Project [RFID READER -> LCD] Report

Benny Chao - A01270575
David Lee - A01251357
November 6th, 2022

List of Figures

Overview

RFID stands for radio frequency identification. RFID tags emit radio signals and are embedded in things such as credit cards, passports, merchandise, or even livestock.

RFID tags can be attached to merchandise in a warehouse so that employees can automatically conduct inventories with handheld readers that send data to the company's servers or databases. Users don't have to go back to a terminal to enter data manually.

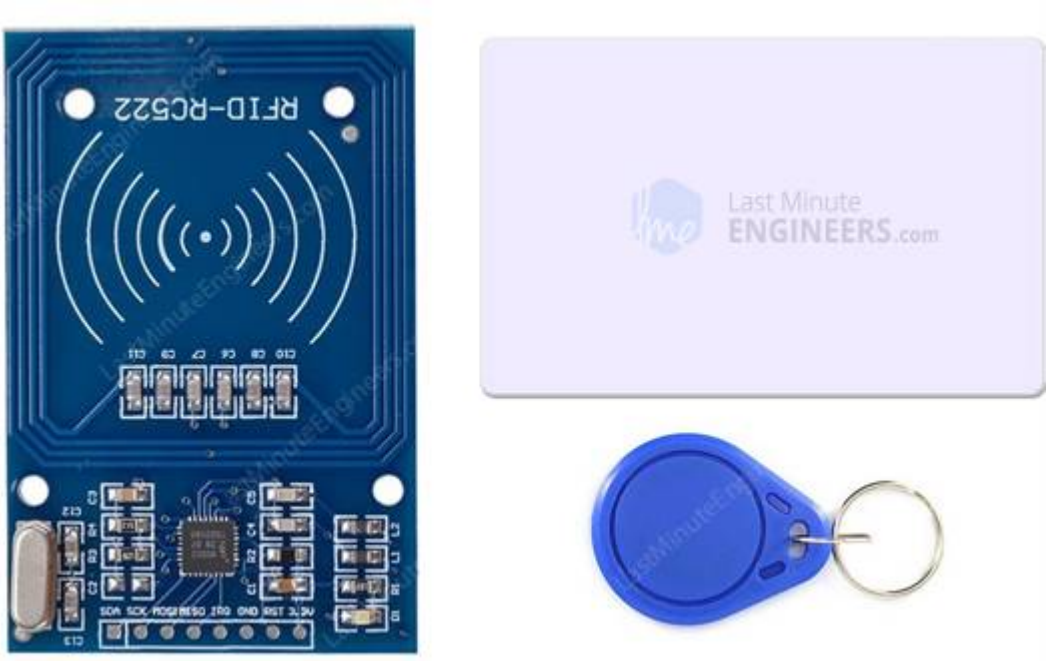

Figure 1. RFID tags (right) and reader (left)

LCD’s (Liquid-Crystal Display), are flat-panel displays that are commonly used in TVs and computer monitors. It is also used in screens for mobile devices, such as laptops, tablets, and smartphones.

Figure 2. 16x2 LCD module

With the prevalence of RFID technology in the real world, we created a project to initialise RFID tags and remotely display the data written to it on an LCD connected to a server.

We have used a reliable UDP (User-Datagram Protocol) as a means of data transmission between the RFID reader module (client), to the LCD (server). A primary concern when using UDP is reliability. To ensure packets aren't lost during communication, the client waits for a response from the server (similar to TCP). This response comes in the form of an ACK (Acknowledgement). When the client receives an ACK from the server, we can then proceed to send another packet. This also prevents packets from arriving out of order by halting data transmission until a lost packet is recovered.

Another concern with UDP is data corruption. To verify packets received from the client, the built-in 16-bit checksum is used to detect multi-bit errors.

