Electric Imp Project Report

Nathan Kim
Professor Glaser
Internet of Things Project Lab Spring 2024

ImpHub Sensor Control Product Explanation

The 'ImpHub Sensor Control' product is an IoT solution inspired by a hierarchical, centralized control model.

- The controller imp acts as the central node or hub that manages and controls multiple minion imps.
- The minion imps are the peripheral nodes or endpoints in the network, each equipped with sensors.
- The controller imp communicates with the minion imps to request sensor readings or send commands.
- The minion imps respond to the controller imp with their sensor readings upon receiving a request.

The network topology is shown in figure 1. The advantage of this solution is that a large number of sensors can be installed, and at different locations, not just the maximum number of sensors that a single imp device can accommodate.

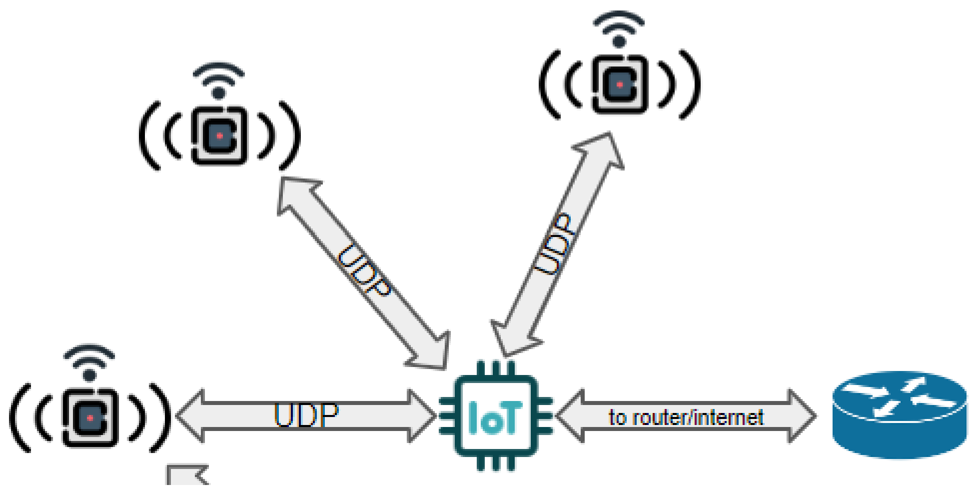


Figure 1. Network Topology of 'ImpHub Sensor Control': central device with multiple connections is the controller imp, other devices are minion imps

The minion imps are devoid of connection to the electric imp cloud, and only listen for requests with their UDP sockets. Realistically, it would be possible to connect the controller imp and minion imps by other things, like MQTT client, or even having each one connect using their own agent.url() webpage, and integrating javascript code into the webpages.

UDP stands for user datagram protocol, and is an internet communication protocol where a connection is not established first before data is sent. It simply sends a packet to its target, and does not wait for confirmation that the packet was received. It has minimal overhead compared to TCP connection, because it does not require maintaining state information or performing error recovery. It is a good protocol for when it is guaranteed that the devices involved in the communication are up and running, and prioritize speed and efficiency.

Selecting UDP as the protocol for this product was a strategic choice, given the proximity of devices and their communication requirements. Unlike TCP, which is better suited for applications like chat servers that require a responsive connection, UDP offers lightweight, connectionless communication. This aligns perfectly with the product's needs, as devices primarily exchange sensor readings without the need for immediate acknowledgment or reliable delivery. With UDP, the focus is on speed and simplicity, ensuring efficient data transmission among devices within the network.

The decision to use UDP was made early in the product's development, and was not changed due to time constraints. In figure 2 is an example of how to start a UDP socket on a device. This code can be installed to any imp device wishing to join the UDP network. The dataReceived function will dictate what the imp device will do upon receiving a UDP packet.

```
function interfaceHandler(state) {
    if (state == imp.net.CONNECTED && udpsocket == null) {
        // We're connected, so initiate UDP
        local ipData = interface.getiptable();
        ucastIP = ipData.ipv4.address;
        bcastIP = ipData.ipv4.broadcast;
        udpsocket = interface.openudp(dataReceived, 1234);
    }
}
```

Figure 2. Code for UDP Socket

The dataReceived function for a controller imp, as implemented in this product, only prints the data to the console. Of course, it can be upgraded to do much more, but for now, it is not used for more than confirming that data has been sent and received.

```
function dataReceived(fromAddress, fromPort, toAddress, toPort, data) {/
        // Log the receipt of data
        server.log("Received: " + data + " length: " + data.len() + " by
}
```

Figure 3. Controller Imp Socket's dataReceived callback function

For a minion imp, the dataReceived function has more functionality, as shown in figure 3. Note line 51, which is responsible for letting the minion communicate back to whatever IP address is passed into the send) function

```
function dataReceived(fromAddress, fromPort, toAddress, toPort, data) {
    // Check that we are the destination (unicast or broadcast)
    if (toAddress == ucastIP || toAddress == bcastIP) {
        // Log the receipt of data
        server.log("Received: " + data + "length: " + data.len() + "bytes from " + format

        data = data.tostring()
        if (data=="Humidity"){
            server.log("reading Humidity")
            data=readHumidity();
        } else if (data=="LED"){
            server.log("toggling LED")
            data=toggleLED();
        } else {
            server.log(data)
        }

        // Echo it back
        local result = udpsocket.send(fromAddress, fromPort, data.tostring());
        if (result != 0) server.error("Could not send the data (code: " + result + ")");
    }
}
```

Figure 3. Minion imp dataReceived code

Product Explanation: Protocol

The protocol the controller imp uses in order to tell the minion to do something:

- Toggle LED: send a UDP packet with a payload of "LED"
- Read Humidity: send a UDP packet with a payload of "Humidity"

This is seen in this code:

```
function readHumidity(option) {
    server.log("wassup");
    local data = "Humidity";
    local result = udpsocket.send("192.168.0.110", 1234, encodedData);
}

function toggleLED(option) {
    server.log("joever");
    local data = "LED";
    local result = udpsocket.send("192.168.0.110", 1234, encodedData);
}
```

The option parameter is not used for anything.

In order to make the controller imp send UDP packets to the minion, interact with the agent's url:

Action	Behavior	Minion Console	Controller Console

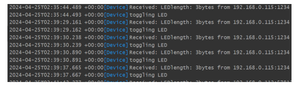
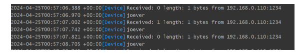
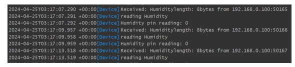
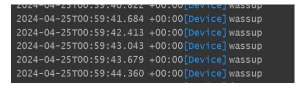
Click "Toggle LED"	LED toggles		
Click	Humidity Reading is taken		

Figure 5. Controller Imp Behavior

Figure 5 shows how to interact with the agent. One can see that in the screenshots of the consoles, the minion device says that it received "LED" or "Humidity". However, the console of the controller is more cryptic, as it says "joever" or "wassup". This is because during the development stage, I thought it would be funny to debug by making the controller imp print to the console "joever" when the "Toggle LED" option is pressed, and then print what it received from the device, and do the same thing for the "wassup" and "Read Humidity" option, but then I forgot to take screenshots after this was fixed.

