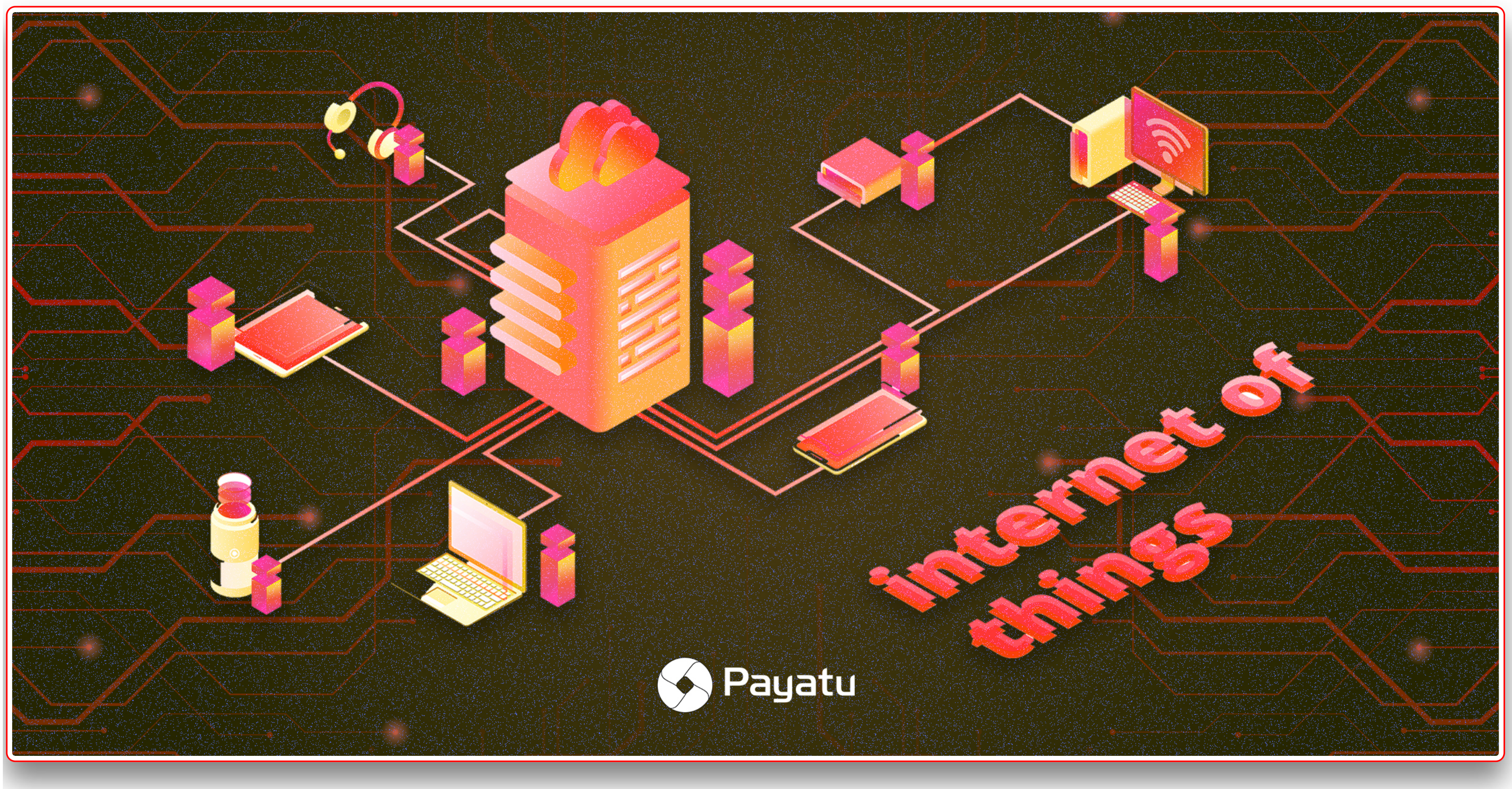
IoT Security – Part 2 (101 – IoT Attack Surface)

