Mitchell Sylvia & Samanvay Upadhyay & Jenny Yang

Submission Due: December 18, 2024

Prof. Raza

ECE-371 H

Research Explanation of Timing Errors

Overview:

Our ECE-371H colloquium project involved learning more about possible security vulnerabilities in 5G SIM IoT using its timing latencies in comparison to an NTP server. We used an nRF9160DK development board created by Nordic Semiconductors along with an iBasis IoT SIM as the primary research setup for this project. Primary set-up for the board involved the use of nRFConnect desktop app to ensure proper connectivity of the SIM card to the mobile network. We used Visual Studio Code alongside the nRF Connect for VS Code [1] and nRF Connect for VS Code Extension Pack [2] to flash C programs on the board with a J-link connection to test NTP timings.

Motive and Introduction:

5G-based Internet of Things differs from traditional IoT networks due to its scalable device connectivity and remote applications. 5G IoT's features in Massive Machine-Type Communication allows it to cater to diverse IoT applications ranging from industrial automation to smart homes. These systems need precise synchronization to ensure functionality, especially for time sensitive tasks such as home security and error detection in manufacturing plants. As a result, the reliance of 5G IoT on coordination and communication with a number of devices makes any form of timing errors the most critical security vulnerability. Timing discrepancies can lead to data packet loss, network congestion and desynchronization of devices which allows adversaries to carry out malicious attacks without detection.

Timing attacks can also lead to spoofing since delayed time-stamped messages can compromise the integrity and authenticity of data in IoT applications. This is particularly concerning in 5G IoT's implementation in places like autonomous driving where a timing error can lead to car accidents. Thus, an understanding of how these timing errors propagate across the 5G network in IoT devices is needed to analyze the reliability of this implementation and the viability of security attacks on this vulnerability.

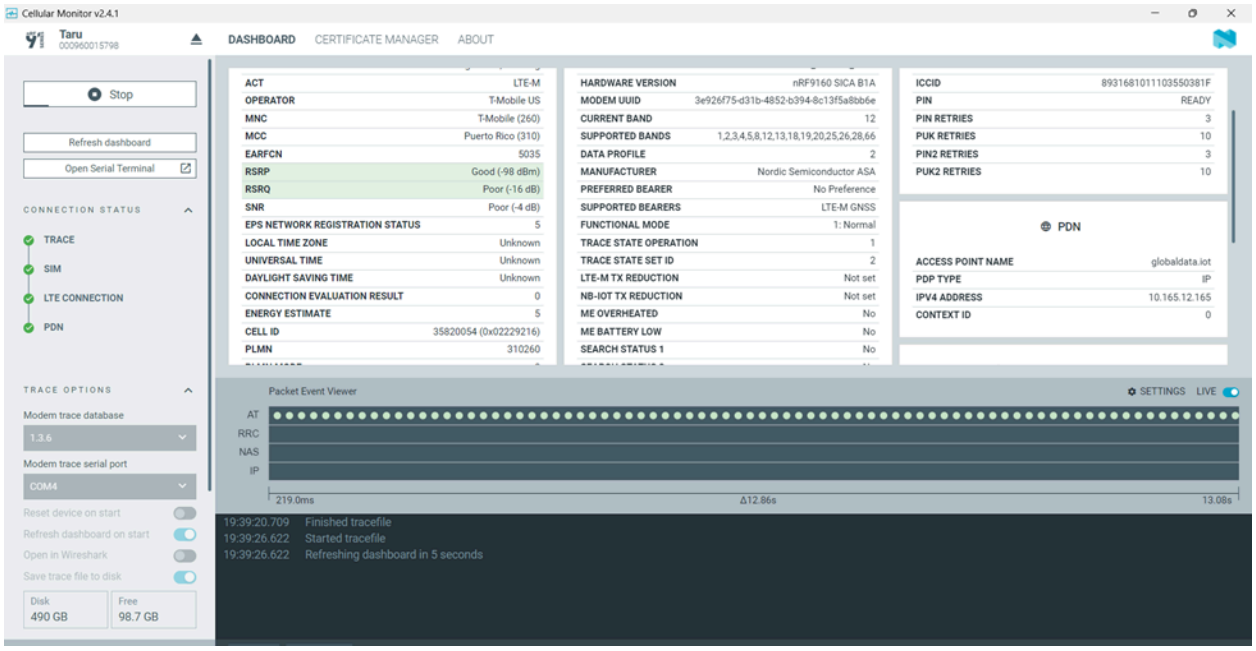
Primary setup:

We use an nRF9160DK development board created by Nordic Semiconductors to test this vulnerability. This board is powered by the nRF9160 SiP which integrates an ARM Cortex M33 application processor, an LTE-M/NB-IoT modem along with GPS and Bluetooth functionalities. The board comes with a built-in J-Link debugger which is used with nRF plug-in for Visual Studio Code to program the device. This board is compatible with the nRF Connect Software Development Kit which provides configuration and monitoring tools for the board. While we were unable to use this functionality, the board also comes with Zephyr RTOS that enables real-time applications on the board. The board includes antennas for LTE and a SIM Card Slot for Cellular Connectivity. We used an iBasis 5G IoT SIM to connect to the mobile network and IoT for this project.

Connection to the Internet:

To get started with the board, we first downloaded the nRF Connect SDK app from the Nordic Semiconductors website [3]. After installing the SDK along with the J-Link Debugger, we connected the board to our desktop using a USB A to Micro B cable. We used the Quick Start menu on the SDK to configure the board. The Quick Start Application first detects the board and then provides a bunch of configuration tools for the board. We kept the default configuration of the board including the 1.3.6 modem trace initially but ensured that the Serial Terminal, Cellular Monitor and nRF Connect for VS Code is installed without any errors.

We faced a minor setback with the iBasis SIM card not being verified by the desktop app but after contacting the support team at iBasis, our issue was promptly fixed. After all issues were resolved, we were able to connect the board to the mobile network using the Cellular Monitor Application in nRF Connect and then used the ‘curl’ command to ping google.com to ensure proper connection to the internet. Additionally, in later tests, we were able to re-confirm connection by flashing application code provided by Nordic as part of their modem firmware installation [10]. This process is covered in the video, and is based on the YouTube tutorial *Unbox and get your Nordic nRF9160 DK up and running* by Nordic Semiconductor [11].



Attempts at Connecting to an NTP server:

Our initial plan was to flash the NTP Pool Project (pool.ntp.org) [4] firmware onto the development board instead of programming it to ensure better reliability of the timings but the development board was not supported by any of the distributions offered by the organization. We also found a date-time implementation that used NTP to provide time in an asset tracker program but its dependencies were not updated in the last five years, leading to failure of this implementation as well. Finally, we tried implementing the SNTP configuration used with the initial date-time program directly in case the dependency errors were caused by other facets of the asset tracker program but discovered that SNTP was no longer supported on Zephyr with the newer version of the asset tracker program removing this component completely.

Troubleshooting - Flashing Code:

The largest struggle when interfacing with this nRF9160DK board has been flashing code onto the board. Here, we will cover all issues we encountered, and how we fixed them.

The first major issue encountered during the setup of this board was connection on MacOS. Nordic Semiconductor offers an install of their nRF Connect Desktop app [3] for Windows, MacOS ARM and Intel, and Linux. Our team had devices on both MacOS ARM and Windows. However, when attempting to run the nRF Connect Desktop app with the nRF9160DK board, we were unable to establish a stable connection, as the board would connect and disconnect on a cycle. We were unable to resolve this issue,

so we instead did all management from Windows devices. This made it more difficult for members who only had an accessible MacOS device to operate the board.

