in the first study. Working from a virtual machine is perfectly fine.

To get started, you will first of all download the following firmware to your Kali box from: https://www.dropbox.com/s/ngdcewv3s8ho0c4/HS110%28US%29_V1_151016.zip?dl=0.

TP-Link still makes it possible to download this firmware from their US web site: https://www.tp-link.com/us/support/download/hs110/v1/#Firmware

We will unfortunately not have a real device available as it is no longer produced by TP-Link it is weaker version. You might however play with other firmware versions from the following page: https://firmware.hddsurgery.com/?manufacturer=Samsung&family=Mango . You might also try your luck at the Austrian repository of TP-link for firmware related tools – you will however need to resort to Wayback Machine as this website is no longer open for the general public: https://web.archive.org/web/20180317021214/http://tp-link-support.de/software/

Next, you will need to install an additional tool to be able to extract the filesystem from its compressed file, as explained in the original blog page. You will have to install sasquatch (https://github.com/devttys0/sasquatch), a set of patches to the unsquashfs utility as detailed here: https://blog.pavel-pi.de/micro/sasquatch-install/

You are now ready to take a sneak peek at the firmware. Start by following the directions of the blog. We will obviously not be able to run tests and do protocol captures. We will instead focus on the information that can be extracted from the firmware.

When you are done with the intial steps (firmware extraction, password cracking with John the Ripper for instance), examine the shd executable with rafind2, a module of the reverse engineering tool radare2 available from your Kali VM.

Try to find out whether you can locate existing vulnerabilities from the libraries or the application and otherwise interesting pieces of information.

If you want to further try decompiling the code, you may start radare2 using command:
r2 -a mips shd (or use the tool of your choice).

If you additionally want to run the shd code in debug mode, you now need to install Qemu in order to emulate the processor.

3) QR Code Authentication

In this section, we will experiment with QR-Codes and how they can be used for authentication in IoT scenarios. What can you logically place in a QR-code in order to securely authenticate an IoT artifact? What are the constraints that you have to satisfy?
First of all, have a look at https://goqr.me/.
Create your own text and try do decrypt it from your phone.
Now download the QR-code (or take a screenshot) – PNG is suggested.
Write a simple program to decrypt it using the Python libraries pyqrcode and pyzbar.
You may also try to create one such QR-code with pyqrcode or with the alternative Python library qrcode.
Try to create malicious QR-Codes, for example through the TEC-IT code generator at https://qrcode.tec-it.com/en/Raw for texting or calling a number (no guarantee it will be processed automatically by your phone, but it should be at least one click/touch away).
Finally try to create first a public key, then a certificate with OpenSSL and to embed them in a QR-Code. Try different key sizes.