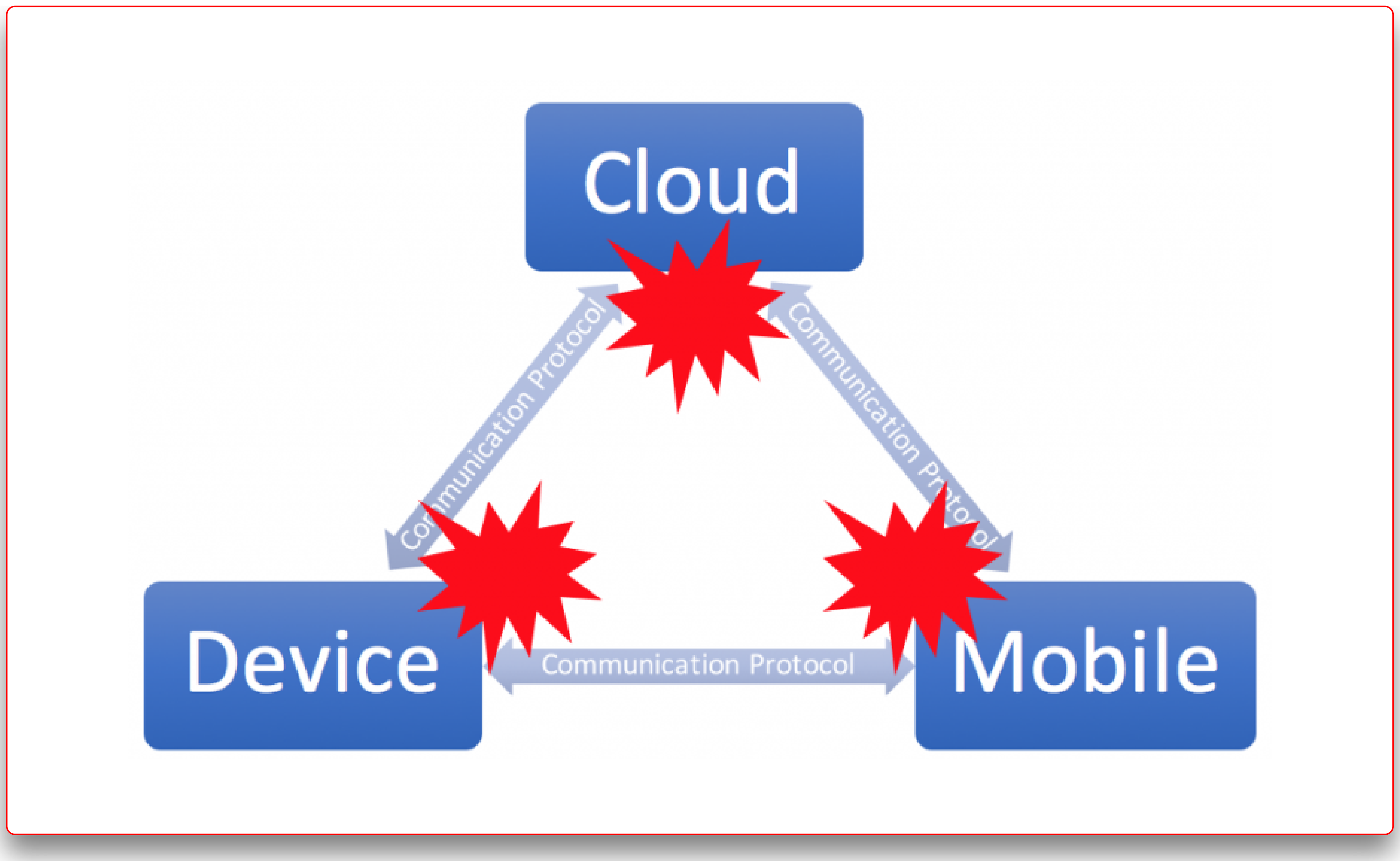
Admin-Payatu
08/09/2017



Internet of Things Attack surface

Welcome! I hope you have gone through the previous blog post "IoT Security – Part 1" If not, I would urge you to go through it to understand the meaning of IoT and IoT architecture. Now we will start getting into security and try to define a way to understand and create a structured process to perform security research or penetration testing of IoT.
If we look at the architecture defined in the previous post, it now becomes clear and easy for us to segregate the components of IoT and try to define the attack surface for each one of them individually and then combine them to create a holistic overview of the IoT ecosystem attack surface. I call it IoT ecosystem instead of IoT product because it indeed is an ecosystem of different components talking to each other and solving a particular real world problem. Let's go ahead and define the attack surface of IoT ecosystem and discuss each component's attack surface in detail. The attack surface by components can be divided into three or four(if we include communication as an attack surface) major areas as follows:

1. Mobile
2. Cloud
3. Communication
4. Device



OWASP is also doing a lot of work in IoT security now. They have also defined the attack surface. I would again urge you to go through it. It is good to understand different ideas and thoughts as it helps you create your own comprehensive attack surface. OWASP is also doing a lot of work in IoT security now. They have also defined the attack surface. I would again urge you to go through it. It is good to understand different ideas and thoughts as it helps you create your own comprehensive attack surface.