Another major issue with the nRF Connect suite of services was SDK and toolchain management (previously mentioned in the *Connection to the Internet* section). For our experimentation, we were running v5.1.0. After installing the nRF Connect Desktop application, the first step is to run the Quick Start tool. This installs and flashes the correct firmware to the board. During this install process, we also recommend installing the VSCode extensions [1-2]. This is mentioned as part of the standard Quick Start installation. However, the installation documentation as part of the nRF Connect VSCode Extension is incorrect in the order it has you run installs. Below is a numbered list of the order in which the softwares should be installed:

1. J-Link V7.94i [5]: This is a *required* install for any flashing. We specifically only had consistent success with this version, as it was the recommended install by the nRF Connect application.
2. nRF Connect VSCode Extensions [1-2]: If you have not already installed these extensions by this point as part of the Quick Start installation, we recommend you do so now, as it makes finding the next tools much easier.
3. nRF Command-Line Tools [6][1]: After installing the nRF Connect VSCode extensions, a small pop-up should appear in the bottom right with the nRF logo. If this appears red, before clicking on any of the other recommended installs, click the button to install nRF Command-Line Tools. This should be done *before* installing the toolchain or SDK to prevent errors. This is not the order recommended by Nordic Semiconductors, but is the only way we were able to get it to function properly.
4. Toolchain v2.9.0: This was installed as part of the recommendations of the VSCode plugin instructions. This was the latest version as of writing this report.
5. nRF Connect SDK v2.9.0: Similarly to the toolchain, this was also the latest version as of our install

After all of these installs are completed, you may need to restart VSCode for the changes to be matched.

[1] An important note: Nordic Semiconductors is archiving nRF Command-Line Tools, as it will be replaced with nRF Util [6]. For our use case, we utilized nRF Command-Line Tools as installed from the VSCode extensions. We did not download the tool from the link provided.

The next major issue was setting up the code for flashing. For this, we utilized sample code from the *Browse samples* tab on VSCode. More specifically, we utilized the *sntp_client* sample [7][2].

Can Timing Errors be a Vulnerability to Security?:

How can something as simple as unsynchronized cause security vulnerabilities? Imagine you are playing a game where you get 50 tokens to spend every 24 hours. Each token can be used to play a round of a game. The game makers only want a player to play 50 rounds before having to pay money. A player could potentially change their date and time manually in their settings to the next day and get unlimited tokens for free. If a user's device time is not properly synced to the central time server that the game application was using, the user's manual changing of the data would allow them to have unlimited rounds.

We will speculate how timing errors could be used as a side channel attack, gaining access to information by exploiting unintended information leakage.

When transmitting information in Cellular IoT, specifically in 5G, the latest generation of wireless cellular technology 5G, user equipment (UE) are allocated resources from the network. These resources are known as Radio Resource Blocks (RRB). The allocation of these RRBs are scheduled, meaning the timing in which certain users are using their network's resources is ordered. The scheduling information and Resource Blocks (RB) information are encapsulated by Downlink Control Information (DCI) and are sent through the air in plain text. This information, managed in the Physical layer of the OSI model [9], can be eavesdropped and an attacker can find out what resource blocks will be used by the UE [8]. They can then identify the user through the user's Cell Radio Network Temporary Identifier (C-RNTI) which is also sent in plain text. [8]

The target of this attack is information regarding RB, not the payload of the transmitted user data, classifying this a side channel attack. Although we were not able to generate any errors in timing, we can ask the broad question, how could errors in NTP time synchronization pose a vulnerability? Well, many security protocols need accurate timestamps to verify the validity of digital certificates to give the user resources. If a device in cellular IoT is monitored for its timing errors, the attacker can keep a log of this information. The attacker can also seek knowledge of when the user will be allocated resources from the cellular network and see which user it is using C-RNTI. Knowing this information together, they can anticipate when a UE will be denied resources because of an error in time stamps, and they will know when a network will have unallocated resources.

