Payatu

Internet of Things Security Assessment - Success Stories

TABLE OF CONTENTS

IOT - HEALTHCARE

The device under testing was a portable IoT enabled wireless detector for digital radiography. It had built-in Wi-Fi and Near Field Communication (NFC) for data transfer and configuration. It had X-ray auto-detection, image storage, and multi-sharing features, among others.

Software updates on the device were performed over the network. There is an Ethernet port in the device, which can be used to access upgrades. The goal of the assessment was to verify the security of firmware upgrade, Wi-Fi, and other communication interfaces.

FINDINGS

We primarily focused on the communication interfaces as the scope did not include disassembly and hardware testing. Next to the Ethernet port, there were a couple of metal pads, which can be either a debug port or a communication line. Further inspection revealed that NFC was not used for data exchange. The device has a forum type-2 tag embedded for detecting it when connected to a casing. The metal pads turned out to be insignificant.

The operation of the device has been described below:
When the detector comes in contact with an IR transceiver controlled via the PC application, the application transfers Wi-Fi credentials and calibration data to the detector over the IR interface. On completion of the transfer, the detector restarts and connects to the configured network. Firmware upgrades can be done over a Wi-Fi network or via Ethernet cable with the companion application.

We decided to start the testing process from IR communication. At this time, we were not aware of the protocol used. We conducted passive sniffing and analyzed the data transfer. The protocol did not provide much information other than that the device used IR. After trying with different IR transceiver modules, one specific transceiver model delivered proper data. Spending some time with the raw data, we were able to figure out the packet structure, and eventually, the configuration data formatted as XML. The configuration is passed as plaintext that includes the Wi-Fi SSID and password. This can be abused by an attacker to gain access to the internal network.

From the way the device handled the commands, there's a possibility of command injection. We tried to send some commands instead of eavesdropping. The XML formatted payload sent to the detector was executed successfully.

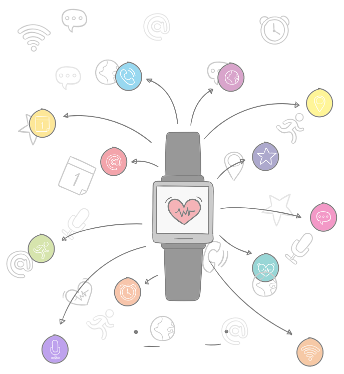
Companion application can be used to upgrade the firmware over a network interface (Ethernet or Wi-Fi). While analyzing the upgrade process, we observed that the same data was transferred multiple times across the network. On extracting and analyzing the data, it was found to be the device firmware.

We needed to find out the cause behind the so many transfers of the firmware. Further study revealed the update process flow. When the companion application requests for a firmware upgrade, the device copies existing firmware from the external memory to the ram and transfers it to the app over the network as a failsafe. The companion application sends the new firmware to the device, and the device flashes it to the external memory. Once the flashing is over, the device reads back and sends the firmware to the application.
The application then computes the checksum and compares it with the original checksum that was calculated before flashing. In conclusion, it is a very inefficient, insecure, and broken firmware upgrade mechanism.

The commands can be replayed to trigger a firmware upgrade.

By analyzing the firmware, we recovered the user credentials. The credentials were reused in all the firmware versions and hardware revisions.

IOT – WEARABLES



Wearable technology is getting much attention lately. Smart devices are continually improving and are used in healthcare, remote monitoring, patient tracking, and industrial safety, among others. The device under testing was a smart band, capable of tracking heart rate, activities, calories, and sleep. The band came with a companion mobile application and a cloud component.
The current project comprised of testing only the hardware and IoT sub-components that directly affect the hardware, such as firmware management.

FINDINGS

All the communication to the band is through the mobile application. This includes sending notifications, syncing user data, firmware upgrades, and others. We were particularly interested in the firmware upgrade mechanism. The firmware is downloaded from a cloud endpoint, authenticated, and encrypted in transit. The application then pushes the firmware to the smart band over BLE. Next, we need to identify who might be decrypting the firmware and the key-store.

We found that the mobile application decrypts the firmware and stores it in plain text on the phone, the application logs, and the decryption key as well. We recovered the device firmware from the phone.

The mobile app decrypts the firmware locally and saves the decrypted plain text in phone storage, and the decryption key is also printed in the mobile app log. We can recover the unencrypted firmware from the app.

Reversing the firmware revealed the BLE characteristics and the features implemented in the band. We exploited this data further to hijack the BLE communication. We were able to send notifications to the band, recover user data, and trigger a firmware upgrade.

Disassembling the band and analyzing the PCB revealed the microcontroller and radio chips used. We tried to look for JTAG or SWD debug port in the MCU. It is an enormous advantage to have a debug port while trying dynamic analysis. We were able to identify the debug port after spending some time, but the readout protection was enabled, and it prevented dynamic analysis.

IOT – WIRELESS

The device under testing was a wireless dictation microphone. The docking station that came with it acted as a wireless charger and paired with the dictation device to enable PC connectivity. Our goal was to identify and mitigate any vulnerabilities that could affect the hardware and wireless communication.

FINDINGS

We tried to get as much information about the wireless technology used. The support documents were very obscure. We observed the normal workflow a few times and tried monitoring the communication with SDR. We did not find any interesting results.

The next step was to disassemble the device and identify the chipsets. We found a lesser-known proprietary radio chip handling the wireless link. Further study about the chip showed that the radio chip was capable of sending USB HID commands, data, and audio. It was controlled by an arm microcontroller.