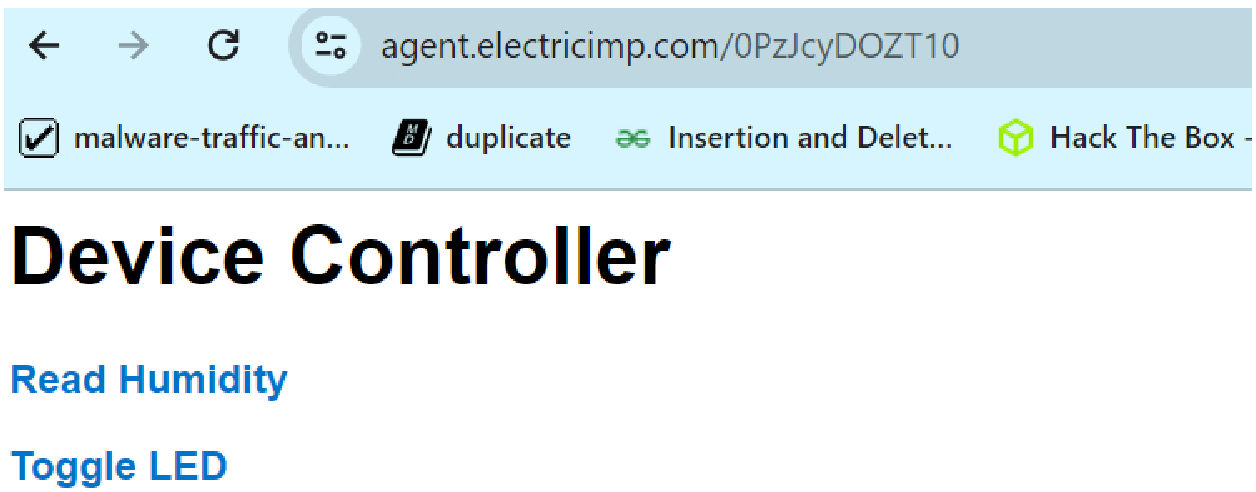


Figure 6. Controller Imp Interface

This concludes the report of the product. The code for the product can be found at https://github.com/skillyskele/IoT-Spring-2024 in the IoT device code folder. The next section explores one of the many vulnerabilities introduced by this solution.

Vulnerability of ImpHub Sensor Control Product

The imp devices have been found to be vulnerable to ARP spoofing. This means that the ARP tables in the devices can have their entries spoofed, and there is no hardware or firmware level preventative measures to detect duplicate entries or any signs of spoofing.

Currently, the imps talk to each other directly, as seen in figure 1. Regardless of what protocol they use to talk to each other, because the communication happens between two vulnerable devices, a middle man can be set up between the two imps, like in figure 7. Later in this report, I will introduce a windows machine, but the connection between the imp and the windows machine is also a vulnerable one, because the windows machine is also vulnerable to arp spoofing, as seen in figure 12. I introduce a windows machine into the demonstration because it is possible to run the command 'arp -a' on it, and see the ARP table printed out, whereas you cannot see the arp tables of the imps.

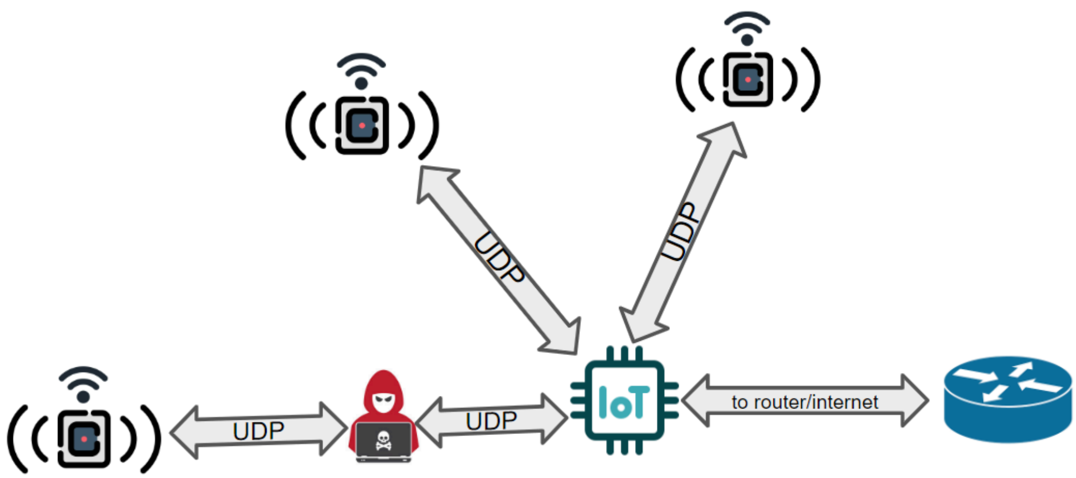


Figure 7: MiTM attack between two IoT devices

So what would a proper solution be? The solution would be that at least one of the two devices that are involved in a connection must be immune to ARP spoofing. That device would be the TP-Link Techno router installed in the Barton 123 room. To communicate with this router first would mean to connect to the actual internet first. Thus, a practical, secure implementation of ImpHub Sensor Control would involve having each minion imp use their own agent, and the agent would send URL requests to and from each other. Every time these requests happen, they go through a secure router, and so a middleman cannot be set up in this secure network, as seen in figure 8.