Note

1. The word microcontroller is used in a generic form to mean microcontrollers, microprocessors or SoC (System on a Chip) unless specifically mentioned with explanation.

2. The below attack surface is defined by us and may be different from other sources.

Mobile

Mobile is one of the important user interface for IoT via which end users get insights into the state of the physical world. Since mobile app communicates with the IoT ecosystem to send commands and read data, it becomes one of the entry point into the IoT ecosystem. We will try to list down the attack surface for the mobile from IoT perspective

1. Storage
2. Authentication
3. Encryption
4. Communication
5. Generic mobile vulnerabilities – OWASP Mobile Top 10 comes to mind

Cloud

The cloud is one of the very important pieces of IoT as usually data from all the instances of the product line converges here. This makes it a very interesting attack point. Remember, I mentioned that IoT is not only about hardware in my previous post. The reason being that cloud will hold the data of all deployed IoT instances and has the privileges to send commands to all of them. Well, generally it happens to be user initiated, but if compromised, the attackers will gain control of the devices (and its data) deployed worldwide, which is dangerous. Overall the attack surface focuses on the interfaces that it provides which include

1. Storage
2. Authentication
3. Encryption
4. Communication
5. APIs
6. Generic Web/cloud vulnerabilities – OWASP Web Top 10 comes to mind

Device

Next is the device, which is the game changer for IoT tech :). It interfaces with the physical world an also communicates with the virtual world. It is the first stop for physical world data. There is a whole debate around user privacy given the sensitive data about the user it stores (for example home stats, body stats, personal information). In future devices may use user's crypto currencies directly through their wallet or a separate temporary wallet to purchase items, repairs etc. The attack surface looks somewhat as follows

1. Storage
2. Authentication
3. Encryption
4. Communication
5. Sensor interface
6. Peripheral interfaces
7. Hardware interfaces
8. Human machine Interface

Communication

Although this is not a tangible attack surface as ideally the tangible attack surface would be the communication interfaces and the respective drivers/firmware responsible for the communication. However, this needs a separate section on its own because there is an endless list of communication protocols that the IoT ecosystem can use on wired as well as wireless medium. The following are some of the areas that make up the attack surface for the communication.

1. Authentication
2. Encryption
3. Deviation from the protocol standard
4. Protocol implementation anomalies

The hardware interfaces allow for the actual communication. However, the actual data communication / packets are defined by the upper layers which are implemented in the software. Hence, in this Attack surface area (communication) we will only discuss the protocols. Although the flaws in the protocol may result in attacks on the protocol end points residing on the mobile, device or the cloud, we have kept it as a separate attack surface for clarity. There are way too many standards to mention in the list here. However, we will list down some of the common protocols that are used in various IoT products.

1. Web

The web or in technical terms HTTP(S) is the most common protocol used for communication and is used everywhere. We dedicate a separate entry for this as the attack surface on web is huge. However, good news is that the attack surface, vulnerabilities and mitigation techniques have mostly been standardized as it has been under research for more than two decades now. There are plenty of resources available online which describe the attacks and protection in detail. For starters OWASP has done a great job with their Web Top 10, testing guide and various open source tools (www.owasp.org)

2. Others

Apart from web there are many protocols, some domain specific, some generic and some for efficiency reasons. There are too many protocols to list down here, for brevity, we will list some of the common protocol standards to give you a fair idea about the kinds of protocols in use. History tells us that all protocols will have their share of implementation flaws, protocol design flaws and configuration flaws. These need to be analyzed during a pentest.

