USENIX Security '24 Artifact Appendix: Unveiling IoT Security in Reality: A Firmware-Centric Journey

Nicolas Nino*
University of Georgia

Ruibo Lu*, Wei Zhou
Huazhong University of Science and Technology

Kyu Hyung Lee
University of Georgia

Ziming Zhao
Northeastern University

Le Guan
University of Georgia

A Artifact Appendix

A.1 Abstract

To study the security properties of the Internet of Things (IoT), firmware analysis is crucial. In the past, many works have been focused on analyzing Linux-based firmware. Less known is the security landscape of MCU-based IoT devices, an essential portion of the IoT ecosystem. Existing works on MCU firmware analysis either leverage the companion mobile apps to infer the security properties of the firmware (thus unable to collect low-level properties) or rely on small-scale firmware datasets collected in ad-hoc ways (thus cannot be generalized). To fill this gap, we create a large dataset of MCU firmware for real IoT devices. Our approach statically analyzes how MCU firmware is distributed and then captures the firmware. To reliably recognize the firmware, we develop a firmware signature database, which can match the footprints left in the firmware compilation and packing process. In total, we obtained 8,432 confirmed firmware images (3,692 unique) covering at least 11 chip vendors across 7 known architectures and 2 proprietary architectures. We also conducted a series of static analyses to assess the security properties of this dataset. The result reveals three disconcerting facts: 1) the lack of firmware protection, 2) the existence of N-day vulnerabilities, and 3) the rare adoption of security mitigation.

A.2 Description & Requirements

This section provides all the information necessary to recreate the same experimental setup to run the artifact. Our artifact includes 1) tools to analyze Android APKs to extract URLs for OTA firmware update; 2) tools to extract Android APKs for candidate firmware; 3) an LLM-powered crawler to download candidate firmware from OTA URLs; 4) a pipeline to validate firmware and extract firmware metadata; 5) a dataset of 3,692 firmware images that were collected using the tools above; 6) firmware analysis tools to detect security defects in the collected firmware.

As discussed in the paper, releasing the firmware dataset poses potential risks, including copyright infringement and misuse. Therefore, only the tools will be released. The firmware dataset is only provided temporarily during the AE process and will be deleted afterwards.

A.2.1 How to access

The artifact can be downloaded from the GitHub repo publicly available at `https://github.com/MCUSec/RealworldFirmware/releases/tag/usenixae`.

A.2.2 Hardware dependencies

A machine with an x86-64 CPU and at least 24 GB of memory and 60 GB of free storage is recommended.

A.2.3 Software dependencies

A Linux environment is needed. While all major distributions should be supported, we recommend Ubuntu $\geq$ 20.04. Our artifacts have been tested on Ubuntu 22.04 LTS. For module specific dependencies please refer to §A.3.

A.2.4 Benchmarks

- APK dataset: A CSV file containing the names and SHA values of the 40,675 APKs used in our experiments.
- Test dataset: Since it is impractical to run our tools against all 40,675 APKs, we selected 20 sample APKs in the folder `apk-dataset` for testing.
- Firmware dataset: The 3,692 firmware images collected in our experiments. It can be downloaded from `https://drive.google.com/file/d/1b0gr-5a7ICIvpylGSNf-Xp3m_dLfmRy2/view?usp=sharing`. We will delete it after the AE process.

A.3 Set-up

This section describes the steps to set up the experiment environment, assuming a fresh Ubuntu 22.04 installation.

A.3.1 Installation

Install general software dependencies:

- `sudo add-apt-repository ppa:deadsnakes/ppa`
- `sudo apt -y update`
- `sudo apt install -y openjdk-11-jdk openjdk-17-jdk python3.11 python3-pip curl z3 unzip rsync`
- `pip3 install --upgrade pip`

Clone the artifact folder from Github. Set the project root directory to the location where the artifact was cloned: `export PROJECT_FOLDER=/home/<USER>/RealworldFirmware`. Then, enter the project folder: `cd $PROJECT_FOLDER`.

Install `apktool` for disassembling the APKs: `sudo ./install_apktool.sh`.

The folder `$PROJECT_FOLDER/otacap` contains URL analysis tool. It depends on the `Z3` solver.