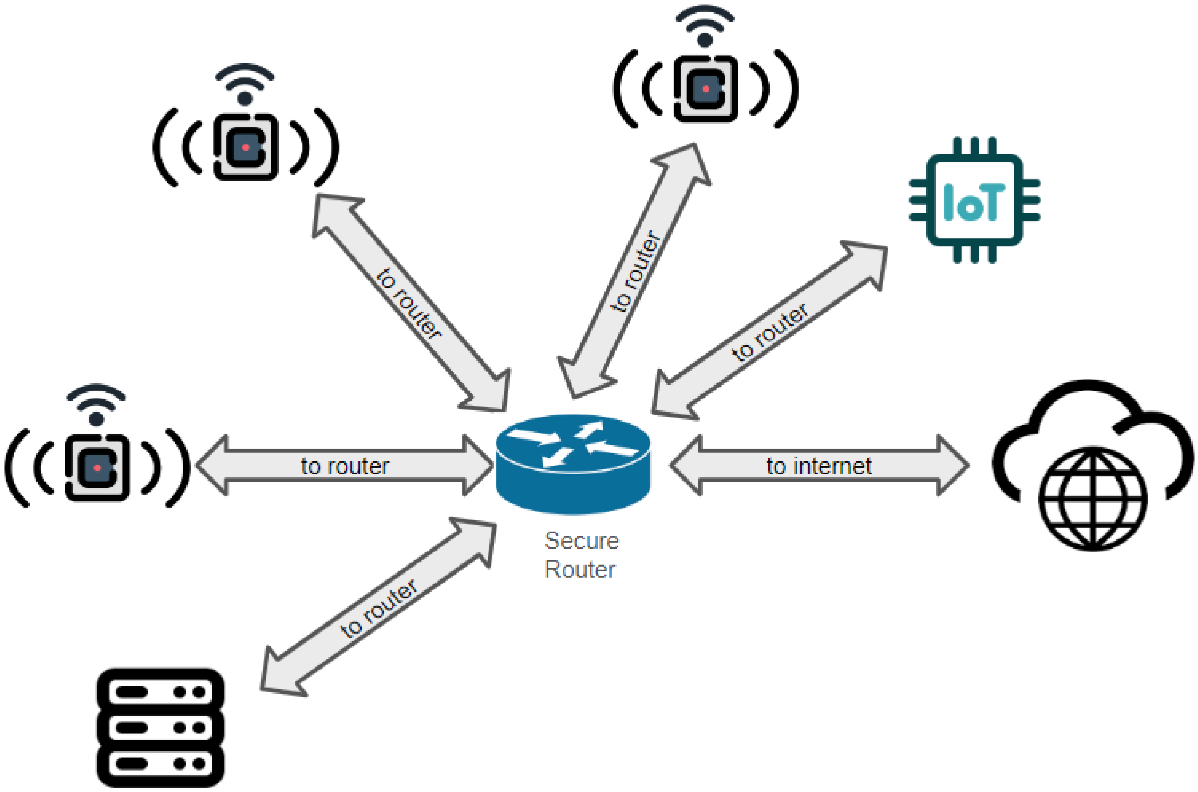


Figure 8: Secure Network Topology

User Story: Server

Let's say that I wanted to let my friends also get the imp device readings. However, I don't want to share my agent.url() with them. This is because anyone who has access to this has access to the IoT device network.

Instead, I decide to have a server running on my computer, that does the exact same thing as the controller imp. It must use the same UDP protocol to talk to the imp device. I will have the server running on port 5000. As of now, here is what the network topology looks like.

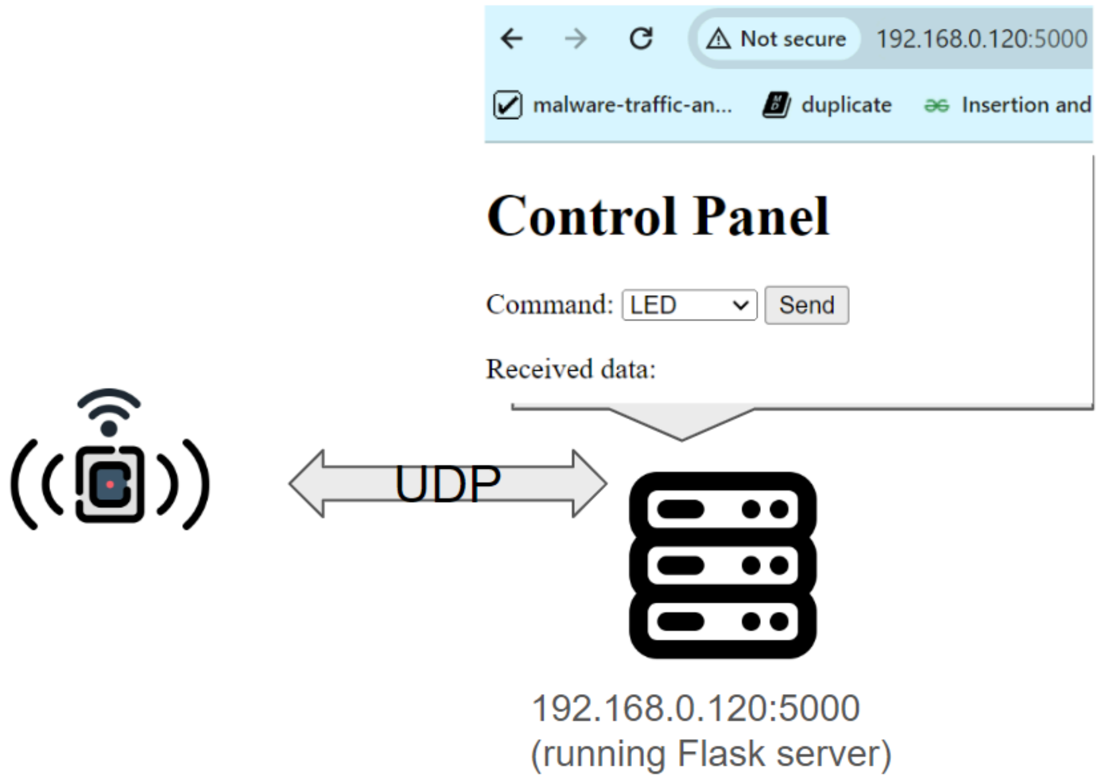


Figure 9. Webpage on 192.168.0.120:5000

Here's an example of it in action:

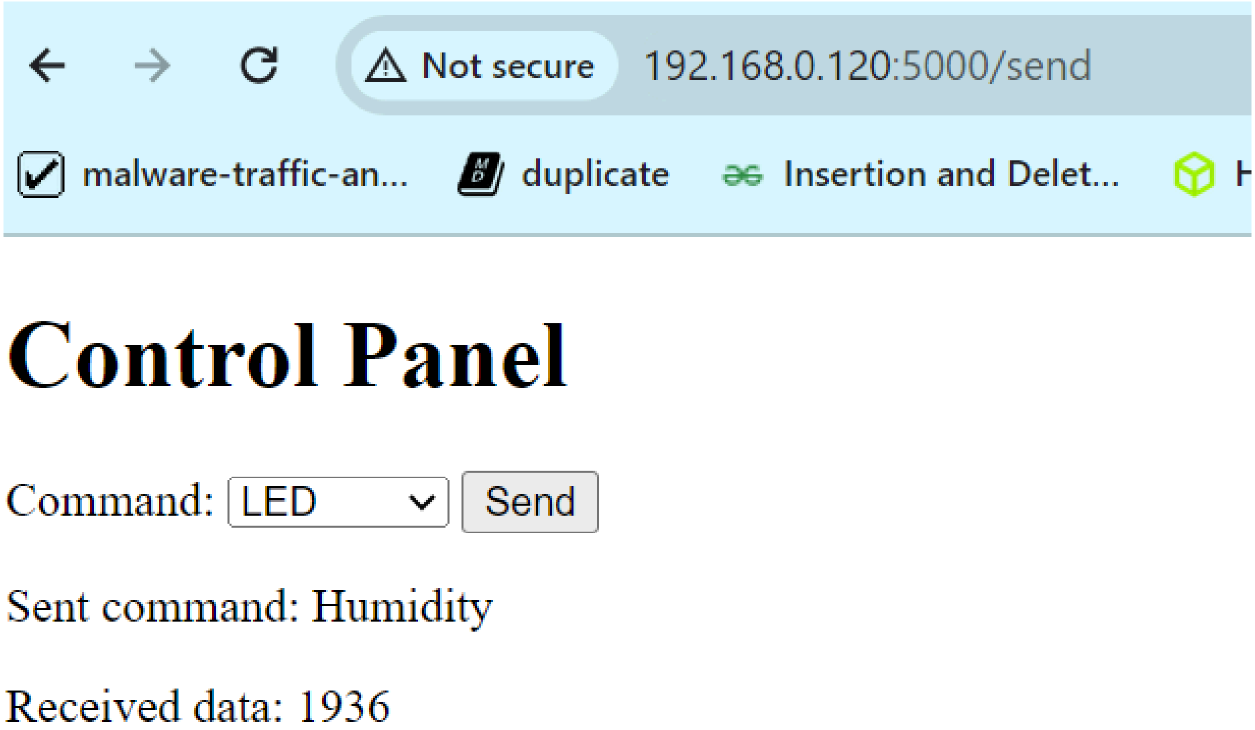


Figure 10. Reading humidity when my finger is on the humidity sensor

Furthermore, to make sure that only trusted people are able to use my server to get sensor readings and turn the LED light on and off, I implemented login functionality:

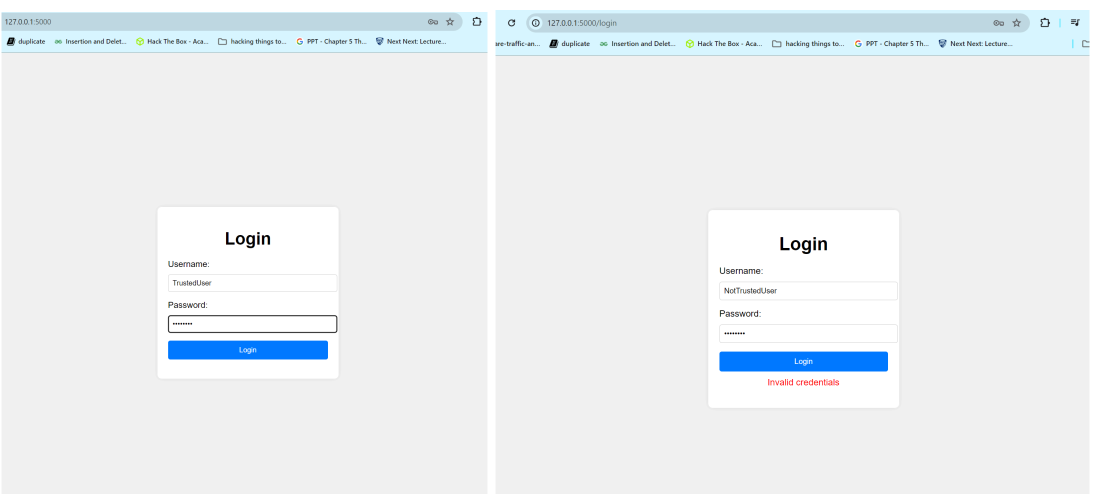
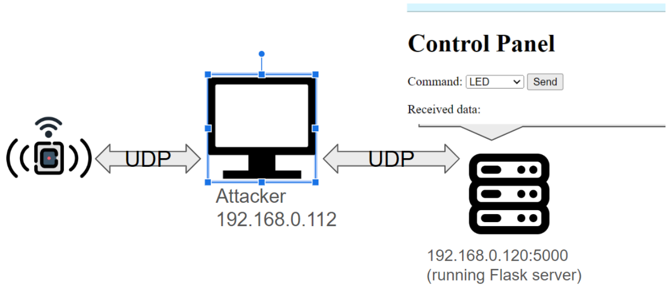


Figure 11. Login to check that a user is a trusted user

This way, only trusted people who log into the server , and I, with the agent.url() of the controller imp, can communicate with the minion imp using UDP.

Hijacking Home Network: ARP Spoofing Edition

How would an attacker be able to communicate with the minion imp, and turn the LED on and off and get sensor readings without knowing the login and password to the server, nor the agent.url()? They would have to ping the device imp directly. However, how would they figure out the protocol? For now it is very simple, as mentioned before, an attacker only needs to send a UDP packet with payload "LED" or "Humidity". But how would the attacker figure that out? Of course, there are a myriad of ways, but one way would be to snoop in on the connection shown in figure 9. This man in the middle attack is shown in figure 10.



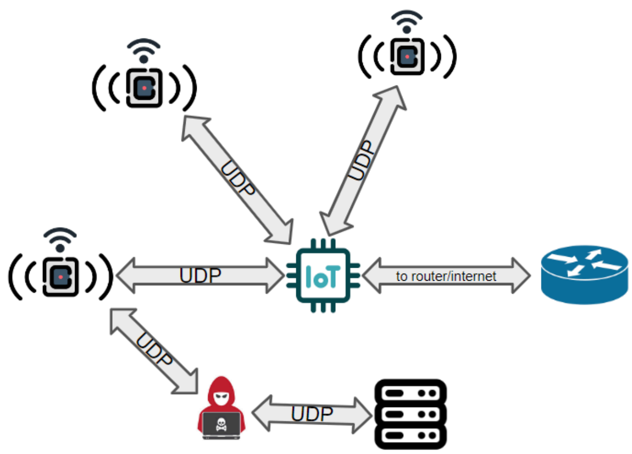


Figure 12. Man in the Middle Placement: between minion imp and Flask server

To make this attack work, there are 4 steps:

- Reconnaissance
    - Know one's own IP and MAC address
    - Discern the IP and MAC address of the device running the Flask server. This is my Windows computer, by the way.
    - Discern the IP address of the minion imp
- ARP Spoofing
    - Tell the minion imp that the IP address 192.168.0.120, or the server, has the MAC address of your attacking machine
    - Tell the server that the IP address 192.168.0.110, or the minion imp, has the MAC address of your attacking machine
    - Now to send a packet from the server to the imp, the server will first send to the attacker

    - Now for the imp to send a packet to the server, the imp will first send to the attacker
- Filter
    - Filter the traffic that has been sent to the attacking machine
    - Look for stuff from the minion to the server, or from the server to the minion
    - The exact filter to type into Wireshark would be: (ip.src==192.168.0.110 && ip.dst==192.168.0.120) || (ip.src==192.168.0.120 && ip.dst==192.168.0.110)
- Final Steps
    - Try controlling the minion imp, thus bypassing all security measures currently in place:
        - login and password
        - being secretive about the agent.url()

<u>Reconnaissance</u>

Alright, let's begin reconnaissance!

```
┌──(kali㉿kali)-[~/Desktop/arpSpoofing]
└─$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.112  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::20c:29ff:feab:f3d5  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:ab:f3:d5  txqueuelen 1000  (Ethernet)
        RX packets 512  bytes 78882 (77.0 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 275  bytes 53573 (52.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 13. ifconfig is a linux command to see internet configuration

A simple 'ifconfig' command returns that the attacker machine is on IP address 192.168.0.112, and its mac address is 00:0c:29:ab:f3:d5.

```
┌──(kali㉿kali)-[~/Desktop/arpSpoofing]
└─$ arp -a | grep -v '<incomplete>'

? (192.168.0.110) at 0c:2a:69:05:36:3f [ether] on eth0
? (192.168.0.115) at 0c:2a:69:11:d3:32 [ether] on eth0
? (192.168.0.1) at ac:84:c6:b0:74:0c [ether] on eth0
? (192.168.0.102) at 16:a6:44:93:7c:e2 [ether] on eth0
? (192.168.0.120) at a4:6b:b6:41:74:c7 [ether] on eth0
```

Figure 14. arp -a is a linux command to see the ARP table

The 'arp -a' command prints the ARP table, which maps IP addresses to MAC addresses. I see a couple, IP addresses. Let's figure out what each of them are. As of now, I don't know which IP addresses correspond to which device yet.