1. CoAP – **https://en.wikipedia.org/wiki/Constrained_Application_Protocol**
2. MQTT – **https://en.wikipedia.org/wiki/MQTT**
3. AMQP – **https://en.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol**
4. WebSocket – **https://en.wikipedia.org/wiki/WebSocket**
5. CANbus – **https://en.wikipedia.org/wiki/CAN_bus**
6. Modbus – **https://en.wikipedia.org/wiki/Modbus**
7. Profibus – **https://en.wikipedia.org/wiki/Profibus**
8. DNP3 – **https://en.wikipedia.org/wiki/DNP3**
9. BACNet – **https://en.wikipedia.org/wiki/BACnet**
10. HL7 – **https://en.wikipedia.org/wiki/Health_Level_7**
11. XMPP – **https://en.wikipedia.org/wiki/XMPP**
12. UPnP – **https://en.wikipedia.org/wiki/Universal_Plug_and_Play**
13. DNS
14. SSH
15. <Your name here> 🙂

The above should give you a high-level overview of the attack surface for the IoT ecosystem. Now that we have a fair idea about it, let's define a detailed attack surface for the device so we know what exactly we need to attack in a standard IoT pentest. This is also helpful for IoT security architects to create a threat model for the IoT Product.
Please note we are not going to (re) define the attack surface for Mobile and Cloud as you can find plenty of resources on the Internet describing the same. The idea of this blog series is to

create a bridge for security researchers to get into IoT security so, we will focus on knowledge that is not currently available or structured. Given that we will still talk about mobile and cloud security wherever it is relevant for the IoT ecosystem from our perspective

Device Attack Surface

Ok, let's do this :). The following is a segregated and structured definition for the IoT attack surface. Please note that this is as per our understanding and has not been picked up from other sources.

1. Storage

The storage used by the device. This can further be segregated into Internal and external, Persistent and volatile.

1.1 SD Card

SD cards are typically used to store configuration and product data. They might be used to store firmware updates as well. It is a very interesting attack surface and we will talk about certain attacks that are possible via the SD card in later blog posts.

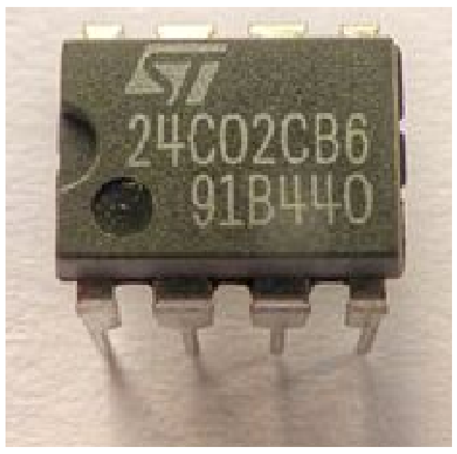
1.2 USB

Certain products may use USB drives to store similar data as in SD Cards as well as read data that is downloaded or stored on the USB drive. Similar attacks as for SD cards are applicable to the USB storage.

1.3 Non-volatile Memory

These are used for various things including read/write sensor data, bootloader, firmware, credentials, keys etc. While testing the hardware board it is crucial to look at the data stored on the chip. We can also perform run-time analysis of the communication between the memory and the microcontroller to analyze what kind of data is stored/read during different operations. This is achieved by having a Logic analyzer sniff the bus communication. You can find interesting data being read/written while triggering specific operations on the device. There are different types of memory chips as follows:

1. EPROM
2. EEPROM
3. FLASH – More commonly used due to its speed and efficiency



An I2C Serial EEPROM

1.4 Volatile Memory

When talking about volatile memory, the word RAM immediately pops up in our mind. These are widely used in PCs and embedded systems and hold the code and data at run-time. The data is lost when the device is powered Off. Some of the common RAM types are as follows

1. SRAM (Static Random Access Memory) – A type of RAM that holds the data which is lost when the chip is powered off.
2. DRAM (Dynamic Random Access Memory) – Data is held for a period after which it is lost unless it is refreshed during run-time. This means that the data has a short lifespan even during the time the chip is powered on as compared to SRAM. The data is also lost when the chip is powered off.

1.5 Microcontroller Internal Memory

Microcontrollers also have their own internal memory which is typically used to store code. These memories are usually accessible while debugging a microcontroller for example debugging via JTAG. The various memories that are used in microcontrollers are:

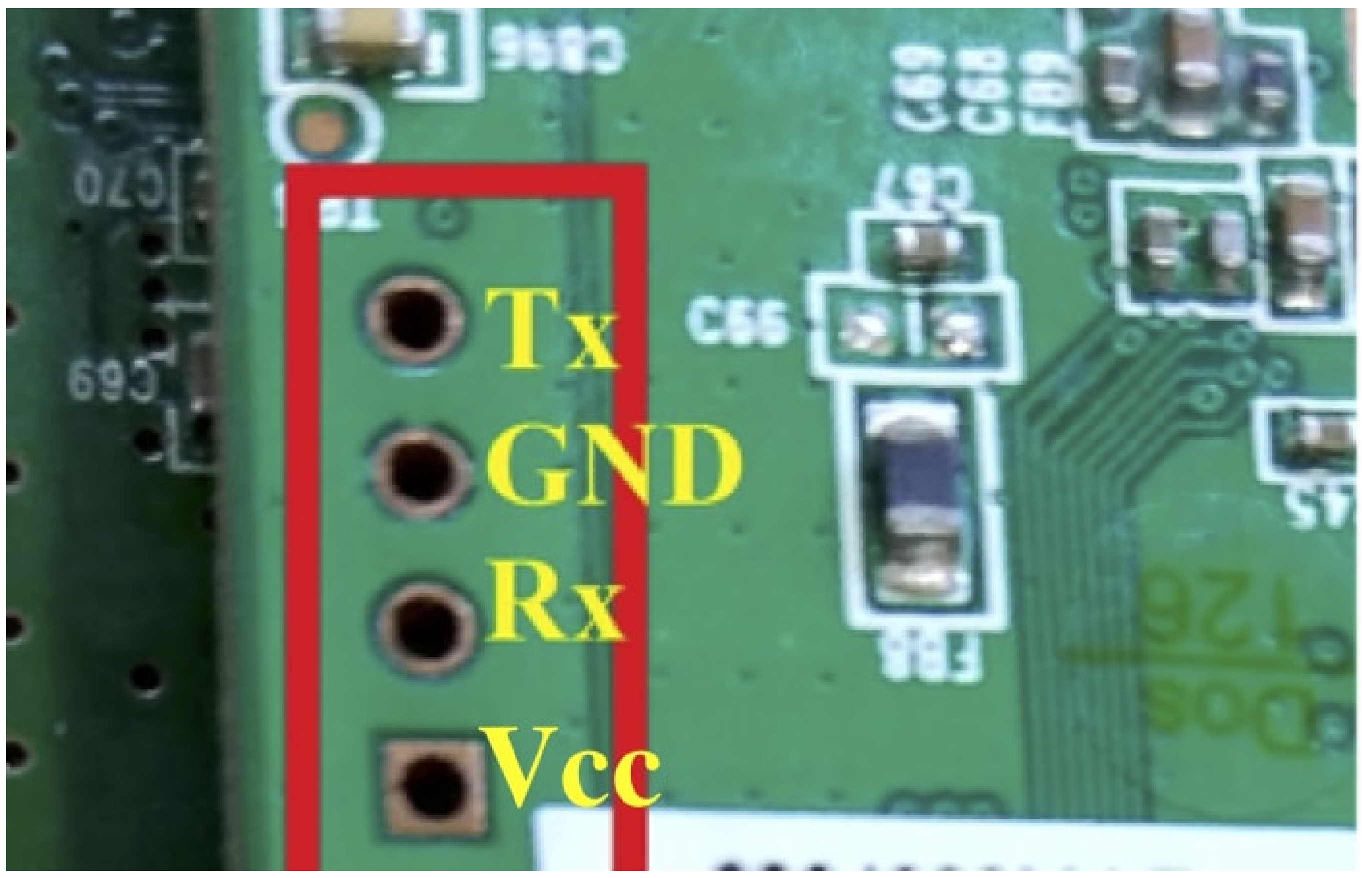
1. SRAM
2. EEPROM
3. FLASH

2. Hardware Communication Interface

Different hardware components on the same board need to talk to each other and to the outside world. All this communication is done using well defined and standard hardware communication protocols. From an attacker perspective, it gives them an insight into the actual communication via sniffing or injecting malicious data. Some of the most common interfaces mentioned below should be analyzed for finding security issues.

2.1 UART

UART (Universal Asynchronous Receiver Transmitter) is a hardware component that allows asynchronous serial communication between two hardware peripherals. These can be on the same board (for example microcontroller talking to a motor or LED screen) or between two different devices (for example device microcontroller talking to a PC). It is an interesting attack surface as it may allow read/write access to the device over serial. In many devices, UART ports on the board are left open which anyone can connect and access over serial to get a console of some sort i.e. simple shell, custom command line consoles, log output etc. A device will typically have a group of pin-outs connected to the microcontroller UART RX and TX pins, which are used for sending and receiving serial data. We will discuss in detail in the future posts how to identify and access the UART ports on a device.



