Andrew Mitchell, Bora Kani, Jessie Wang, Lucas Crawshaw
ECE H371
Fall 2024

Research Explanation Regarding Timing Errors Becoming Attacks

Timing errors occur when time synchronization fails, thus causing several detrimental consequences such as data inconsistencies, operational inefficiencies, security vulnerabilities, and increased energy consumption within IoT systems. The most critical effect to consider is how these timing errors can be exploited and turned into various types of attacks. One way this can happen is through timing-based password cracking which is when an attacker can measure the time it takes for a system to reject a password guess, and from this information they can speed up brute-force attacks by inferring specifics on the password length or characters it might use. Another form of these password guessing attacks is through cache-based timing attacks where an attacker could deduce parts of a password by analyzing timing differences in password verification. Denial of service attacks are also possible where attackers can take advantage of timing vulnerabilities in resource allocation mechanisms which might result in resource exhaustion, thus making the system unavailable to legitimate users. Other various attacks such as the bypassing of security controls in systems that rely on timing based checking, or the manipulation of system behavior where attackers could exploit the system to their advantage by carefully timing requests [1]. Timing side-channel attacks can also occur from these timing errors in which sensitive information like encryption keys or secret data can be leaked to an attacker by means of measuring the time taken for cryptographic processes [2]. Attacks specifically related to the Cellular IoT may also occur. Specifically, these could look like network timing vulnerabilities where inconsistencies in time synchronization can lead to breaches where communication is disrupted and malicious data can be inserted into the network. These errors can also open up device specific vulnerabilities where information about its internal state or software implementation can be leaked based on its response to network requests in Cellular IoT. Lastly, cryptographic operations within IoT can reveal sensitive information due to timing differences in the network [3]. These can be mitigated by using secure coding practices, random delays that mask timing patterns, and performing regular security audits [4].

Sources:

[1] Time Base Attacks

[2] Side channel Attacks

[3] IoT Security

[4] Mitigating timing attacks

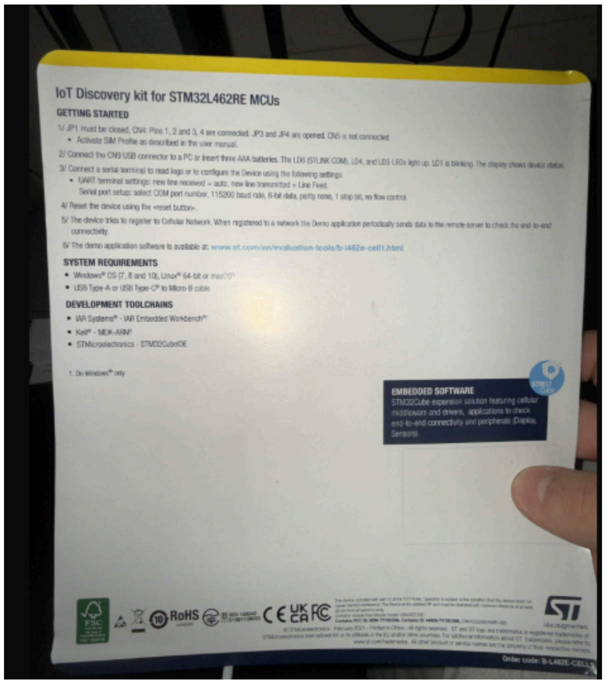
Documentation and Guide: Problems we Faced and How we Solved Them

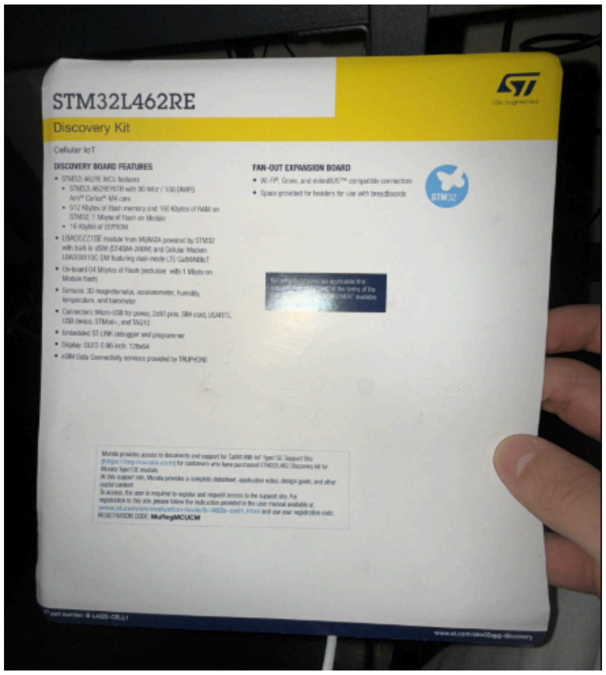
Getting Started:

We had a lot of issues starting this assignment. To begin the assignment, we used a different board than what we ended up completing the assignment with. Unfortunately, the first board did not have a lot of documentation and information on how to access and write to the board. We watched various Youtube videos, tried accessing the board via a serial connection through Putty, and looked at the existing documentation online about this board. In order to try to experiment with this board, we downloaded many suggested drivers and applications such as the MultiTech Connection Manager, the Win32DiskImager and more. None of these applications or the documentation given was able to give us any insight into actually interacting with this board and writing code to it. Because of this, we inevitably gave up on this board and moved to the B-L462E-CELL1 which is another type of Cellular IoT development board. At first interaction, this board seemed to have much more documentation surrounding how to use the board as well as a trusted graduate student who knows who had previously done a project on the board. To begin with this board, we analyzed the package the board came in. The package came with a small description about the board (Figures 1 and 2 below) that gives different specifications about the board and how to get started with it. The package informed us to go to https://www.st.com/en/evaluation-tools/b-l462e-cell1.html ST website in order to learn more about the board. One problem we faced in order to reach this step was the misleading step of trying to find out more information about the board by accessing the MBED website given when you access the board initially. When you go through your file explorer after connecting the board to the computer, it contains the MBED website https://os.mbed.com/platforms/ST-B-L462E-CELL1/ which is meant as an open blog. Unfortunately, there is not much information pertaining on how to use the board or access it. After going to the link provided by the package, we were able to find out more about the board. We discovered here an in depth description of the board, board documentation (https://www.st.com/resource/en/data_brief/b-l462e-cell1.pdf), and the STM32Cube Software (https://www.st.com/en/embedded-software/stm32cubel4.html). The STM32Cube is an IDE that allows you to compile and run code on different STM boards. It has a lot of different uses such as changing the pin functions and layout, changing the timers and clock setups, and running and compiling different code. Once downloaded, the main problem we faced was getting started both with knowing how to use the STM32Cube as well as how to send the code over to the board for it to run. From here, we contacted Murari Yalamanchili , a graduate student that had worked with the board previously, who was able to give us advice and a skeleton code to start us off with in following the next project steps of establishing a connection to the internet and NTP server. He led us to https://www.st.com/en/embedded-software/x-cube-cellular.html which is the software package that we would use to develop the lab. This expansion pack contains the building blocks for cellular interactions. The starting code given runs a program that creates an IP address for the device and is able to find a network server IP address essentially establishing an internet connection. In order to reach this step, we first had to understand how to interact with the STM32Cube to compile this code and send it to the board and what the starter code was doing. Murari was able to help us with these parts of the lab.

Note, you also have to make sure that your board has a SIM card that is registered and running. To do this, we followed the instructions provided on Hologram.io to register our SIM card. It may take up to a few days for your SIM card to be registered and working. You cannot start this project until you have this done.

Figures 1 and 2: B-L462E-CELL1 Description Page





Installing the Cellular Expansion Package on the STM32Cube and Accessing the Board

After having following the links provided in the previous section and downloading and installing the STM32Cube, the Cellular Expansion pack, and Putty (not included in this installation process), the first action item is to be able to get the demonstration of the B-L462E-CELL1 from the downloaded pack into the Cube. The Cube has many files and inclusions that are not necessary to this time synchronization project. The directory that we need is the Nx_TCP_Echo_Client package in the B-L462E-CELL1 folder (Directory Path: en.stm32cubeexpansion-cellular-v7-1-0.zip > STM32CubeExpansion_CELLULAR_V7.1.0 > Projects > B-L462E-CELL1 > Demonstrations > Nx_TCP_Echo_Client). This folder contains drivers and C files for Azure RTOS and NetXDuo that are the main interfaces to use for this project. In NetXDuo, the file app_netxduo.c is the C file that contains the functions to manipulate for this project. Murari Yalamanchili informed our group to understand the functions listed here and to add code to this file, as well as understand the functions listed in app_azure_rtos under the Azure RTOS folder.

