Volume: 13 Issue: 03 Pages: 4988-4992(2021) ISSN: 0975-0290

OTA Secure Update System for IoT Fleets

Laurentiu-CristianDuca
Department of Computer Science, University Politehnica of Bucharest
Email: laurentiu.duca@upb.ro
Anton Duca
Department of Electrical Engineering, University Politehnica of Bucharest
Email: anton.duca@upb.ro
Cornel Popescu
Department of Computer Science, University Politehnica of Bucharest
Email: cornel.popescu@upb.ro

----------**ABSTRACT**----------

In this paper, the authors present an Over-The-Air secure and scalable update system for Internet of Things fleets which run embedded Linux. The system uses the SWUpdate Linux update agent and Eclipse hawkBit framework to offer a scalable and server and client fault tolerant update mechanism for IoT fleets where no physical access is available. Security is achieved using the Linux OpenSSL library. By using Buildroot we implement a complete automated build system.

Keywords - **Embedded, Eclipse hawkBit, IoT, Linux, OpenSSL, SWUpdate, U-boot.**

Date of Submission: Oct 05, 2021 Date of Acceptance: Nov 11, 2021

I. INTRODUCTION

It is clear that embedded devices need software updates. This is especially needed in systems like [1] to solve bugs, to add new features or for security upgrades. There are multiple programs that need to be updated: the n stage boot loader, the operating system's kernel, the device tree blob that describes the hardware and the root file system which contains the applications.

The classic way to update a system is the manually update, made by the system administrator, either by simply copying the new files or using the distribution's package manager. OTA software updates are remote updates and they are especially needed when is hard to get access to the device to be updated. In this case, the target devices are clients and they connect to the server to check for updates. In case of multiple devices, the overall update process can be eased by automating it.

Regarding the system organization for updating, classified by the method used to recover the system in the event of a failure, there are the so called symmetric and asymmetric systems [2].

Asymmetric systems have one recovery partition which allows the user to replace the main OS partition in case of a failure; the most common example is the Android operating system.

Symmetric systems have dual main operating system partitions: the active one and the inactive one. To make a system update, the inactive partition is updated and will became the active one; in case of a failure, the fall back can be made by the boot loader, which boots the system from the inactive partition. The asymmetric and symmetric system representations are shown in Fig. 1 and Fig. 2. The fall back can be done by using a watchdog in the boot loader. The symmetric representation has advantage of less system downtime in case of a failure, so, if space is not an issue this variant is more adequate to be chosen [3]. In our implementation we chose the symmetric representation.

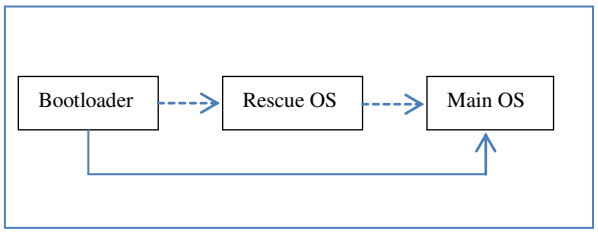

Fig. 1.Asymmetric systems.

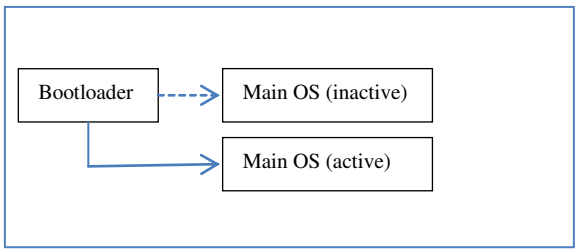

Fig. 2. Symmetric systems

II. LINUX UPDATE TOOLS – STATE OF THE ART

Among the most famous open source tools for embedded Linux update [4], [5], we find Mender, RAUC and SWUpdate.

Mender is described as being secure, risk tolerant and efficient Over-The-Air up-date system for all device software [6]. It is written in the Go language and is free for individuals and "do it yourself" projects, but for teams the price varies from 29 to 249 dollars per month.

Volume: 13 Issue: 03 Pages: 4988-4992(2021) ISSN: 0975-0290

RAUC stands for “Robust Auto-Update Controller”. RAUC can be a target application which runs the update client on the embedded device, and can also be a host application that manages the installation artifacts [7]. It is written in C and is designed to be a lightweight update tool.