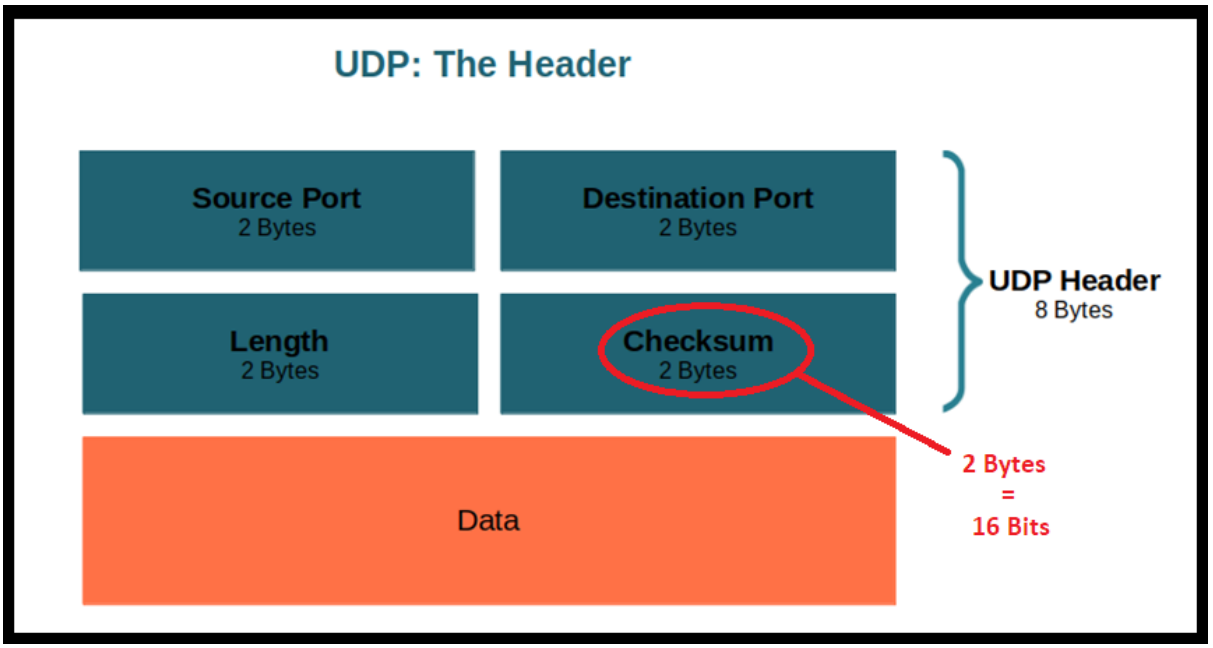

Figure 3. 16-bit checksum as seen in a UDP header

Although RFID technology is widely used and streamlines workflow, they pose a serious security risk. Given the nature of how RFIDs communicate wirelessly, a simple ARP poisoning attack can easily intercept communication between the two devices and potentially steal or corrupt sensitive data. To sniff and capture UDP traffic between the client and server, we have used Ettercap, a free and open source network security tool for MITM (man-in-the-middle) attacks on a LAN (local area network).

Purpose

The client and server work together to form a UDP protocol. In addition to the client and server, we have added hardware from Raspberry PI and the SunFounder Da Vinci Kit. The goal of our project was to create a client that will take input from STDIN and write it to the RFID card provided in the Da Vinci kit. Once the RFID card is tapped, it will send the

message written to the RFID tag to the server. The server will then display the message received on the LCD screen.

Data Gathering

To gather the data for this project we utilised Ettercap and Wireshark to intercept and capture network traffic between a Raspberry Pi acting as a server, and another Raspberry Pi acting as a client. Ettercap and Wireshark were run on a MacBook Air to perform an ARP poisoning attack and capture traffic between the two Raspberry Pi's after successfully spoofing the victim devices' ip addresses.

