2IC80 - OCS Foscam Project Report

Jeremi Cichawa 2273497
jerc.b0@gmail.com
j.r.cichawa@student.tue.nl

21 June 2025

https://github.com/Nekyi4/Foscam_C1_Security_Analysis

https://youtu.be/xT1AmdON0yg

1 Introduction

The aim of this project is to find vulnerabilities that would allow a potential attacker to gain unauthorized access to the camera feed of an IP camera. The specific model used in this project is the Foscam C1 IP camera. The access will be gained by exploiting the vulnerability of the credentials being sent in plain text during the communication between the Foscam plugin, software that is required to access Foscam cameras, and user's browser. The tool will extract these credentials during this communication, and with them we will be able to gain access to the camera.

2 Possible Attack Vectors

There are two possible ways to acquire the camera feed. The first is to obtain the packets that contain the video feed, for example, by executing Man-in-The-Middle attack and sniffing on the communications between the camera and the user. The second one is to obtain access to the camera itself, either by exploiting some sort of vulnerability to gain unauthorized access to the camera without using the credentials or by in some way obtain the credentials and use them to gain access.

3 Video Feed Packet Acquisition

To check how feasible the first attack vector, so decoding the video feed from packet sniffing, I took a look at how the video feed is being send to the user. It quickly turns out to be quite challenging because Foscam does not use generic HTTP video-over-TCP protocols but rather they use their own proprietary streaming protocol.

Figure 1: Snippet of video feed communication between user and camera

After the HTTP handshake opens the ServerPush stream on port 88, the video-stream packets start to arrive. And they are in FOSC format which is from my research encrypted H.264 data. This in itself does not mean that this method is impossible. It is possible that there is a way to break this encryption and I don't have the expertise to judge how hard it would be. Despite extensive searching, I could not find any no open-source project that fully decodes the Foscam 'FOSC' stream into raw H.264 or anything else. Because of that I decided that before I attempt that I will first try the second method and come back to this if I fail there, but that never happened since I found success wit the credential sniffing method.

4 Credential Sniffing

Because the camera connection is over HTTP and there are instances where some IoT devices send the credentials in plain text over HTTP I decided to try it. It is very easy to learn the username of the user, but the password is being scrambled by the JavaScript on the website and never appears in plain text in HTTP traffic.

Figure 2: Username is in many packets but password is always scrambled

Even though the camera uses CGI for its communication I had very hard time seeing anything related to user authentication from packet traffic. Until I looked at the websocket traffic and saw that the plugin is responsible for this process. Since the browser communicates with the plugin first and then with the camera a portion of the authentication process is done purely on localhost, meaning it is not visible when sniffing

communication between user's machine and the camera. Because of that it was hard for me to fully understand how the communication occurs during the authentication process, however through my observations and testing I believe that it probably occurs like in a diagram below [Figure 3]