1. `mv otacap/VSA/build/dependencies/libz3java.so /usr/lib/x86_64-linux-gnu/jni/`
2. `mv otacap/VSA/build/dependencies/libz3.so /usr/lib/x86_64-linux-gnu/jni/`

In folder `$PROJECT_FOLDER/FirmXRay`, we have a customized FirmXRay. Add the `ghidra.jar` file, found in https://drive.google.com/file/d/1emNNUBO61lLMdDBbVeI8z5NCdl5GypsG/view?usp=sharing to `$PROJECT_FOLDER/FirmXRay/lib/`. To build it, run `cd $PROJECT_FOLDER/FirmXRay && make`.

In folder `$PROJECT_FOLDER/binwalk`, we have a customized binwalk. To set up its dependencies and install it:

1. `cd $PROJECT_FOLDER/binwalk`
2. Install dependencies: `pip3 install -r requirements.txt && sudo pip3 install protobuf==3.6.1`
3. Install binwalk: `python3 setup.py install`

Install Ollama with llama3:

1. `curl -fsSL https://ollama.com/install.sh | sh`
2. `ollama serve &`
3. `ollama pull llama3`
4. `pip3 install ollama==0.2.0`

In folder `$PROJECT_FOLDER/crawler`, we have the Crawler. To set up its dependencies:

1. `cd $PROJECT_FOLDER/crawler`
2. Install scrapy: `pip3 install scrapy==2.11.2`
3. In `crawler/httpftp/source/settings.py`, modify the field `FILES_STORE` to `$PROJECT_FOLDER/crawler/httpftp/results/files`. Note that `$PROJECT_FOLDER` must be replaced with the real path.

Folder `$PROJECT_FOLDER/FirmFlaw` contains the binary analysis tool. To set up its dependencies:

1. `chmod 1777 /tmp`
2. `cd $PROJECT_FOLDER/FirmFlaw`
3. `mkdir logs res db fidb ghidra_projects firmwares`
4. Install pyhidra: `pip3 install pyhidra==1.2.0`
5. `curl -L -O https://github.com/NationalSecurityAgency/ghidra/releases/download/Ghidra_11.1_build/ghidra_11.1_PUBLIC_20240607.zip`
6. `unzip ghidra_11.1_PUBLIC_20240607.zip`
7. `export GHIDRA_INSTALL_DIR=$PROJECT_FOLDER/FirmFlaw/ghidra_11.1_PUBLIC`

A.3.2 Basic Test

[OTACap] Extracting OTA URLs from APKs.

1. `cd $PROJECT_FOLDER/otacap/VSA`
2. `./gradlew build -Dorg.gradle.java.home=/usr/lib/jvm/java-11-openjdk-amd64/`
3. `cd .. && java -Xms5g -Xmx16g -jar VSA/build/libs/IoTScope-1.0-SNAPSHOT-all.jar -d config/combined.json -p ../Android/Sdk/platforms/ -o ./output-jsons/ -t config/taintrules.json -a ../apk-dataset/com.brocel.gdb -dj dex_tools_2.1/d2j-dex2jar.sh`

OTACap should output a JSON file for the APK to `otacap/output-jsons`. It contains the reconstructed URLs along with some metadata.

[bin-unpack] Extracting firmware from APKs.

1. `cd $PROJECT_FOLDER/bin-unpack`
2. `python3 decompress-apks.py`
3. `chmod +x ./extract-binaries.sh && ./extract-binaries.sh`

It should extract APKs in `apk-dataset` and copy the found firmware images to `$PROJECT_FOLDER/bin-unpack/fw_images`.

[FirmProcessing] Pipeline to recognize firmware and extract metadata.

1. `cd $PROJECT_FOLDER/FirmProcessing`
2. `sudo python3 run_step1_convert2bin.py`
3. `python3 run_step2_binsorter.py --enable-firmxray`

The script `run_step1_convert2bin.py` takes in candidate images in `FirmProcessing/originals` and decodes them. The results are stored in the folder `FirmProcessing/step1_bins`, which are further analyzed by the script `run_step2_binsorter.py` to recognize and categorize firmware. The final results are stored in `FirmProcessing/step2_PostSig`. Each image is accompanied by a JSON file containing firmware metadata such as base address, entry point, and architecture when found.