Requirements

Task	Status
PROTOCOL	
Design the reliable UDP	Fully implemented
The protocol must be reliable	Fully implemented
Design the packet structure	Fully implemented
CLIENT	
Configure client with command line arguments	Fully implemented
Configure client listen for STDIN	Fully implemented
Configure client listen for STDIN redirection (instead of input from a keyboard use a file to input messages)	Fully implemented
Client sends messages through network	Fully implemented
Client waits to receive ACK/NAK	Fully implemented
Client timeout and resend message	Fully implemented
Client send FIN packet	Fully implemented
Sensor (rc522 RFID module) connected to Raspberry Pi	Fully implemented
SERVER	
Configure Server with command line arguments	Fully implemented
Server to listen for messages	Fully implemented
Server to display messages	Fully implemented
Server to send ACK when message is received	Fully implemented

Output device (LCD display) connected to Raspberry Pi	Fully implemented
ADDITIONAL CONSTRAINTS	
Use 2 or more Raspberry Pi computers	Fully implemented
Use your UDP protocol to communicate between the Raspberry Pi computers	Fully implemented
(bonus) more than a simple send-and-wait protocol	Not implemented

Platforms

rfid_to_lcd has been tested on:

- Raspberry Pi OS 3.03-17
- macOS Monterey ver. 12.5

Language

- ISO C17
- Compiles with gcc and clang

Documents

- Design
- Testing
- User Guide

Findings

Client to Server Network Traffic

Current filter: etherip

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.83	192.168.1.78	UDP	1078	43482 → 5050 Len=1036
2	0.003117	192.168.1.83	192.168.1.78	UDP	1078	43482 → 5050 Len=1036
3	4.844885	192.168.1.83	192.168.1.78	UDP	1078	43482 → 5050 Len=1036
4	4.851428	192.168.1.83	192.168.1.78	UDP	1078	43482 → 5050 Len=1036
5	9.896258	192.168.1.83	192.168.1.78	UDP	1078	43482 → 5050 Len=1036
6	9.902186	192.168.1.83	192.168.1.78	UDP	1078	43482 → 5050 Len=1036
7	14.443173	192.168.1.83	192.168.1.78	UDP	1078	43482 → 5050 Len=1036
8	14.444569	192.168.1.83	192.168.1.78	UDP	1078	43482 → 5050 Len=1036
9	24.580624	192.168.1.83	192.168.1.78	UDP	1078	43482 → 5050 Len=1036
10	24.587649	192.168.1.83	192.168.1.78	UDP	1078	43482 → 5050 Len=1036

Figure 4. Network traffic captured from Client to Server using UDP

As seen above when using Ettercap, Wireshark appears to display duplicate packets being sent shortly after each original packet. In actuality, packets sent within milliseconds after its

preceding packet is the result of the attacker's device spoofing MAC or physical address of the source and destination of the victim devices. This effectively places the attacker in the middle of data transmission, hence why it is called a MITM (man in the middle) attack. The attacker forwards packets after initially intercepting them to prevent detection and to continue data transmission between the two Raspberry Pi's. Below is a figure with more detail on the difference between the original and spoofed packet.