SWUpdate is an acronym for “Software Update for Embedded Linux Devices” [8]. Its goal is to provide a safe and efficient Linux update mechanism for embedded systems [9] at no charge, being completely free. It is written in C, is highly configurable and has security features, so we have chosen it in our IoT OTA secure update system.

III. IMPLEMENTATION AND CONFIGURATION

The project consists in one server which run Eclipse hawkBit and multiple clients which compose the IoT fleet. Each client runs a Buildroot Linux distribution which use the SWUpdate program tool. The clients repeatedly try to connect to the server and verify if an upgrade is available, and, if so, they will update their Linux system image.

We started from [5], a simple and not secure, one target board only, update system. To this project, we added the following features: allow multiple clients (targets), scalability, security using OpenSSL, rollback mechanism if the client update fails, reduced server uptime – the clients try to repeatedly connect to the server at discrete time intervals, and we ported our software on the cheap and handy Raspberry Pi Zero W.
The sources of our project are free and open, being available on the Internet [10].

III.1. Server configuration

For the update server we used Eclipse hawkBit. Eclipse hawkBit is a program which allows admin triggered software updates to the remote connected embedded targets [11]. It is written in Java, being platform independent. The server can be run on http or https.

In order to enable secure updates, we used https and a login username and password. To enable https, we must first generate the private and public keys, then we create a self signed certificate (for proof of concept, we use “laurPC-100”, the development hostname) and finally we store the cryptography objects in a single PKCS #12 file. The commands are listed in Listing 1.

```
$ opensslgenrsa -aes128 -out jetty.key
passwd: admin12
$ openssl req -new -x509 -days 3650 -sha256 -key
jetty.key -out jetty.crt -subj
"/C=RO/L=Bucharest/CN=laurPC-100"
$ openssl pkcs12 -inkey jetty.key -in jetty.crt -
export -out jetty.pkcs12
export passwd: admin12
```

Listing 1.Generating the cryptography objects for https.

We used the default Docker image with hawkBit server in order to deploy our server application. We start the Docker container with the script shown in Listing 2. Here we use the cryptographic objects that we have generated and install them in the container file system; we establish the username and password credentials to login; we set up the server port and set the polling time; and finally we set TLS as the SSL protocol. The last line is commented out and we kept it because was useful to check that the server was functioning with the Eclipse hawkBit device simulator.

The upgrade can be triggered by the system administrator. The hawkBit server allows selecting the embedded devices by using filters (for example for filtering by name we use “name =ge= ddi2” to select the devices which names are greater or equal to “ddi2” string). The most important thing is that the upgrade can also be done by using the rollout function that applies to devices of a specified filter, and these devices can be grouped. The system starts to upgrade the first group, and, after it had upgraded a specified percentage of targets from this group (for example 50%), it starts upgrading the next group and so on. This way the system is scalable and does not hang when it has many targets to upgrade.

```
sudodocker run -p 8443:8443 \
-v "$PWD/jetty.pkcs12:/opt/hawkbit/jetty.pkcs12"
\
hawkbit/hawkbit-update-server:latest \
--security.user.name=admin \
--security.user.password=admin
--spring.main.allow-bean-definition-
overriding=true \
--
hawkbit.server.ddi.security.authentication.target
token.enabled=true \
--hawkbit.server.repository.publish-target-poll-
event=false \
--hawkbit.controller.pollingTime=00:00:30 \
--hawkbit.dmf.rabbitmq.enabled=false \
--hawkbit.artifact.url.protocols.download-
http.protocol=https \
--hawkbit.artifact.url.protocols.download-
http.port=8443 \
--security.require-ssl=true \
--server.use-forward-headers=true \
--server.port=8443 \
--server.ssl.key-store=jetty.pkcs12 \
--server.ssl.key-store-password=admin12 \
--server.ssl.key-password=admin12 \
--server.ssl.protocol=TLS \
--server.ssl.enabled-
protocols=TLSv1,TLSv1.1,TLSv1.2,TLSv1.3
```

Listing 2.hawkBit starting script.

The IoT fleet update time (SN) is a linear function of the time to update a single board (S1) multiplied with the number of boards (N) and divided by the number of parallel updates (Np):

$$S_N = O\left(\frac{S_1 \cdot N}{N_p} \right) \quad (1)$$

Building the effective upgrade file is shown in the section “The target system” because is dependent of the SWUpdate tool.