[FirmFlaw] The binary analysis tool.

1. `cd $PROJECT_FOLDER/FirmFlaw`

2. `./build.sh ../FirmProcessing/step2_postSig`
3. `python3 Mitigation.py ./ghidra_projects arm_bins`
4. `./FunctionID.sh && ./SimMatch.sh`
5. `python3 ResGen.py`

After FirmFlaw is complete, we can find the results documented in `./res/results.md`. This file presents the results of the complexity analysis, mitigation detection and library adoption in markdown table format. Additionally, it includes detailed descriptions explaining the items in the table and their meaning.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): OTACap can recover URLs used in firmware update from APKs. This is proven by the experiment (E1), which runs OTACap against 20 sample APKs. Our paper shows the results obtained from running all the 40,675 APKs in Section 6.3.

(C2): Our LLM-powered crawler can fully exploit the recovered URLs from E1 and download potential firmware. This is proven by the experiment (E2) described in Section 6.4 of our paper.

(C3): Our firmware validation pipeline leverages the footprints left in the firmware compilation and packing process to confirm firmware and to extract metadata. This is proven by the experiment (E3) described in Section 6.4 of our paper.

(C4): Our binary analysis tool supports Arm- and Xtensa-based firmware. For each image in the provided firmware dataset, it counts the function number (`Fig. 3` and `tables 6` and `7`), firmware size (`Fig. 4` and `table 6`), mitigation adoption rates (`Table 10`), and library adoption (`Tables 8` and `9`). This is proven by the experiment (E4) described in Section 7.2 of our paper.

A.4.2 Experiments

(E1): [OTACap] [3 human-minutes + 1 to 6 compute-hours]: Run OTACap on the APKs stored in apk-dataset.
How to: We provide a script that runs OTACap on the APKs in apk-dataset.
Preparation: Set up and install apktool and OTACap.
Execution: `cd otacap && ./run.sh`
Results: The results are located in the `outputs-json` folder inside the `otacap` folder. Each JSON file corresponds to an APK. The reconstructed URLs that will be used in next steps appear in the `ValueSet` fields. The file comes with more fields that contain metadata.

(E2): [Crawler] [5 human-minutes + 1 to 6 compute-hour]: Run our crawler to download potential firmware images using the URLs collected in E1.
How to: We provide a script that executes our crawler using the URLs obtained in the previous experiment (E1).
Preparation: Install and set up Ollama, Scrapy and the crawler environment.
Execution: `cd crawler && ./run.sh`
Results: The tool should download around 40 potential images, which will be stored in the folder `crawler/httpftp/results/files`.

(E3): [Firmware recognition pipeline] [5 human-minutes + 30 compute-minutes]: Run the firmware recognition pipeline to delete false positives, extract metadata from the images, and prepare a copy for firmware analysis.
How to: We provided a script to run the complete pipeline on the images crawled in E2.
Preparation: Set up and build FirmXRay and binwalk dependencies.
Execution: `cd FirmProcessing &&./pipeline.sh`
Results: We expect around 30 confirmed firmware images, which will be stored in the folder `FirmProcessing/step2_postSig`.

(E4): [Binary Analysis pipeline] [1 human-minutes + 8-10 compute-hour]: Run the binary analysis pipeline to analyze firmware complexity, detect attack mitigation and library adoption in firmware images. The target is the whole firmware dataset we collected from 40,675 APKs, instead of the 20 APK samples.
How to: We provide a script to execute the analysis pipeline on the target firmware images.
Preparation: Set up the FirmFlaw environment and download the firmware dataset (link in §A.2.4) to `$PROJECT_FOLDER`. Then, `unzip all_firmware.zip`.
Execution: `cd FirmFlaw && ./pipeline.sh ../all_firmware/arm_and_xtensa`.
Results: This file presents the results of the complexity analysis, mitigation detection and library adoption in markdown table format. Additionally, it includes detailed descriptions explaining the items in the table and their meanings. It should agree with the tables in Section 7.2 of our paper.

A.5 Notes on Reusability

Our firmware collection tool downloads firmware using URLs obtained from APKs. Because the device manufacturers might invalidate the URLs from time to time, the experiment results could change.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2024/.