A typical 4-pinout UART port

2.2 Microcontroller Debug Port

Microcontrollers have provisions for debugging during run-time using specified pins which are connected to pin-outs on the board. These pin-outs (ports) are used by the developers and designers to debug, read/write firmware and microcontroller internal memory, control/test microcontroller pins post production. This makes the debug ports one of the most critical attack surface, given the power and access it gives to the attacker. There are a few standard interfaces used for this purpose which are as follows:

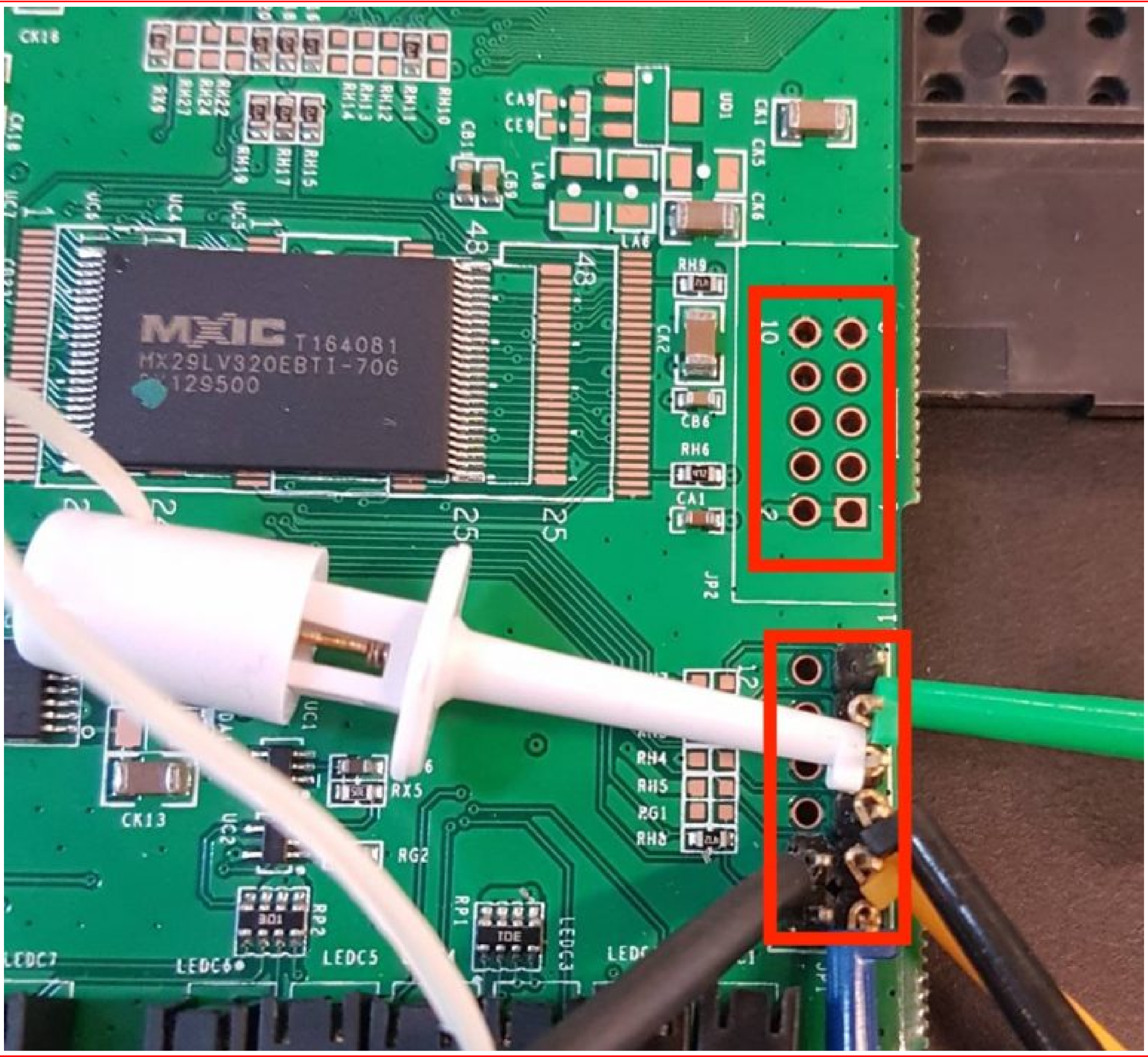
1. JTAG (Joint Test Action Group): As the microcontrollers and PCBs were becoming smaller and smaller, it was getting difficult to test them after production. So, to efficiently test the boards post production, the electronics Industry created an association with the same name and defined a method to test the Boards after production. It was later adapted as IEEE standard 1149.1. The JTAG protocol defines standard interfaces and commands that can be used to test and debug the microcontroller. JTAG defines four pin interface (and one additional optional pin TRST):