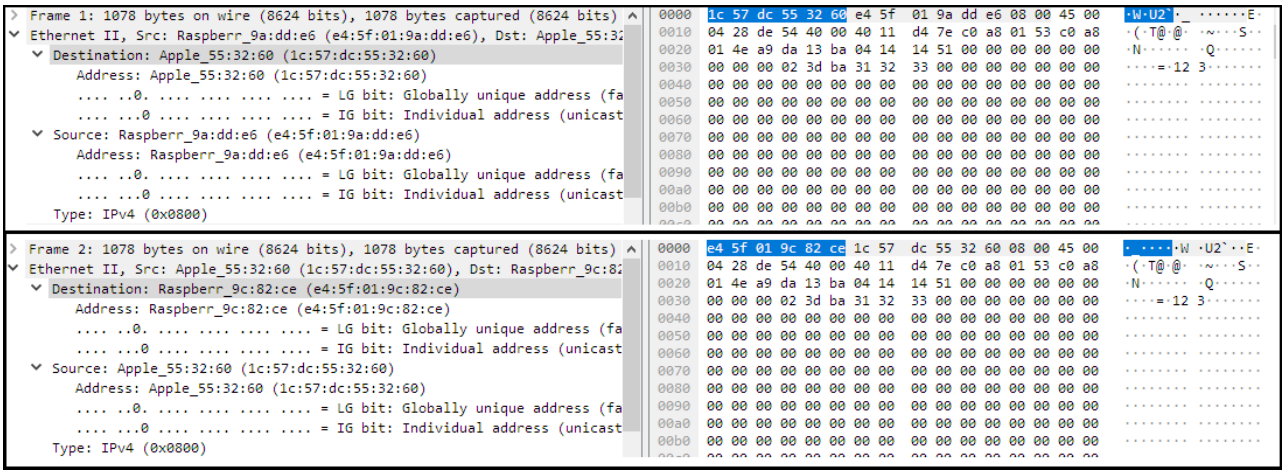

Figure 5. Two similar data packets (original pictured above and spoofed pictured below)

As you can see, both packets contain the same checksum, length, and data. The difference is found in the source and destination ports. For the original packet coming from the client, the source remains the same, but the destination has been changed from the server to the attacking device. The following packet, which is sent by the attacker, has the destination port set to the server to follow through and send the data to its intended device.

Server to Client Network Traffic

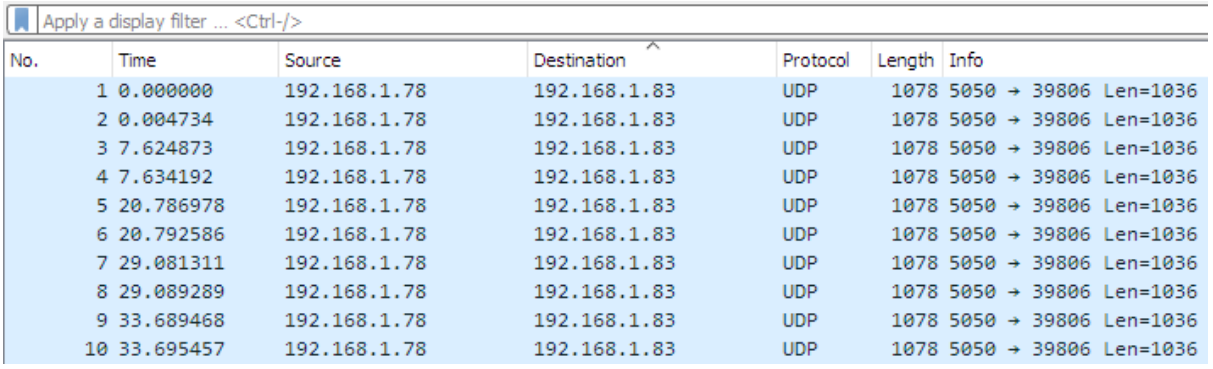

Figure 6. Network traffic captured from Server to Client using UDP

Similarly for acknowledgement packets sent from the server to client, the attacking device also intercepts and spoofs these packets to later send to the client. Figure 6 shows two similar packets captured by Wireshark.

