A Artifact Appendix

A.1 Abstract

Our artifact is the code to create an ultra-wide band (UWB) high-Rate pulse repetition frequency (HRP) sniffer that generates accurate timestamps and forwards timestamps and UWB frames to Wireshark. We used this sniffer to identify timings in UWB ranging sequences and to attack the frames using the Ghostpeak attack. The artifact includes all necessary source code to run it on a recent DWM3000EVB from Decawave attached to a NUCLEO-F429ZI. A different board can be used, but the speed may suffer due to different SPI speeds.

We do not publish sample code for the attacks demonstrated in our paper, since this would violate German laws and might allow malicious actors to enter a system secured by UWB distance ranging.

A.2 Artifact check-list (meta-information)

- **Algorithm:**
- **Compilation: For the UWB sniffer we use the free STM32CubeIDE. The host a Python script.**
- **Binary: We do not include binaries, since the configuration needs to be changed depending on the UWB channels to listen on.**
- **Run-time environment: The STM32CubeIDE runs on Linux, macOS and Windows**
- **Hardware: We use a DWM3000EVB as the UWB receiver and attach it to a NUCLEO-F420ZI. The NUCLEO needs some slight hardware modifications according to the manufacturer Decawave.**
- **Execution: To actually sniff UWB signals some properties about the signals are needed: The channel, the preamble code and the start of frame delimiter (SFD) used.**
- **Output: Using the sensniff Python script the sniffer reports rx accurate timestamps and received frames**
- **Experiments: We do not apply for the reproducibility badge**
- **How much disk space required (approximately)?: 500KB**
- **How much time is needed to prepare workflow (approximately)?: 2 hours**
- **Publicly available: `https://github.com/seemoo-lab/uwb-sniffer`**
- **Code licenses: MIT License**
- **Archived: `https://github.com/seemoo-lab/uwb-sniffer/releases/tag/v1.0`**

A.3 Description

Our sniffer includes the necessary source code for the UWB board and the host machine to receive and process frames. Furthermore, we include a manual and a YouTube video on how to get started.

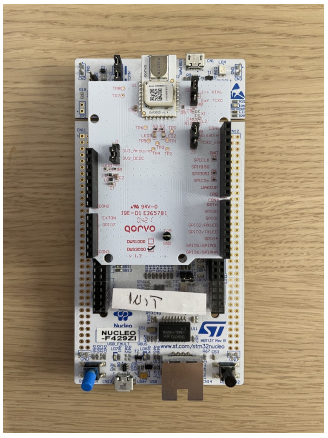

Figure 1: Shows a NUCELO-F429ZI with the DWM3000EVB attached.

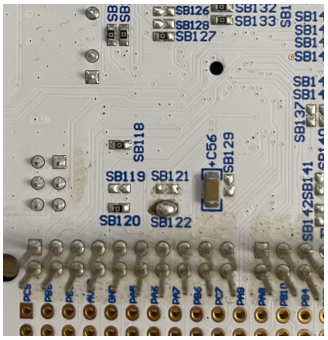

Figure 2: Necessary modifications on the NUCLEO board.

A.3.1 How to access

The artifact is available on GitHub: `https://github.com/seemoo-lab/uwb-sniffer/tree/usenix22-artifact-evaluation`.

Furthermore, we provide a YouTube video that show how to setup the sniffer and how to run it: `https://youtu.be/akCwyHqgbhY`.

A.3.2 Hardware dependencies

To run it we require the NUCELO-F429ZI and the DWM3000EVB attached to the NUCLEO as shown in Figure 1.

The NUCELO-F429ZI needs to be slightly modified to behave correctly when the DWM3000EVB is attached. These modifications are not necessary for nRF boards. Remove solder on SB121 and solder SB122. These modifications are shown in Figure 2.

A.3.3 Software dependencies

We use the STM32CubeIDE to compile and flash the attached NUCELO. We use Python to run a host script that receives input from the UWB Sniffer.

A.3.4 Data sets

N/A

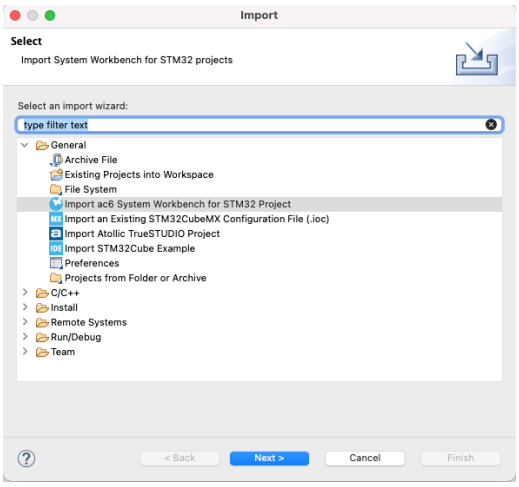

Figure 3: Importing a project in STM32CubeIDE.