III.2. The build system

The build system used in this implementation is Buildroot, version 2020.11.3. We choosedBuildroot because is an efficient tool which can be used to generate embedded

Volume: 13 Issue: 03 Pages: 4988-4992(2021) ISSN: 0975-0290

Linux systems from boot loader to kernel and root file system through cross compilation [12]. After downloading and extracting the Buildroot archive, the user can build the system by using three commands listed in Listing 3.

```
make raspberrypi0w_defconfig
make menuconfig
make
```

Listing 3.Buildroot commands

Here we will focus on the Raspberry Pi Zero W board which we have used to implement our update system. First command sets up the default configuration in Buildroot for our board, the second is used to make specific configuration and the third simply builds the Linux system.

The specific configuration that we chose for Buildroot is listed in Listing 4. The settings are self explanatory.

```
Build options
    Enable Compiler Cache (fast recompile)
Toolchain
Toolchain type ---> external toolchain
Toolchain (Bootlintoolchains)
Toolchain origin (Toolchain to be downloaded and
installed)
    C library (glibc)
System configuration
Init system
systemv
    /bin/sh bash
    Run a getty login prompt after boot
Kernel
Defconfig name (bcm2835_defconfig)
    In-tree Device Tree Source file names
(bcm2835-rpi-zero-w)
Target Packages
    Networking applications
wpa_supplicant - Enable 80211 support
dropbear
ntpd    # for SWUpdatelibcurl
ntpdate
 Compression and decompression
zlib support #for SWUpdate
 Crypto
     libsha1
openssl support
ssl library (openssl)
openssl binary
openssl additional engines
    Hardware handling
        Firmware
rpi-firmware
rpi-wifi-firmware
rng-tools
      u-boot tools
mkimage
fw_printenv
    Networking
libcurl
        curl binary
    Shell and utilities
        bash
    System tools
SWUpdate
         (package/SWUpdate/SWUpdate.config)
SWUpdate configuration file
       watchdog
Filesystem images
       ext2/3/4
exactsize            300M
Boot loaders
    U-boot
        ((rpi_0_w_defconfig) Board defconfig
filename
        CONFIG_ENV_FAT_FILE="uboot.env" # default
```

Listing 4.Buildroot configuration.

Before issuing the “make” command in Buildroot, because we use SWUpdate, we also must configure this tool by entering the “make swupdate-menuconfig” command and selecting the settings in Listing 5. Suricatta feature adds support for hawkBit with SSL.

```
SSL implementation to use (OpenSSL)
Suricatta
    Features → SSL support
    Server Type → (hawkBit support)
Image handlers
    Raw
```

Listing 5.SWUpdate settings.

In the “board/raspberrypi0w” folder, the “genimage-raspberrypi0w.cfg” was set such that we have a symmetric update system with three partitions: one with boot loader plus dtb, and two with rootfs (to which we’ll automatically add kernel and dtb, by using a post build script). We have downloaded from the device a U-boot saved environment named “uboot.env” and made it available with the project sources. A “rpi-firmware” folder must be built on Buildroot “output/images” path and its contents set as shown in Listing 6.

```
image boot.vfat {
vfat {
    files = {
      "bcm2835-rpi-zero-w.dtb",
      "rpi-firmware/bootcode.bin",
      "rpi-firmware/cmdline.txt",
      "rpi-firmware/config.txt",
      "rpi-firmware/fixup.dat",
      "rpi-firmware/start.elf",
      "rpi-firmware/overlays",
      "u-boot.bin",
      "uboot.env"
    }
  }
  size = 32M
}
image sdcard.img {
hdimage { }
  partition boot {
    partition-type = 0xC
    bootable = "true"
    image = "boot.vfat"
  }
  partition rootfs1 {
    partition-type = 0x83
    image = "rootfs.ext2"
    size = 1024M
  }
  partition rootfs2 {
    partition-type = 0x83
    image = "rootfs.ext2"
    size = 1024M
  }
}
```

Listing 6.The “genimage-raspberrypi0w.cfg” script

U-Boot (also written as Uboot) is an open source boot loader. It is used in embedded systems to boot the target operating system kernel [13]. Ue used version 2020.10. In the same “board/raspberrypi0w” folder we wrote the “uboot-fragment.config” file which defines variables in U-boot boot loader at system build time. Its contents are