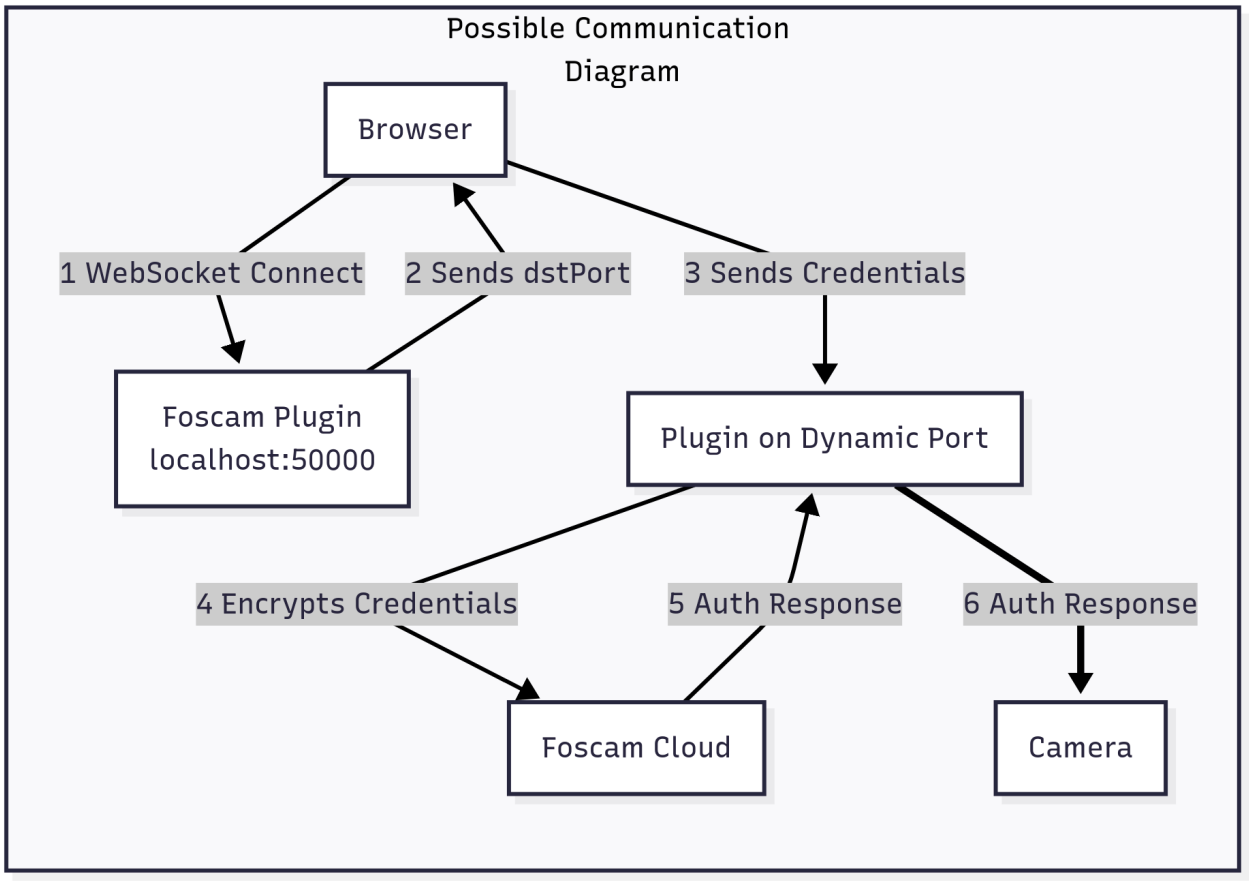

Figure 3: Possible communication diagram during authentication process

By analyzing the communication between the plugin and the browser I have discovered that in a specific packet the credentials are being transferred to the plugin from the browser in plain text.