A.3.5 Models

N/A

A.3.6 Security, privacy, and ethical concerns

None.

A.4 Installation

Download the STM32CubeIDE and install it: https://www.st.com/en/development-tools/stm32cubeide.html

Download the software samples from Decawave, which include the API to communicate with the DWM3000EVB. Due to license issues, we cannot host it on our GitHub. https://www.decawave.com/wp-content/uploads/2022/03/DW3xxx_XR6.0C_24Feb2022.zip

1. Open a new workspace in the STM32CubeIDE
2. Go to File → Import → General → "Import ac6 System Workbench for STM32 Project" (see Figure 3)
3. Select the root folder of the sample project and import the project
4. Accept to convert the project to the new format
5. Now you can build and run the examples
6. Make sure that the examples build without an error

A.4.1 Integrate the sniffer

1. Copy all source files from this project to the root folder of the DWM3000 sample code
2. Overwrite the main.c with the one in this project
3. Compile the project
4. Run it on an NUCLEO-F429ZI

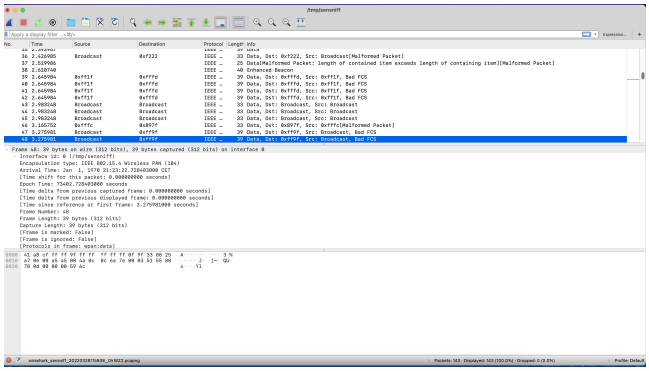

Figure 4: Screenshot of Wireshark with received frames.

A.4.2 Configure the sniffer

UWB has a variety of available configurations: channel, preamble code, data rate, sts mode, and sts length. Most of them have to be known in advance to sniff a communication. In most cases these values can be identified through means of reverse-engineering. For iOS UWB communication, we use the iOS system logs from the nearbyd to identify those values. The values can be changed in the `uwb_sniffer.c` file in the `config` struct.

In the current implementation the sniffer also transfers frames with incorrect headers or frame lengths to the host. So make sure to check the Wireshark output if it is correct. An incorrect configuration leads to long and incorrect frames where the STS or pars of the preamble will be interpreted as data.

A.5 Experiment workflow

With the sniffer any UWB communication following the IEEE802.15.4-2020 HRP standard can be sniffed. This includes frames and accurate timestamps. For this the sniffer needs to be configured as described in Section A.4.2.

With the STM32CubeIDE you can flash the NUCLEO with the attached DWM3000EVB board. When powered on it will immediately start sniffing and trying to forward the packets to the attached computer. To receive the frames on the computer it's necessary to run the provided python script.

```
$ python3 sensniff.py -a
```

The script will try to automatically detect the connected device. If this fails (due to different OS support), we recommend passing the right port directly:

```
$ python3 sensniff.py -d /dev/cu.usbmodem230d
```

Then launch Wireshark on the same computer and add a new pipe at `/temp/sensniff`. When listening to this pipe in Wireshark all frames received by the UWB sniffer will appear here. This includes accurate timestamps when the frame has been received. Figure 4 is a screenshot of a running Wireshark instance that receives UWB frames.

A.6 Evaluation and expected results

In our paper we state that we are able to run the attack Ghostpeak that achieves distance reductions in UWB ranging environments using the modern IEEE 802.15.4z standard. The attack works against the

Apple U1 chip in combination with any other compatible UWB chip (DW3000 from Qorvo/Decawave and SR150 from NXP). We do not publish the attack code, but provide our code for the UWB sniffer, which we used to measure accurate timestamps of the attack and analyze ranging sequences. The sniffer also forwards UWB frames to the host machine which can then display them in Wireshark.

Most U1 to U1 ranging scenarios can be monitored using the sniffer. Depending on the devices used and the ranging sequence in use the sniffer may need to be modified.

A.7 Version

Based on the LaTeX template for Artifact Evaluation V20220119.