There was no public documentation on how the authentication works between the dictation device and dock. At any given time, a dictation device can only pair with a single dock. On trying pairing the device with a different dock, the previous connection terminated. Reconnecting to the previous dock was possible by keeping the device tapped to the dock for 5-6 seconds.

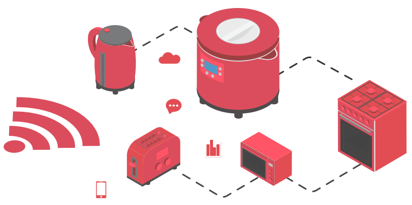
The PCB contained an I2C EEPROM. Extracting the content revealed some data. However, the data looked random and contained no strings or any descriptive information. We decided to extract the EEPROM content during the pairing process and found that the logic of the device was reversed. Whenever a dock pairs, 3 bytes are written to the EEPROM at a fixed offset. When a second dock pairs with the device, the 3 bytes get replaced, and all the remaining content is retained. We subsequently replaced the bytes with a different one recovered while pairing the device to another dock. The device immediately connected to the second dock remotely, without tapping it to the dock.

The HID functionality provided in the device can be abused by an attacker to gain remote access to the PC to which the dock is connected.

The firmware upgrade mechanism was our next target. The companion app triggered a firmware upgrade if a dock was attached to the PC. The dock wirelessly forwarded the incoming firmware from the PC to the device. The PC to dock firmware transfer was not encrypted. We recovered the firmware, monitoring the USB packets during an upgrade.

IOT - CONSUMER ELECTRONICS

The IoT device under testing was a smart kitchen appliance that can download recipes from the internet and cook. It is capable of steaming, emulsifying, blending, precise heating, mixing, milling, whipping, kneading, chopping, weighing, grinding, and stirring. The goal was to assess the overall security of the device.



FINDINGS

Initially, the security architecture of the device appeared well thought. High assurance boot was enabled to prevent the execution of unsigned code, and firmware was encrypted in transit as well. However, on spending some time with the device and observing the normal operation, the firmware was being downloaded onto an external USB storage device as a compressed file in zip format during an upgrade. We could not recover the firmware because the zip file only contained encrypted firmware.

There was still a possible attack vector if the device used an outdated zip utility, which, in our case, was true. By carefully crafting a zip file to overwrite the password file on decryption, we were able to change the admin credentials of the device. This bug can be exploited to execute several attacks.

The next step was to disassemble the device and inspect it further. The device had two sections – a front-end that interacts with the user and does all the 'smart' things and a back-end that communicates with the hardware. The main processor was in the front-end. On identifying the components in the PCB, it was observed that there is no direct way to fully lock the JTAG port of the main processor, and the processor manuals did not specify anything. Unless the device manufacturer worked closely with the chip vendor to figure out a solution, JTAG is the next step to follow.

The PCB has test points everywhere. Automating brute-forcing the pins would have taken quite some time. As the main processor comes in a vast BGA package, no conductivity testing was possible. We superimposed the test points, BGA balls, and PCB images to narrow down the JTAG pins, and we did find out the JTAG port.

The JTAG port was open, and read-write access to the external RAM, internal SRAM, and the CPU registers was gained. We were able to run malicious code in the device over JTAG, which bypassed protection provided by the high assurance boot.

Finally, we uploaded the code to read the external NAND flash through JTAG and extracted the device firmware. There was a timeout mechanism that restarted the front-end during the idle state of the processor, which made the extraction difficult.

Over the Universal Asynchronous Receive and Transmit lines, the front-end is always communicating with the back-end. All the commands were in plaintext, and the device was not maintaining a session or ID for each transaction.

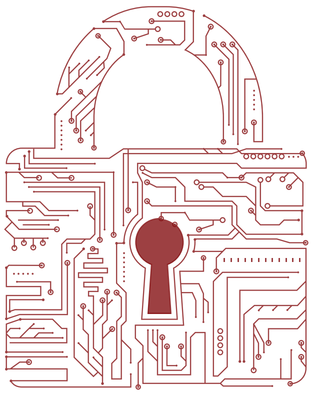
A command was repeatedly being sent to the back-end to reset the timeout counter, which prevented the back-end from resetting. We built a hardware implant that sent a reset command repeatedly and inserted it in-between the front-end and back-end UART lines. This removed the timeout mechanism.

Once the firmware is out, it was easy to figure out the inner workings of the device and to recover critical device information.

INTERNET-CONNECTED SECURITY SYSTEMS

Smart security systems have been an integral part of the building management system because of the boom in the IoT industry due to affordable rates. This is one case to prove the vulnerability of a Smart security system.

We got a request to perform a security assessment of a Smart Security system. To be precise, it was a smart doorbell with a camera and other sensors connected to it.



FINDINGS

We performed a full end-to-end security analysis of the device, ranging from the cloud to hardware. We did find a lot of common developer mistakes in the mobile like hardcoded encryption key and other low impact vulnerabilities. Upon analyzing some logs sent to an open s3 bucket, it was discovered that a lot of critical information was being leaked, such as device ID, Wi-Fi password, and some firmware update requests. This made us want to gain access to the hardware. We found a hardware debug port in the device and gained access to a custom console that lets you check the health of the device and other device-specific information. In one of the options, we found an entry point to command injection and managed to change the login password of the root and gained access to the root shell. Once you get the root shell, the device is compromised. We reversed a few binaries and were able to decrypt some information that was saved in the device. We were able to retrieve all the videos and sensor details from the device, upload fake videos to the server, and so on. However, these issues were fixed very soon after the assessment.