   1. TMS – Test Mode Select
   2. TCK – Test Clock
   3. TDI – Test Data In
   4. TDO – Test Data Out
   5. TRST – Test Reset (optional pin)

In addition to testing the chips, these pins are used to by the debuggers to communicate with TAP (Test Access Port) which is implemented on the microcontroller. From security perspective Identifying the JTAG port and interfacing with it allows attackers to extract firmware, reverse engineer the logic, and flash malicious firmware on the device. More on it later, in the future blog posts.

2. cJTAG (Compact JTAG): This is a new JTAG protocol defined in the standard IEEE 1149.7. It does not replace 1149.1 standard but extends it further and is backwards compatible with JTAG. It defines a two-pin interface (TCK and TMS) and a new TAP that implements the new features.

3. SWD (Serial Wire Debug): SWD is another interface/protocol that is used for debugging microcontrollers. It is a two-pin interface:a. SWDIO (bidirectional)b. SWCLK (clock) It is an ARM specific protocols that uses ARM CPU standard bi-directional wire protocol, defined in the ARM Debug Interface v5. The benefit of SWD is that it claims to be more efficient that JTAG.



How a JTAG port might look like on a PCB board

Note that JTAG port may not necessarily be in a group of 10-pinouts as in the above image.

2.3 I2C

Inter-Integrated Circuit is a short distance communication protocol used for communication between chips on the same board. It was invented by Philips (now NXP). It has a master-slave (multi) architecture and uses two-wire bus

1. SDA – Serial Data

2. SCL – Serial Clock

One of the use case of I2C is in EEPROM chips that are connected to the microcontroller I2C pins and typically store data or code. Typical attacks would include tampering with the data, extracting sensitive information, corrupting the data etc. We should analyze the data at rest on the EEPROM chip as well as perform run-time analysis by sniffing the I2C communication to understand the behavior and security implications. As mentioned previously, we will have a dedicated blog post in the series for understanding and analyzing I2C communication.

2.4 SPI

Serial Peripheral Interface is also a short distance communication protocol used for communication between chips on the same board. It was developed by Motorola. It is full-duplex and uses master-slave architecture (single master). It also has higher throughput as compared to I2C. It uses a four-wire serial bus:

1. SCLK – Serial Clock. Other names include SCK

2. MOSI – Master Out Slave In. Other names include SIMO, SDI, DI, DIN, SI, MTSR.

3. MISO – Master In Slave Out. Other names include SOMI, SDO, DO, DOUT, SO, MRST.

4. SS – Slave Select. Other names include $\overline{SS}$, SSEL, CS, $\overline{CS}$, CE, nSS, /SS, SS#

It is used for talking to a variety of peripherals. Flash and EEPROM chips also use SPI. The methodology of testing and analyzing is similar to I2C, just that we have a different bus interface. We will talk about SPI in detail in the later Blog posts.

2.5 USB

The device can have a USB (mini/micro etc) interface for either charging or communication. For latter, it becomes necessary to test the interface for known or unknown issues. We should sniff the communication for run-time analysis as well as fuzz the USB interface for unknown bugs.

2.6 Sensor

It is a loose name that we have given to mean the interface to the physical world. It may not necessarily be limited to a sensing type interface. For example, a temperature sensor would be a perfect example, but also a door lock which does not sense anything but controls the physical world by the "Lock/Unlock" action. These can be divided into three types based on their operation:

1. Monitor: This is more closely linked to the literal meaning of a sensor i.e. to sense or monitor the physical world for any changes. Ex. temperature, motion, Pulse, Blood pressure, tyre pressure etc.

2. Control: These types of devices control the physical world in some way or the other. Ex. Locks, dispensers etc.

3. Hybrid: These are a combination of both the above types i.e. temperature control, Lights based on time of the day etc.This is one of the critical interfaces as all values and data derived from the physical world will be transferred to the cloud. If attackers can force the device with malformed (wrong) data then the whole ecosystem gets affected as all decisions and statistics are based on this data. In other words, this is the crux of the IoT ecosystem. Wrong values here can have catastrophic effects on the decisions that the ecosystem makes.