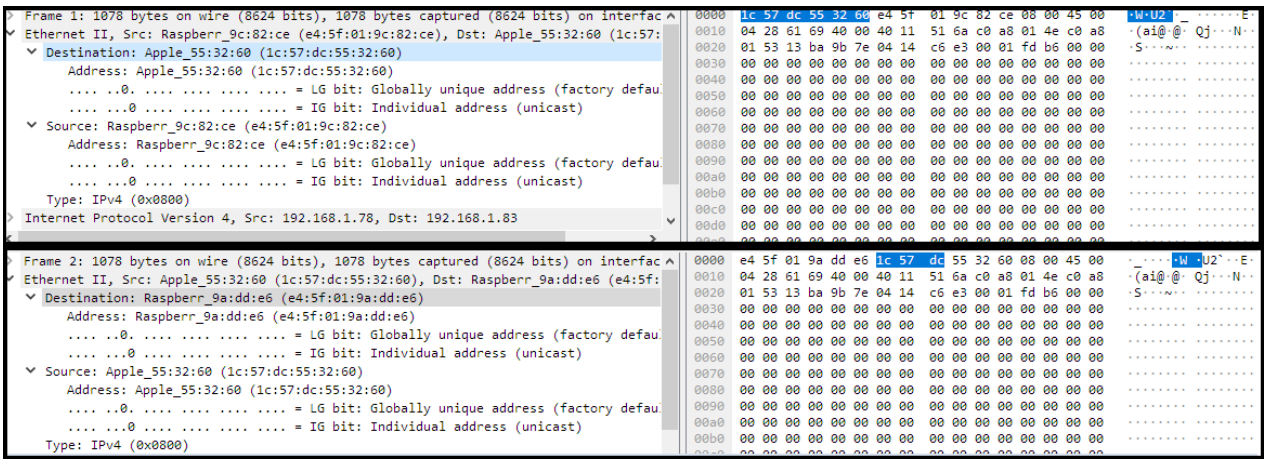

Figure 7. Two similar ACK packets (original pictured above and spoofed pictured below)

Again, both packets contain the same checksum, length, and data. This time the original packet comes from the server, source port remaining the same, but the destination port has been redirected from the client to the attacking device. The following packet, which is sent by the attacker, has the destination port set to the client to follow through and send the ACK packet to its intended device.

Below is a flowchart of UDP traffic captured and intercepted between the two Raspberry Pi's and the attacking machine.