```
┌──(kali㉿kali)-[~/Desktop/arpSpoofing]
└─$ sudo nmap -sS -sV -O -T4 192.168.0.110 192.168.0.115 192.168.0.1 192.168.0.102 192.168.0.120
```

Figure 15. nmap is a tool for probing computer networks

A great tool to use in this situation is Nmap. Nmap will scan a network. Here, I ask Nmap to scan a specific list of IP addresses-the ones found in my ARP table.

- The '-sS' option is a TCP SYN option, meaning it will send TCP SYN packets.
- The 'sV' option is an option that asks Nmap to learn, if possible, what service is running on the machine.
- The '-O' option asks Nmap to learn what operating system is being used, if possible.
- The '-T4' option tells Nmap to scan at speed of 4 on a scale of 1 to 5.

Let's see the results:

```
[sudo] password for kali:
Starting Nmap 7.94 ( https://nmap.org ) at 2024-04-25 14:26 EDT
Nmap scan report for 192.168.0.110
Host is up (0.015s latency).
All 1000 scanned ports on 192.168.0.110 are in ignored states.
Not shown: 1000 closed tcp ports (reset)
MAC Address: 0C:2A:69:05:36:3F (electric imp, incorporated)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Aggressive OS guesses: 2N Helios IP VoIP doorbell (96%), Advanced Illumination DCS-100E lighting controller
a logger (96%), Daysequerra M4.2SI radio (96%), Denver Electronics AC-5000W MK2 camera (96%), DTE Energy Br
8266 firmware (lwIP stack) (96%), Espressif ESP8266 WiFi system-on-a-chip (96%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop

Nmap scan report for 192.168.0.115
Host is up (0.014s latency).
All 1000 scanned ports on 192.168.0.115 are in ignored states.
Not shown: 1000 closed tcp ports (reset)
MAC Address: 0C:2A:69:11:D3:32 (electric imp, incorporated)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Aggressive OS guesses: 2N Helios IP VoIP doorbell (96%), Advanced Illumination DCS-100E lighting controller
a logger (96%), Daysequerra M4.2SI radio (96%), Denver Electronics AC-5000W MK2 camera (96%), DTE Energy Br
8266 firmware (lwIP stack) (96%), Espressif ESP8266 WiFi system-on-a-chip (96%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop

Nmap scan report for 192.168.0.1
Host is up (0.010s latency).
Not shown: 996 closed tcp ports (reset)
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     Dropbear sshd 2012.55 (protocol 2.0)
53/tcp   open  domain  (generic dns response: NOTIMP)
80/tcp   open  http    TP-LINK TD-W8968 http admin
1900/tcp open  upnp    Portable SDK for UPnP devices 1.6.19 (Linux 2.6.36; UPnP 1.0)
1 service unrecognized despite returning data. If you know the service/version, please submit the following
SF-Port53-TCP:V=7.94%I=7%D=4/25%Time=662AA064%P=x86_64-pc-linux-gnu%r(DNSV
SF:ersionBindReqTCP,2D,"\0\+\0\x06\x85\0\0\x01\0\x01\0\0\0\0\x07version\x0
SF:4bind\0\0\x10\0\x03\xc0\x0c\0\x10\0\x03\0\0\0\0\0\x01\0")%r(DNSStatusRe
SF:questTCP,E,"\0\x0c\0\0\x90\x04\0\0\0\0\0\0\0\0");
MAC Address: AC:84:C6:B0:74:0C (TP-Link Technologies)
```

Figure 16. nmap scan output

Looks like 192.168.0.110 and 192.168.0.115 seem to be some electronic devices. However, Nmap could not figure out exactly what they were. As the one who is in control of the project, I know that those are my minion and controller imp nodes, respectively. Nmap does a decent job of guessing that those are some system-on-chip devices.

The 192.168.0.1 IP address is the router, as the first IP address of a network is almost always saved for the router. I can also search up TP-Link Technologies to find out that it is a wifi router company.

Finally, let's look at 192.168.0.120:

```
Nmap scan report for 192.168.0.120
Host is up (0.00034s latency).
Not shown: 998 filtered tcp ports (no-response)
PORT      STATE SERVICE            VERSION
5000/tcp open  upnp?
6881/tcp open  bittorrent-tracker?
```

Figure 17. Probe results for 192.168.0.120

Looks like it's running services on ports 5000 and 6881. What are these ports used for? Google is often a hacker's best friend:

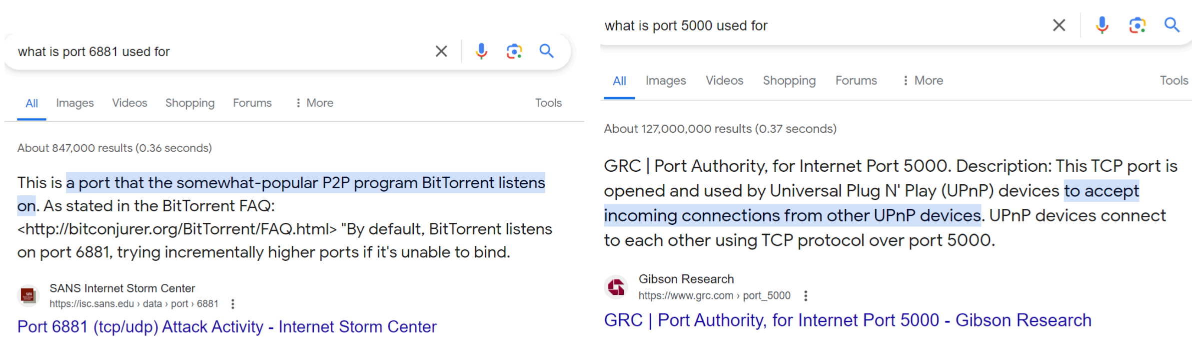


Figure 18. Common port usages

Looks like port 5000 is the most promising. Over to the browser we go!

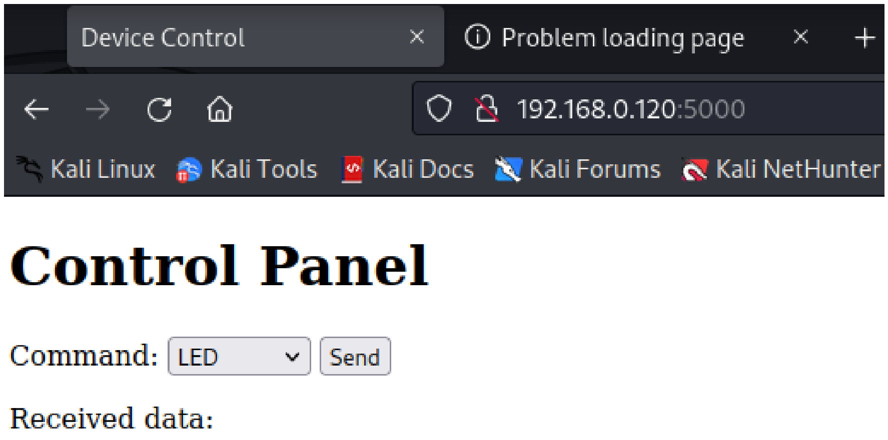


Figure 19. Discover that 192.168.0.120 is a webpage

Looks like our server is being hosted at 192.168.0.120:5000. As we saw from the 'arp -a' command (in which I filtered out the incomplete entries, but the 'grep' command is not needed), the server, 192.168.0.120, seems to have mac address a4:6b:b6:41:74:c7.

<u>ARP Spoofing</u>

The arpSpoof.py code is here: https://github.com/skillyskele/IoT-Spring-2024
The flask server code is up there too, as it uses app.py and the templates folder.
All the agent and device code written in squirrel is up there too.

Let's begin the attack:

```
┌──(kali㉿kali)-[~/Desktop/arpSpoofing]
└─$ sudo python3 arpSpoof.py
Enter the target IP address (default is 192.168.0.100): 192.168.0.120
Enter the attacker MAC address (default is 00:0c:29:ab:f3:d5):
^C
ARP spoofing stopped by user.
```

Figure 20. run arpSpoof.py, enter target IP and attacker's (your) MAC Address

The spoofing should have gone through. I can verify this by going over to the victim device, and looking at the ARP table. From my windows machine, I see:

```
Interface: 192.168.0.120 --- 0xf
  Internet Address      Physical Address      Type
  192.168.0.1           ac-84-c6-b0-74-0c     dynamic
  192.168.0.110         0c-2a-69-05-36-3f     dynamic
  192.168.0.112         00-0c-29-ab-f3-d5     dynamic
  192.168.0.255         ff-ff-ff-ff-ff-ff     static
```

Figure 21. Before ARP spoofing

But after the spoofing attack, I see:

```
Interface: 192.168.0.120 --- 0xf
  Internet Address      Physical Address      Type
  192.168.0.1           ac-84-c6-b0-74-0c     dynamic
  192.168.0.110         00-0c-29-ab-f3-d5     dynamic
  192.168.0.112         00-0c-29-ab-f3-d5     dynamic
  192.168.0.255         ff-ff-ff-ff-ff-ff     static
```

Figure 22. After ARP spoofing

The windows machine thinks the minion imp's mac address is the mac address of the attacker.

One can further verify that the attack worked by going to the victim's computer, and looking at its internet traffic with wireshark. By filtering for arp packets, one can multiple duplicate uses of an IP address, as one IP address is claiming to be multiple devices, as it claims to have multiple mac addresses:

```
Ethernet II, Src: electricimp_05:36:3f (0c:2a:69:05:36:3f), Dst: DESKTOP-058GEMV.local (a4:6b:b6:41:74:c7)
Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: electricimp_05:36:3f (0c:2a:69:05:36:3f)
    Sender IP address: 192.168.0.110 (192.168.0.110)
    Target MAC address: DESKTOP-058GEMV.local (a4:6b:b6:41:74:c7)
    Target IP address: it-pclt-tlsm.press.jhu.edu (192.168.0.112)
[Duplicate IP address detected for 192.168.0.110 (0c:2a:69:05:36:3f) - also in use by 00:0c:29:ab:f3:d5 (frame 692)]
[Duplicate IP address detected for 192.168.0.112 (a4:6b:b6:41:74:c7) - also in use by 00:0c:29:ab:f3:d5 (frame 686)]
```

Figure 23. Wireshark notices duplicate usages of IP address

Filter

Okay, now back to pretending we're the attacker, and that we can't see the victim's computer. As the attacker, I have wireshark open, so that I can monitor traffic on my own computer. Once I run arpSpoof.py, I should be watching for traffic that goes:

- From the server to my computer to the minion imp
- From the minion imp to my computer to the server

This is shown below:

(ip.src==192.168.0.110 && ip.dst==192.168.0.120) || (ip.src==192.168.0.120 && ip.dst==192.168.0.110)S

.	Time	Source	Destination	Protocol	Length	Info	
3481	106.380587575	192.168.0.120	192.168.0.110	UDP	60	55578 → 1234 Len=3	
3482	106.380606100	192.168.0.120	192.168.0.110	UDP	45	55578 → 1234 Len=3	
3483	106.387116972	192.168.0.110	192.168.0.120	UDP	60	1234 → 55578 Len=1	
3484	106.387153749	192.168.0.112	192.168.0.110	ICMP	71	Redirect	(Redirect for host)
3485	106.387248701	192.168.0.110	192.168.0.120	UDP	43	1234 → 55578 Len=1	
3501	109.107405455	192.168.0.120	192.168.0.110	UDP	60	51844 → 1234 Len=3	
3502	109.107423473	192.168.0.120	192.168.0.110	UDP	45	51844 → 1234 Len=3	
3503	109.116457110	192.168.0.110	192.168.0.120	UDP	60	1234 → 51844 Len=1	
3504	109.116483224	192.168.0.112	192.168.0.110	ICMP	71	Redirect	(Redirect for host)
3505	109.116551515	192.168.0.110	192.168.0.120	UDP	43	1234 → 51844 Len=1	
3550	110.465461443	192.168.0.120	192.168.0.110	UDP	60	62922 → 1234 Len=3	
3551	110.465479454	192.168.0.120	192.168.0.110	UDP	45	62922 → 1234 Len=3	
3552	110.471843182	192.168.0.110	192.168.0.120	UDP	60	1234 → 62922 Len=1	
3553	110.471874006	192.168.0.112	192.168.0.110	ICMP	71	Redirect	(Redirect for host)
3554	110.471979709	192.168.0.110	192.168.0.120	UDP	43	1234 → 62922 Len=1	
3574	114.930735284	192.168.0.120	192.168.0.110	UDP	60	55303 → 1234 Len=8	
3575	114.930771015	192.168.0.120	192.168.0.110	UDP	50	55303 → 1234 Len=8	
3580	114.938091172	192.168.0.110	192.168.0.120	UDP	60	1234 → 55303 Len=1	

Figure 24. Filtering wireshark traffic

I can see that they are communicating using UDP, and that 192.168.0.110 is using port 1234. Now let's see what the payload is:

Figure 25. UDP packet sending "LED"

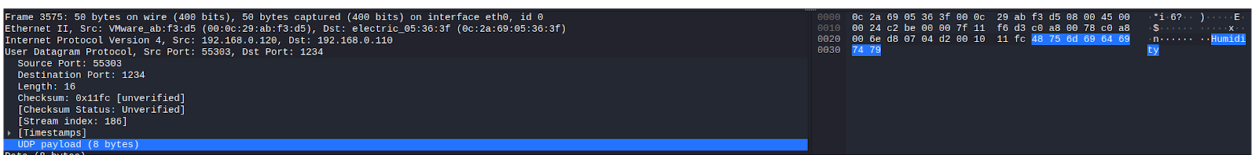


Figure 26. UDP packet sending "LED"

Looks like it's just sending "LED" and "Humidity" in plaintext.

<u>Final Steps</u>

Okay, let's try using the attacker machine to directly control the minion imp! I will construct a UDP packet, and listen for the response using the handy linux tool, socat:

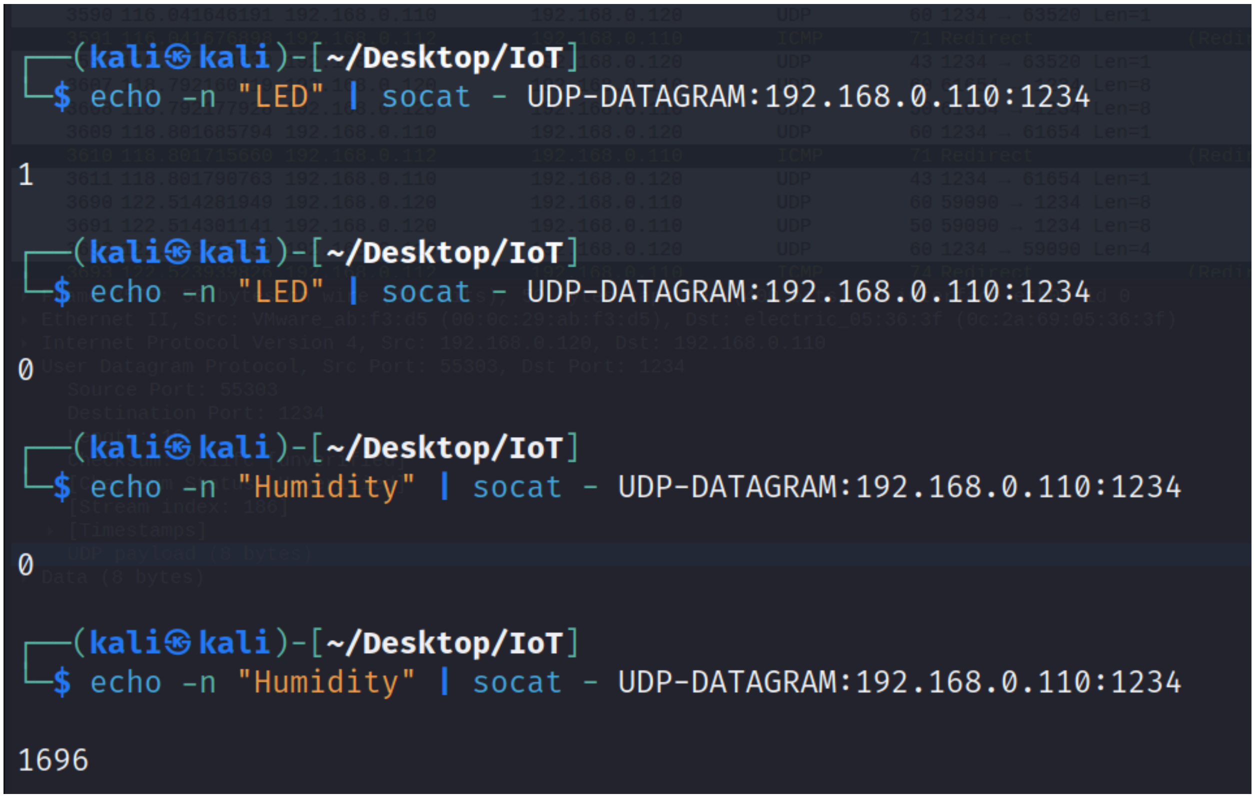


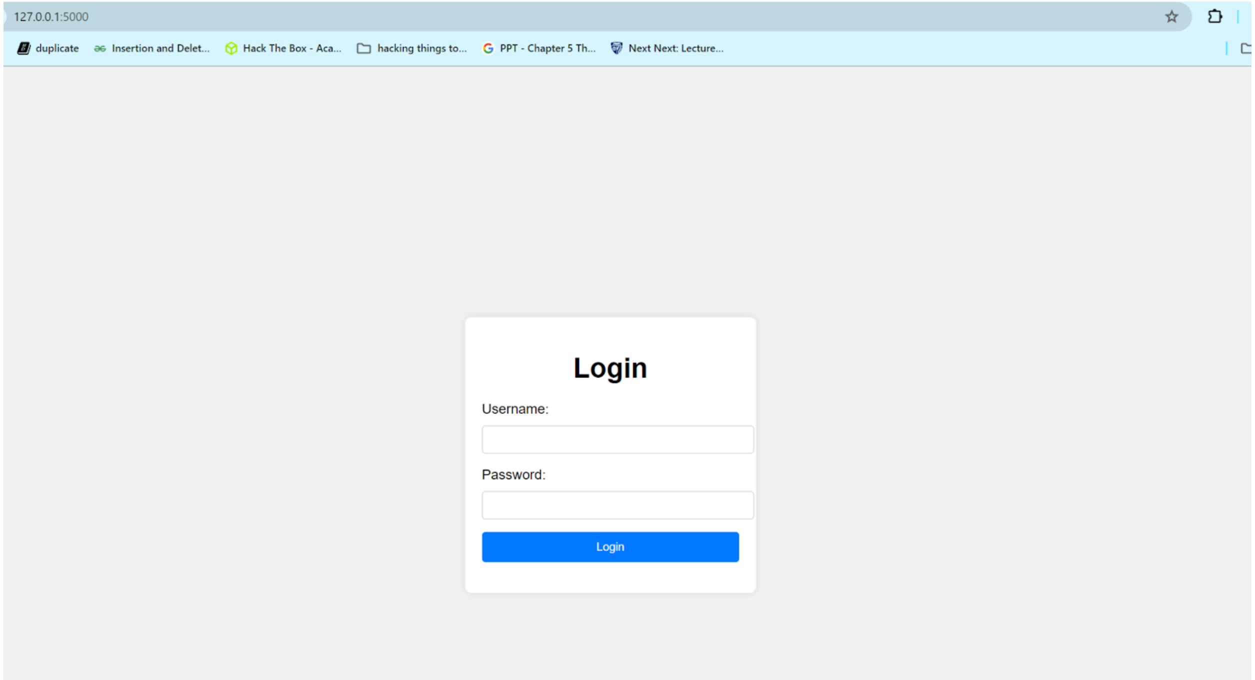
Figure 27. socat can send UDP packets

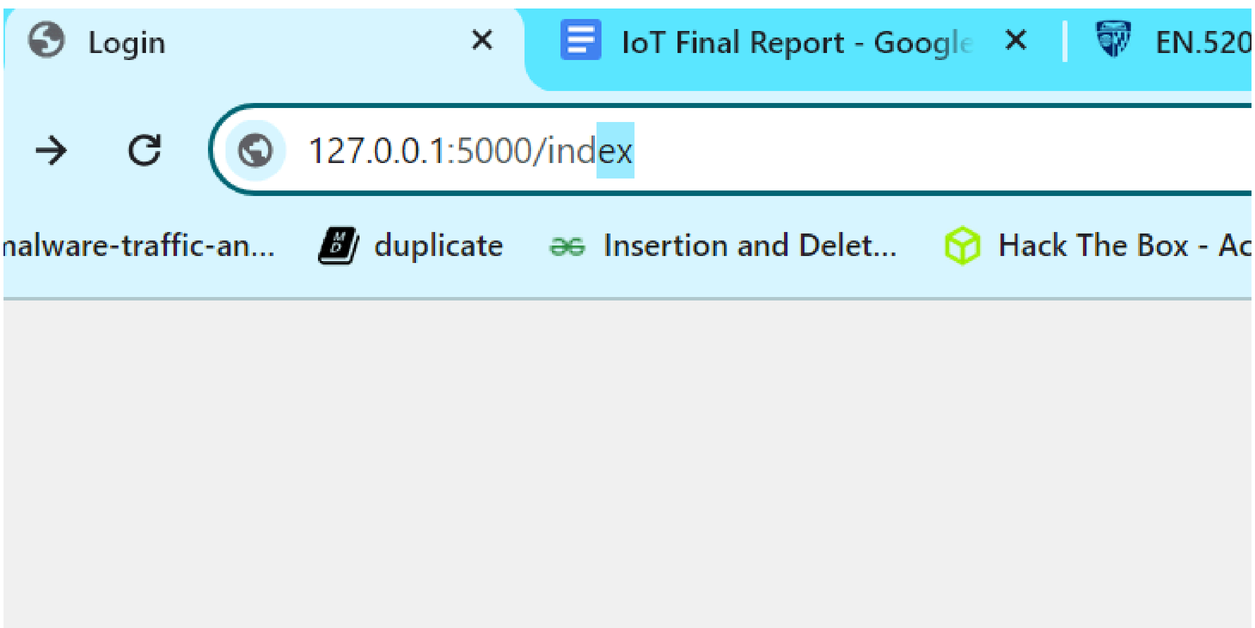
You can see that the LED gets toggle on and off, and the humidity was 0, and then 1696, without having the agent url, nor having login access to the server.

Moving Forward

The reason that the network was vulnerable was because I implemented my own communication protocol, using UDP. The network could be better secured by encrypting the UDP messages in something other than plaintext.

A couple other obvious vulnerabilities would be hiding the username and password in plain text on the login.html page, having extremely weak username and password credentials (TrustedUser and password, by the way), and directory traversal. A quick demo of directory traversal is shown in figure 28 . Instead of typing in the username and password, get to the index.html page by simply typing it into the URL.


127.0.0.1:5000
duplicate
Insertion and Delet...
Hack The Box - Aca...
hacking things to...
PPT - Chapter 5 Th...
Next Next: Lecture...
Login
Username:
Password:
Login


Login
IoT Final Report - Google
EN.520
127.0.0.1:5000/index
nalware-traffic-an...
duplicate
Insertion and Delet...
Hack The Box - Ac

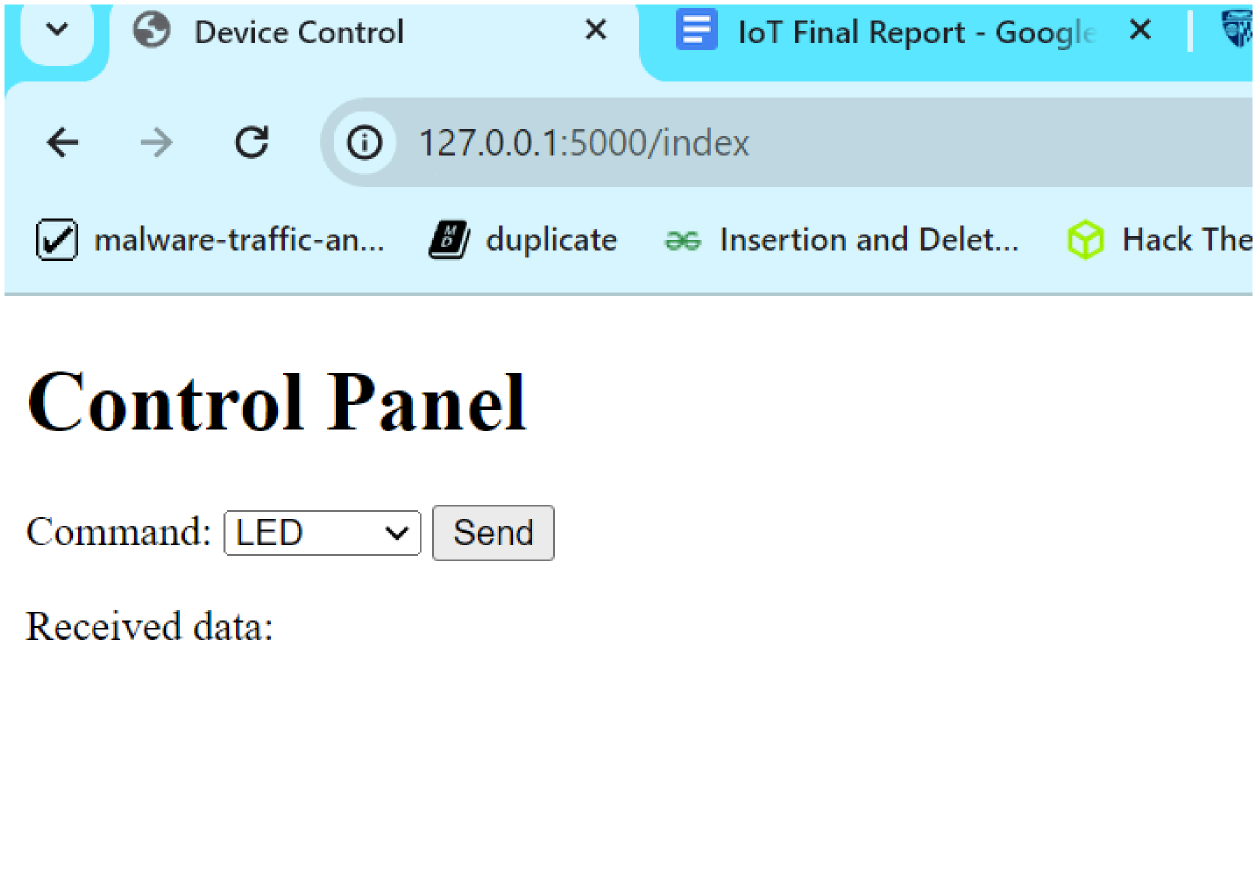


Figure 28. Directory Traversal Vulnerability, explained with 3 screenshots

However, the focus of this demonstration was for ARP spoofing, so I did not bother to make a complex login function.

The lesson learned here is that whether communication is happening between the Windows machine and minion imp, or between the controller and minion imp, both connections are able to be monitored by anyone, because both devices are vulnerable to ARP spoofing.

On design day, I plan to let people try being the middleman in any situation, but for the report, I felt explaining being a middleman between the Windows machine and a minion imp would be best for explaining ARP spoofing, due to being able to see the attack happening in real time using the 'arp -a' command and seeing duplicate entries in wireshark.

If there was no direct communication between the vulnerable devices, and they all used their agent to communicate, then this entire ARP spoofing attack would not work, because the router, or 192.168.0.1, is a secure device.

As a matter of fact, for a good month and a half, I was entering "192.168.0.1" and "192.168.0.101" as the targets for my arpSpoof.py code, as I wanted to be between the router and the controller imp, but I was constantly unsuccessful. I hypothesized that the reason must be because the electric imp devices were secure, and changed the course of the project to implementing a server on my windows machine for the electric imp devices. However, when I noticed that the connection between the windows machine and the imp device was insecure, I concluded that the reason why my previous attacks were unsuccessful must have been because the router or network security of the Barton 123 lab was solid.

That would be the end of the writeup, thanks for reading!