Volume: 13 Issue: 03 Pages: 4988-4992(2021) ISSN: 0975-0290

shown in Listing 7. The BOOTCOMMAND sets the default partition to boot Linux, to the second partition. Also, the watchdog is enabled with a maximum delay of 30 seconds; if Linux does not boot in 30 seconds, the system will be reset. If the system will not boot well Linux for 3 tries, then the ALTBOOTCOMMAND will be used to set boot parameters for U-boot instead of BOOTCOMMAND.

```
CONFIG_BOOTCOUNT_LIMIT=y
CONFIG_BOOTCOUNT_BOOTLIMIT=3
CONFIG_SYS_BOOTCOUNT_MAGIC=0xB001C041
CONFIG_BOOTCOUNT_GENERIC=y
CONFIG_SYS_BOOTCOUNT_ADDR=0x7000A000
CONFIG_BOOTCOMMAND="setenvbootargs
\"8250.nr_uarts=1 root=/dev/mmcblk0p2 rootwait
console=tty1 console=ttyS0,115200\"; load mmc 0:2
$kernel_addr_rzImage ; load mmc 0:2 $fdt_addr_r
bcm2835-rpi-zero-w.dtb ; bootz $kernel_addr_r -
$fdt_addr_r"
CONFIG_WATCHDOG_TIMEOUT_MSECS=30000
CONFIG_BCM2835_WATCHDOG=y
```

Listing 7. "uboot-fragment.config".

In order for the watchdog to work in U-boot for Raspberry Pi Zero W, we needed to apply the Paolo Pisati's patch to U-boot sources [14]. This was made by copying the (slightly modified) patch to folder "boot/uboot/2020.10" Buildroot folder.

In the Linux kernel the watchdog must also be enabled, by setting CONFIG_WATCHDOG=y and CONFIG_BCM2835_WDT=y. We also built a program tool which will reset the U-boot's BOOTCOUNT variable when Linux successfully boots; this program uses "fw_printenv" and "fw_setenv" commands to access U-boot's environment.

The "config.txt" file from the boot partition needs to contain "enable_uart=1" to use the UART for debugging and "dtparam=watchdog=on" in order to use the watchdog.

III.3 The client system

On the embedded Linux targets we use the SWUpdate tool for remote software updates. In order to generate the effective update file, a script named "SWUpdate-image.sh" was written (see Listing 8) which creates a cpio archive that SWUpdate will use to upgrade the system. This script packs the "sw-description" file (see Listing 9), which tells SWUpdate what are the updatable partitions, along with the file used to install the new root file system which is "rootfs.ext4.gz" and is generated by Buildroot.

```
IMG_FILES="sw-description rootfs.ext4.gz"
for f in ${IMG_FILES} ; do
    echo ${f}
done | cpio -ovL -H crc>buildroot.swu
```

Listing 8.The "swupdate-image.sh" script.

In the Buildroot folder "board/raspberrypi0w", we wrote the "post-build.sh" script and added a folder named "overlay" which contains additional files to be written to the target's root file system.

In "post-build.sh" we made the following additions:

- wrote to "/etc/hosts" the IP of the "laurPC-100" host which is used in OpenSSL;
- set the DAEMON_ARGS variable in "/etc/default/rngd" to "-o /dev/random -r /dev/urandom" (this is useful for developing speed where we used rngd to get faster random entropy);
- replaced in "config.txt" file of the boot partition "zImage" with "u-boot.bin", so the Raspberry Pi Zero W will boot first the U-boot bootloader instead of Linux;
- in "config.txt" we set "device_tree=bcm2835-rpi-zero-w.dtb" such that U-boot can find this device tree blob;
- copy "zImage" and "bcm2835-rpi-zero-w.dtb" to the root file system;
- setup the path and length of "uboot.env" to "/etc/fw_env.config" in order to have access to the U-boot BOOTCOUNT variable from Linux;
- setting the Linux "watchdog-timeout" to 15 seconds in "/etc/watchdog.conf".

In the Buildroot "overlay" folder various files were created:

- "S30watchdog" and "S98swupdate" services available in "/etc/init.d" on the target;
- "postupdate.sh", "swupdate.cfg" and other files available in "/etc/swupdate" on the target.

```
software = {
    version = "0.1.0";
rootfs = {
        rootfs-1: {
            images: (
            {
                filename = "rootfs.ext4.gz";
                compressed = "zlib";
                device = "/dev/mmcblk0p2";
            });
        }
        rootfs-2: {
            images: (
            {
                filename = "rootfs.ext4.gz";
                compressed = "zlib";
                device = "/dev/mmcblk0p3";
            });
        }
    }
}
```

Listing 9. The "sw-description" file.