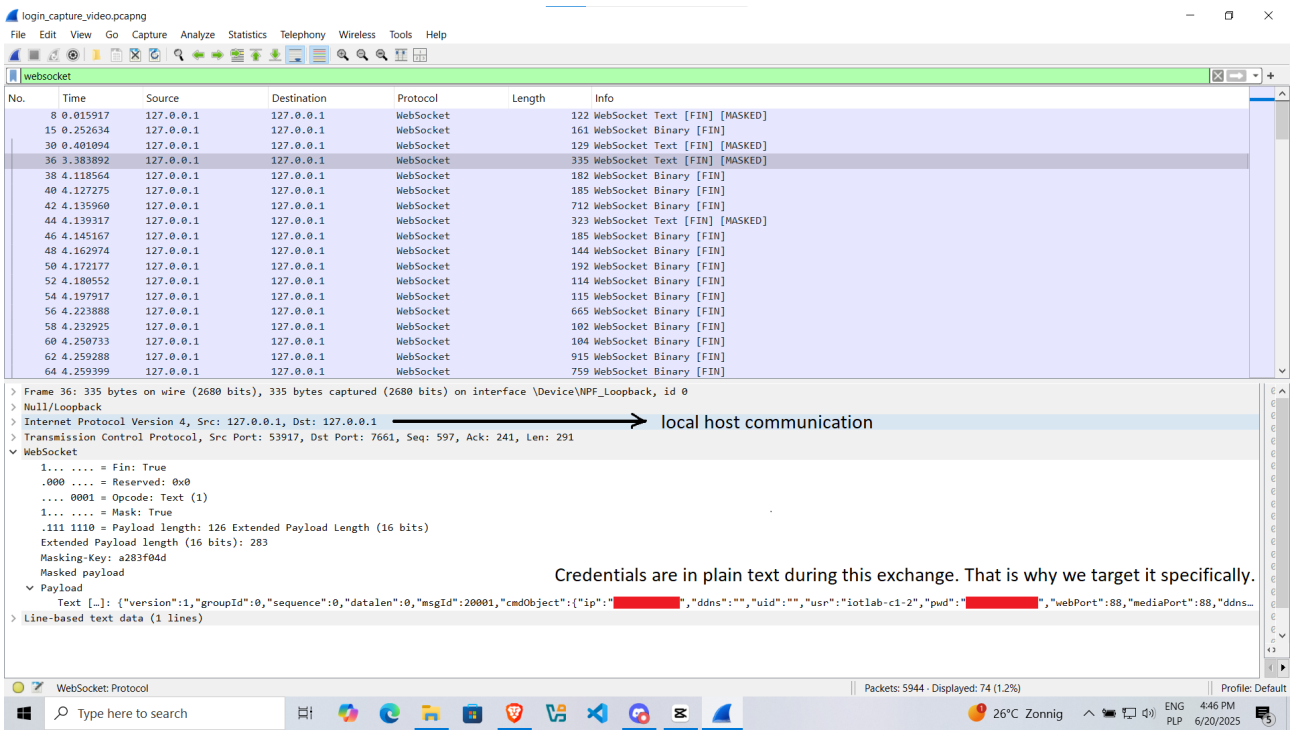

Figure 4: Packet with credentials in plain text

The attack focuses on intercepting this packet and extracting the credentials from it. To do so we need to understand how this communication occurs. It is on localhost which we know because the IP is 127.0.0.1 which is reserved for communications on local machine. This means that our tool needs to run on 'victim's' machine since this packet is never transferred over the network and we won't see it by sniffing on 'victim's' communication. By observing a few authentication attempts I also observed that the ports are assigned dynamically for each authentication attempt. This means that we need to find a way to know on which port the exchange will take place. It is important because if we can't do that we will be forced to monitor all local-host communications, on all 65535 ports. That would be incredibly inefficient and highly error-prone. However by analyzing packets using Wireshark and using Windows console we can find that the Foscam plugin listens on port 50000.

```
C:\Users\jeere>tasklist | findstr FosIPCCoreManager.exe
FosIPCCoreManager.exe         4856 Services                   0      8,924 K

C:\Users\jeere>netstat -ano | findstr 4856
  TCP    127.0.0.1:50000        0.0.0.0:0              LISTENING       4856

C:\Users\jeere>_
```

Figure 5: Foscam Plugin always listens on port 50000

It is important because this port is always used during the initial websocket handshake between the browser and the plugin. And during that handshake one of the dynamic ports that is used during credential exchange is revealed.

```
{"version":1,"msgId":20000,"groupId":248667737,"sequence":0,"datalen":0}
{
   "dstPort" : 7661,
   "groupid" : 248667737,
   "msgid" : 50000,
   "seviceVer" : "5.1.0.12",
   "ver" : "1"
}
```


Figure 6: Initial handshake between browser and plugin using port 50000 and revealing new dstPort used for further communications

The initial handshake on port 50000 follows a rigid two-step pattern:

- The client sends a text-formatted JSON request (msgId 20000) to initiate the session.
- The server responds with a binary WebSocket frame (msgId 50000) containing a JSON payload with critical parameters.

Despite the binary transport encoding (which uses WebSocket opcode 0x2), the payload contains unencrypted text data. By converting the raw hexadecimal representation to bytes and applying standard UTF-8 decoding, we recover plaintext JSON that directly exposes security-critical parameters like the dynamically assigned dstPort that is later used while passing the credentials.

5 Attack Overview

Thanks to my previous discoveries I could develop a simple proof of concept tool that can consistently intercept the credentials during the authentication process. First the tool listens on port 50000 for the initial WebSocket handshake, extracting the dynamically assigned dstPort from the server's binary response frame. The tool then switches to monitoring the discovered dstPort for packets containing msgId: 20001 since I identified that this msgId is always used for the packets with the cmdObject we are interested in. After capturing that packet it simply parses the cmdObject which is a JSON object to extract the credentials.