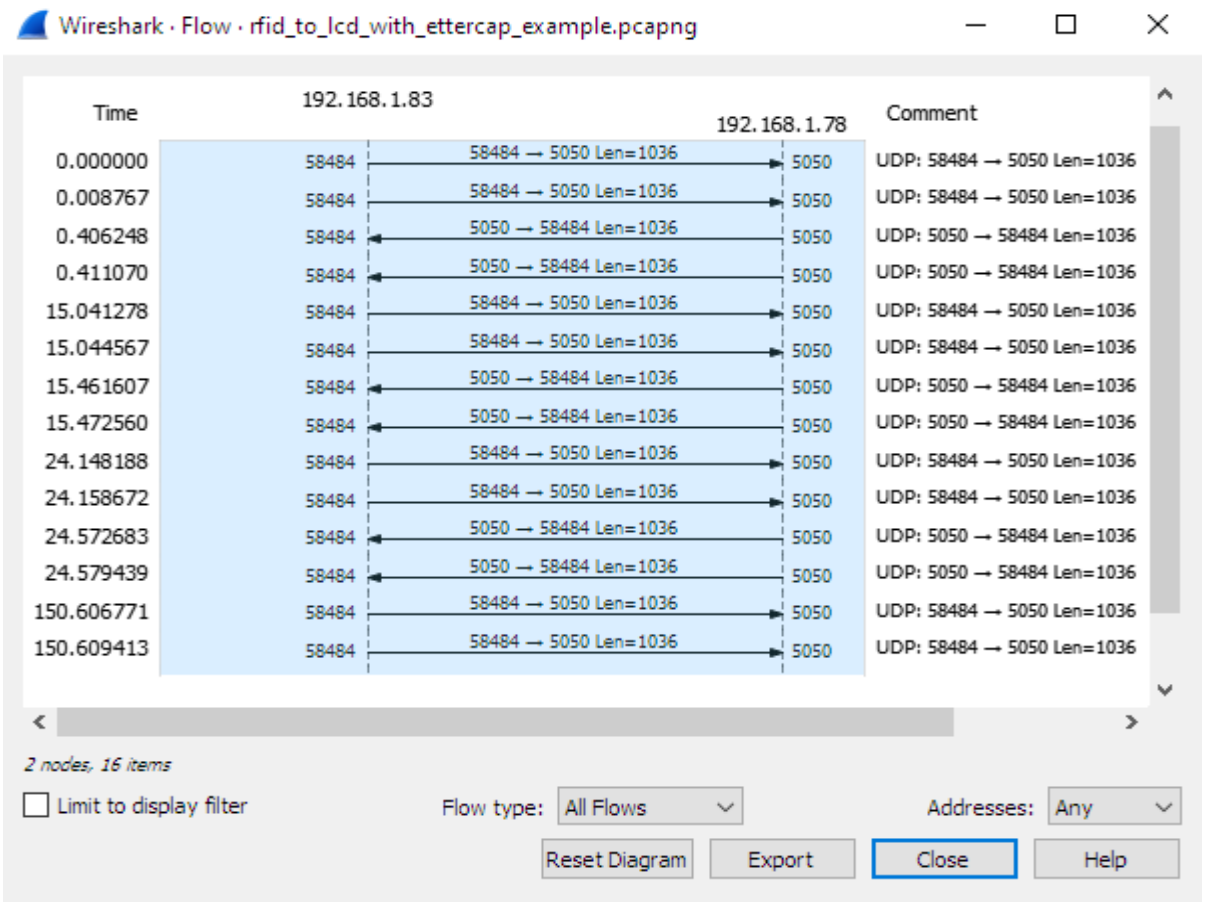

Figure 8. UDP traffic captured w/ Wireshark during a MITM attack

As seen above, communication between the two Raspberry Pi's appears to be normal aside from the presence of duplicate packets. The IP address of both the server (ending in .78) and client (ending in .83) remain the same and the attacker's IP address is never revealed.

Since both the networking and electronic components of the project operate as intended, it is hard to recognize an ARP poisoning attack is taking place. One solution for detection is to use the command "arp -a" (same on Unix and Windows devices). This shows the ARP table of your computer. From here, the attacker can be found when there are two devices with different IP addresses but the same physical address (the attacker being the device associated with the IP address that differs from the intended device's IP address). Another way to recognize an attack is to use Wireshark to detect duplicate packets with different source and destination ports during data transmission.

Writing to and Sending Data From an RFID Tag

RFID cards store data as a sequence of unsigned char. Sending that as the data would result in the server receiving the wrong information to display on the LCD. To fix this issue, we created a helper function to convert data read from the RFID tag upon tapping, into a sequence of char which are stored and sent as the data in a packet from the client side. Figure 8 below shows the helper function used to convert data from the RFID tag into a human readable message.

```
22  void insert_info(const unsigned char *p,int cnt, char* cardData)
23  {
24      int i;
25      for(i=0;i<cnt;i++) {
26          cardData[i] = p[i];
27      }
28      char delim = p[cnt+1];
29      cardData[cnt + 1] = delim;
30  }
31
```

Figure 9. unsigned char* to char* helper function for data conversion

By running a for-loop until all data is read from the RFID tag, we were able to convert each unsigned char to its char counterpart for each iteration of the loop. Finally, we inserted a delimiter to the char buffer as an unsigned char buffer is just a sequence of characters with no delimiter.

After implementing this function, we were able to recognize the data sent from the client to server using Wireshark but the message was being truncated when displayed in the server console as well as the LCD. Therefore, we increased the packet data length by one when passed as a parameter for the strncpy function. Following that, the size of the char buffer that holds the copied data is also increased by one and the final value is set to '\0'. Below is the code detailed as well as code responsible for setting the file descriptor of the LCD and displaying the final message on the liquid-crystal display.

```
// copy data from the client, print it on stdout
strncpy(buffer, packet.data, packet.data_length + 1);
buffer[packet.data_length + 1] = '\0';
fprintf(stdout, "\t%s\n", buffer);
fd = wiringPiI2CSetup(LCDAddr);
init();
writeToLCD(0, 1, buffer);
```

Figure 10. Code responsible for initialising the LCD and copying plus displaying the message