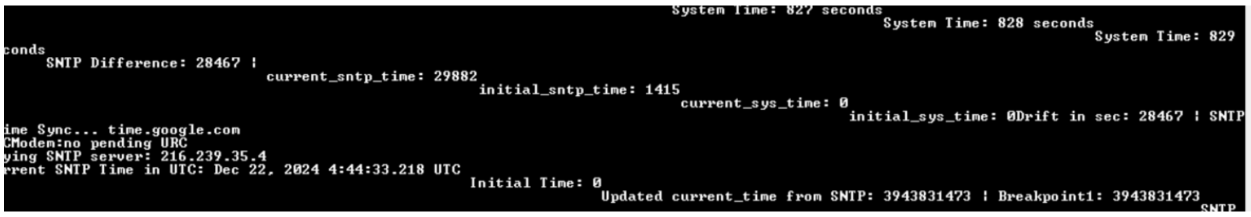
To get this onto the Cube, you first want to open the Cube and have the unzipped expansion pack ready. Have the Cube open as well as the unzipped expansion pack. In the expansion pack, go into and go into STM32CubeIDE from Nx_TCP_Echo_Client (en.stm32cubeexpansion-cellular-v7-1-0.zip > STM32CubeExpansion_CELLULAR_V7.1.0 > Projects > B-L462E-CELL1 > Demonstrations > Nx_TCP_Echo_Client > STM32CubeIDE). Double click on the .cproject. This will import the project into the STM32CubeIDE. If this does not import the project into the STM32Cube, then double click on the .project file. You should then connect your board to your computer using the USB to micro-USB connector. Put the micro-USB connector into the ST-LINK/V2-1 slot on the board. This is so that when the program runs, the Cube will recognize your board and run the programs you compile through to this board. Then, you have to build the program. To do this, click the run button to build the project (the green play button on the taskbar near the top center of the screen). You can see the progress of this on the bottom right corner where it has a progress bar. Once it has finished building. Click the debug button (the button that looks like a bug next to the green play button that you just pressed) to debug the program. Once the debugger builds the program, it will ask you to switch screens, which you should click switch. Then, once in the debugger interface, access the board through Putty. To access the board through Putty, go to Putty, click on Serial (for a serial connection), and type in the COM / port number (which can be found in your device manager under ports), and change the baud rate to 115200. Click OK and it will open the port. Once there, you may need to restart your board by clicking the black “Reset” button on the board. You should be able to see now that the skeleton program is running on your board and it should create an IP address for your board and find the address of an NTP server. It will also try to create a TCP connection which you should not worry about at the moment.

The main code that is running, and that you should understand and change when working through this project is in appnetxduo.c . Here, this contains the functions that you should understand and change.

The difficulty in achieving this process is first finding out this process as well as understanding what is happening in the terminal. It was difficult and took a lot of trial and error before being able to successfully import the project into the Cube as well as figure out that you must debug the program in order to run it. Then, once it’s run, understanding that the board first first establishes necessary connections and then IP addresses took a bit of learning. The next steps were to understand the given skeleton code and implement the C functions to connect to the NTP server and sync the board with the NTP Server.

Implementing Our Code:

We started searching the function names from the skeleton code to figure out the code and that's when we started making headway with our project. I can definitely say we learned a lot through this project. As instructed by Murari, we learned stuff about Azure RTOS. We used this website, https://wiki.st.com/stm32mcu/wiki/Introduction_to_Azure_RTOS_with_STM32 to understand and utilize tx_thread where we were using to do tx_thread_sleep which does time delays for thread execution. We also used tx_semaphore to coordinate between the different and many threads. We also learned a lot about SNTP time synchronization where we used functions to create, initialize, start, update time, and stop and delete clients in appnetxduo. Overall, we were able to talk to and collaborate with the other group as we were having issues together, especially John Dariotis. We came with resolutions together. Originally, we were working on a different function and changing a differ timer, and we had spent days wondering why our outputs were wrong. This is when we emailed Murari about our issues to where he said we were looking at the wrong timer and SNTP function, and we had to maneuver to a different section of the code which we made our progress on. We had issues where the clock system time were 0 (disabled channels) which we were able to change the pin configurations to solve that problem. There was also another issue which we thought was related but our current_sys_time and drift would always output 0 as well where we did the calculations of current sntp subtracting the initial sntp to realize that they were the same variables. This calculation was made in the skeleton code that we were given so we had to edit that to produce actual data.



```
                                                                            System Time: 827 seconds
                                                                                                    System Time: 828 seconds
                                                                                                                            System Time: 829
conds
     SNTP Difference: 28467 ¦
                              current_sntp_time: 29882
                                                      initial_sntp_time: 1415
                                                                             current_sys_time: 0
                                                                                                initial_sys_time: 0Drift in sec: 28467 ¦ SNTP
ime Sync... time.google.com
CModem:no pending URC
ying SNTP server: 216.239.35.4
rrent SNTP Time in UTC: Dec 22, 2024 4:44:33.218 UTC
                                                     Initial Time: 0
                                                                    Updated current_time from SNTP: 3943831473 ¦ Breakpoint1: 3943831473
                                                                                                                                        SNTP
```

With drifts, there’s a bunch of sntp times that keep updating slowly and we see multiple incremental time readings. We needed 1 system time to 1 sntp time but we got multiple system time updates. We would need more time to fix these issues. We were thinking about changing some of the structures of the nested loops that may cause looping or re-updating multiple SNTP times. We could also make it so that the SNTP client only stores 1 time for SNTP and only 1 time for the system time and then measure the drift without the re-syncing aspect. We were also thinking we needed more time as there could be syncing problems with the board.