2.7 Human Machine Interface

As with Sensor interface, we use the HMI as a general term to define the interface between the user and the device without restricting it to the term used in Industrial Control systems. This is the interface that users can use to communicate with the device and operate on it directly. Some of the common examples would be, touch screens, push buttons, touchpads etc. It is important to test this interface to find out any bypass mechanisms, security flaws etc.

2.8 Other Hardware interfaces

There are many other hardware interfaces used to communicate with the device. As a pentester, it is important to analyze and find flaws and bypass mechanisms in all the interfaces. Some of the well-known interfaces include (but not limited to):

1. D-Subminiature – **https://en.wikipedia.org/wiki/D-subminiature**

2. ecommended Standards (RS232, RS485 etc) – More details on RS protocols can be foind on https://en.wikipedia.org/wiki/EIA_standards
3. On-board Diagnostics (OBD) – **https://en.wikipedia.org/wiki/On-board_diagnostics**

3. Network Communication Interface

This interface allows the device to talk to the rest of the virtual world which includes the sensor network, cloud and mobile. The hardware interfaces responsible for network communication may have their own separate microcontroller/firmware that provides the communication functionality. The attack surface in this case is the firmware or driver code that implements the low-level communication.

3.1 Wifi

The wifi interface has some known issues. etc. From the attack surface perspective, it would be interesting to attack the wifi chip possibly to damage it, DOS, bypass security restrictions or code execution.

3.2 Ethernet

The Ethernet interface, like wifi interface, has its share of low-level TCP/IP stack vulnerabilities as well as hardware implementation vulnerabilities and similar attack vectors.

3.3 Radio

The Radio interface has become one of the most important attack surface considering many IoT products have shifted to/are being built with radio communication. The preference stems from the fact that it is more efficient to use Radio in many cases over Wifi/Wired network connectivity. The reason I have categorized Wifi separately and not in this section is primarily to make a clear distinction between devices that can connect directly to the internet (Wifi/Wired) and the ones that require a gateway (ex. Smart Hubs) that implements both the Radio as well as Wifi/Wired interfaces, to communicate with the sensors and the internet respectively.From the actual communication perspective, think of it as two different modes of communication:

1. Simple/Unstructured: This type is usually used in simple products like shutters, locks, doorbells etc. By Simple and unstructured we mean that it uses simple (mostly proprietary) data (stream) and sends it across via the radio interface. As a penetration tester, you need to reverse engineer the communication to find out flaws in the implementation. It is easy to sniff the radio communication using radio sniffing hardware tools like SDR (Software Defined Radio) etc.

2. Complex/Structured: Complex and structured communication means that it uses structured packets for radio communication which are complex as they carry additional and meta information about the protocol in addition to just the data. These protocols have become quite famous in the IoT world due to efficiency, standardization, economic chips, convenience of implementation. Again, there are various tools available to sniff and parse the protocol to extract application specific data that is sent across. Some of the common protocols include:
a. Bluetooth (and BLE)
b. ZigBee
c. Zwave
d. NFC

e. RFID
f. LoRA
g. Wireless HART

If you have any queries, feel free to get in touch with us at info [_a t_] payatu DOT com, aseem [_a t_] payatu DOT com.

Payatu Software Labs specializes in IoT, embedded, Mobile and Cloud penetration testing services and Practical IoT Hacking training worldwide. If you are interested in corporate training or security testing of your IoT products, kindly get in touch with us – info [_a t_] payatu DOT com

Continue to part 3

Reference

1. UART – **https://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter**
2. Types of memory – **https://en.wikipedia.org/wiki/Semiconductor_memory**
3. JTAG – **https://en.wikipedia.org/wiki/JTAG**
4. SWD – **https://www.arm.com/files/pdf/Serial_Wire_Debug.pdf**
5. JTAG Vs. SWD – **https://electronics.stackexchange.com/questions/53571/jtag-vs-swd-debugging**
6. I2C – **https://en.wikipedia.org/wiki/I%C2%B2C**
7. IoT Protocols – **https://www.postscapes.com/internet-of-things-protocols/**
8. I2C Serial EEPROM image source – **https://upload.wikimedia.org/wikipedia/commons/thumb/7/71/AT24C02_EEPROM_1480355_6 AT24C02_EEPROM_1480355_6_7_HDR_Enhancer.jpg**
9. JTAG port image source – **https://i.stack.imgur.com/IiUqm.jpg**