[2] This link (from the nRF Connect VSCode extensions) is a file not found link. This is likely due to it being removed in a more recent branch. However, the installed code locally as part of the nRF Connect VSCode extensions is functional for SDK version 2.8.0.

This is only a potential attack that could be conducted and would require more testing to realize if this is possible.

Conclusion:

Although we could not overcome the roadblock of outdated NTP support for the nRF9160 DK development board, this project gave us significant insights into the intricacies of timing synchronization in 5G IoT systems. We gained valuable hands-on experience with the nRF9160DK, the nuances of configuring 5G IoT systems, and the challenges of working with legacy protocols in new technologies. This outcome underlines the importance of robust, up-to-date support for timing protocols in ensuring the reliability and security of 5G IoT systems. Future research might involve investigating alternative timing synchronization approaches or developing custom solutions for the NTP program.

Additionally, the work we were able to complete sets the foundation for any future research to continue where we left off. Any further research done into NTP with the nRF9160DK board can utilize this foundational experimentation as a point of reference when setting up and programming the board.

Citations:

[1] Nordic Semiconductor, “nRF Connect for VS Code,” *Visual Studio Marketplace*, Dec. 10, 2024. Available: https://marketplace.visualstudio.com/items?itemName=nordic-semiconductor.nrf-connect. [Accessed: Dec. 20, 2024]

[2] Nordic Semiconductor, “nRF Connect for VS Code Extension Pack,” *Visual Studio Marketplace*, Sep. 05, 2024. Available: https://marketplace.visualstudio.com/items?itemName=nordic-semiconductor.nrf-connect-extension-pack. [Accessed: Dec. 20, 2024]

[3] Nordic Semiconductor, “nRF Connect for Desktop,” *Nordic Semiconductor*. Available: https://www.nordicsemi.com/Products/Development-tools/nRF-Connect-for-Desktop/Download. [Accessed: Dec. 20, 2024]

[4] NTP Pool Project, “pool.ntp.org: public ntp time server for everyone,” *NTP Pool Project*. Available: https://www.ntppool.org/en/. [Accessed: Dec. 20, 2024]

[5] Segger, “J-Link / J-Trace Downloads,” *SEGGER*, Dec. 18, 2024. Available: https://www.segger.com/downloads/jlink/. [Accessed: Dec. 21, 2024]

[6] Nordic Semiconductor, “nRF Command Line Tools - Downloads,” *Nordic Semiconductor*. Available: https://www.nordicsemi.com/Products/Development-tools/nRF-Command-Line-Tools/Download. [Accessed: Dec. 21, 2024]

[7] nrfconnect, “sntp_client,” *GitHub*. Available: https://github.com/nrfconnect/sdk-zephyr/tree/v3.7.99-ncs2/c:/ncs/v2.9.0/zephyr/samples/net/sockets/sntp_client/. [Accessed: Dec. 21, 2024]

[8] M.R. Islam, S. Mastorakis, R.H. Anwar, M.T Raza, “Characterizing Encrypted Application Traffic through Cellular Radio Interface Protocol,” IEEE International Conference on Mobile Ad-Hoc and Smart Systems, Massachusetts, 2024, pp.1-3.

[9] C. R. China, “What Is the OSI Model? | IBM,” *IBM*, Jun. 11, 2024. Available: https://www.ibm.com/think/topics/osi-model. [Accessed: Dec. 23, 2024]

[10] Nordic Semiconductor, “nRF9160 DK - Downloads,” *Nordic Semiconductor*. Available: https://www.nordicsemi.com/Products/Development-hardware/nRF9160-DK/Download. [Accessed: Dec. 23, 2024]

[11] Nordic Semiconductor, “Unbox and get your Nordic nRF9160 DK up and running,” *YouTube*, Sep. 16, 2022. Available: https://www.youtube.com/watch?v=Cuf9JPVqTIM. [Accessed: Dec. 23, 2024]