The "S98swupdate" service does the following:

- saves to a file "/root/mmc-part-id" the id of the partition which Linux booted;
- gets and saves the target machine id (every Raspberry Pi board has associated a unique id, see Listing 10);
- starts the effective swupdate daemon and specifies it the target id, target token, hawkBit URL and the partition that booted Linux;

Volume: 13 Issue: 03 Pages: 4988-4992(2021) ISSN: 0975-0290

- runs "/etc/swupdate/connect.sh" that will try to login to hawkBit - using username and password that are stored encrypted on disk, every 5 minutes, to be server fail safe;
- if the file "/update-ok" exists then will acknowledge to the hawkBit server that the update was successful (and then delete "/update-ok").

```
BOARD_ID=`cat /proc/cpuinfo | grep Serial | awk -
F': ' '{print $2}'`
```

Listing 10.Extracting the Raspberry Pi Zero W board identifier.

After the administrator ordered an upgrade and the SWUpdate tool has finished installing it, the "/etc/swupdate/postupdate.sh" bash script specified in "/etc/swupdate/swupdate.cfg" will be run. This one sets up the U-boot environment and Linux kernel command line such that the upgraded partition will become the boot partition, then creates the file "/update-ok" and reboots the target. The upgraded partition is the one different from "/root/mmc-part-id".

IV. CONCLUSION

In this paper we presented our free and open source implementation of an OTA secure update system for IoT fleets. The target system runs Linux and uses the SWUpdate tool. The server hawkBit that we used from Eclipse is platform independent.

The update system is scalable by allowing a theoretically unlimited number of targets in the IoT fleet, by using the hawkBit's rollout feature for fleet upgrade.

The system is target fail safe, by being a symmetric update system and using the U-boot's BOOTCOUNT mechanism.

The server can be started only when the upgrade is available, because the clients try to login to the server at discrete time intervals (5 minutes).

The IoT fleet used to test the update system is handy and cheap being composed by Raspberry Pi Zero W boards.

REFERENCES

[1] A. Sharif, J.P. Li, M.A. Saleem, "Internet of Things Enabled Vehicular and Ad Hoc Networks for Smart City Traffic Monitoring and Controlling: A Review", International Journal of Advanced Networking and Applications, Volume 13, Issue 02, pp. 4925-4930, 2021.

[2] M. Krak, The ultimate guide to software updates on embedded Linux devices, pp. 12-18, FOSS-North conference, 2018.

[3] V. Baillie, OTA updates for Embedded Linux, part 1 – Fundamentals and implementation, https://www.embedded.com/ota-updates-for-embedded-linux-part-1-fundamentals-and-implementation, last accessed 2021/09/31.

[4] V. Baillie, "OTA updates for Embedded Linux, part 2 – A comparison of off-the-shelf update systems", https://www.embedded.com/ota-updates-for-embedded-linux-part-2-a-comparison-of-off-the-shelf-update-systems, last accessed 2021/09/31.

[5] T. Petazzoni, Building a Linux system for the STM32MP1: remote firmware updates, https://bootlin.com/blog/tag/SWUpdate, last accessed 2021/09/31.

[6] Mender update system web page, mender.io, last accessed 2021/09/31.

[7] RAUC update system web page, https://github.com/rauc/rauc, last accessed 2021/09/31.

[8] SWUpdate update system web page, https://sbabic.github.io/SWUpdate, last accessed 2021/09/31.

[9] S. Babic, Software Update on Embedded Systems, pp. 18-28, Embedded Linux Conference Europe, 2014.

[10] L.-C. Duca: IoT OTA secure update system sources, https://github.com/laurentiuduca/iot-ota-sus, last accessed 2021/09/31.

[11] Eclipse hawkBit web page, https://eclipse.org/hawkbit, last accessed 2021/09/31.

[12] Buildroot build system web page, https://buildroot.org, last accessed 2021/09/31.

[13] U-boot boot loader web page, https://www.denx.de/wiki/U-Boot, last accessed 2021/09/31.

[14] P. Pisati, U-boot patch – support for the BCM2835/2836 watchdog, https://lists.denx.de/pipermail/u-boot/2017-January/279573.html, last accessed 2021/09/31.