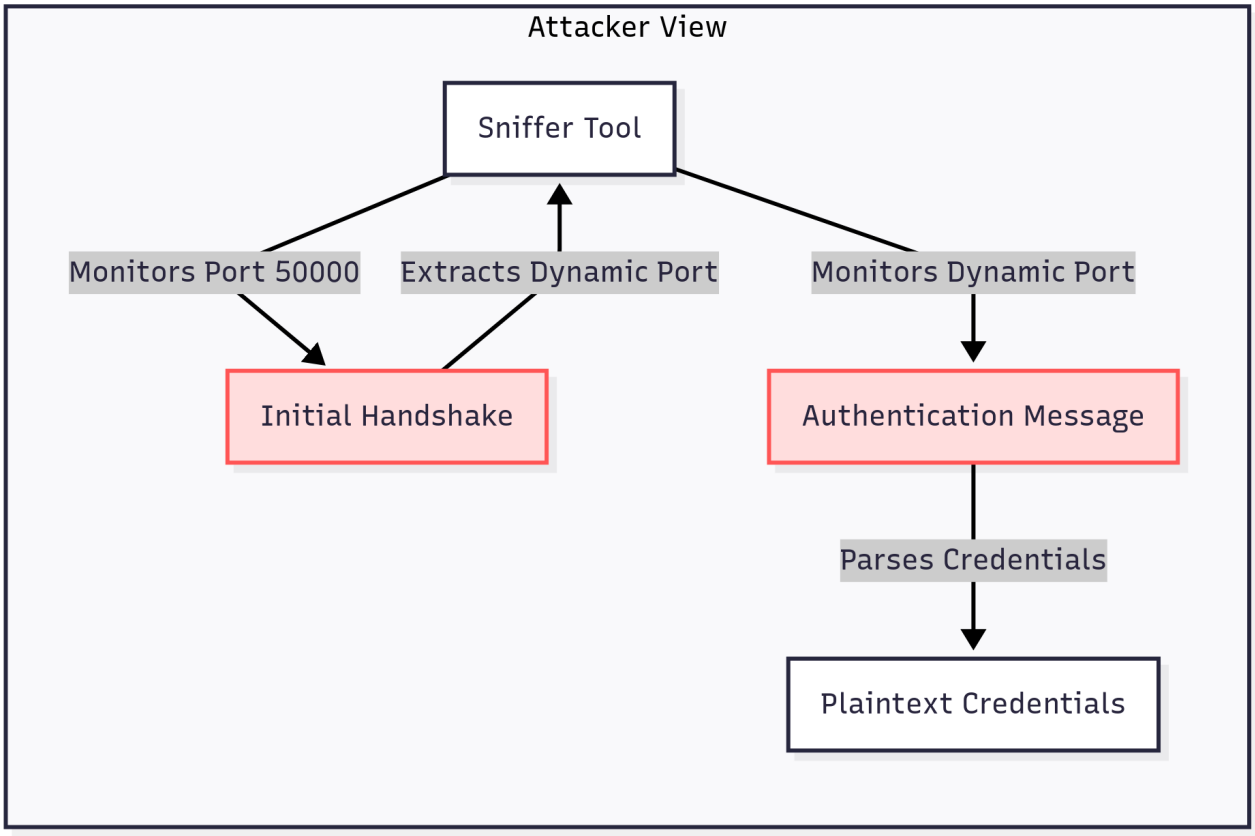

Figure 7: Diagram of how the attack/tool functions

6 Attack Setup

For the attack/tool to function the attacker/tool needs to be able to spy on localhost communications on the 'victim's' machine for the entire 'victim's' login process to the camera. How to deliver the tool to 'victim's' machine or gain such access however is out of scope for this project. That being said if I had to explore a possible attack vector for that it would be to utilize the fact that each user needs to have a plugin installed for the camera to function. Because the users know that they will not be able to access the camera without the plugin a Man-in-The-Middle attack could be orchestrated during which we would deny the user the access to the camera unless he downloads an 'update' for a plugin which would in fact be our tool or any other malicious software. As for less original approaches malicious browser extension or a USB drop may also be considered. I will however not go into detail on how to carry out such attacks since these are just suggestions and out of scope for this project.

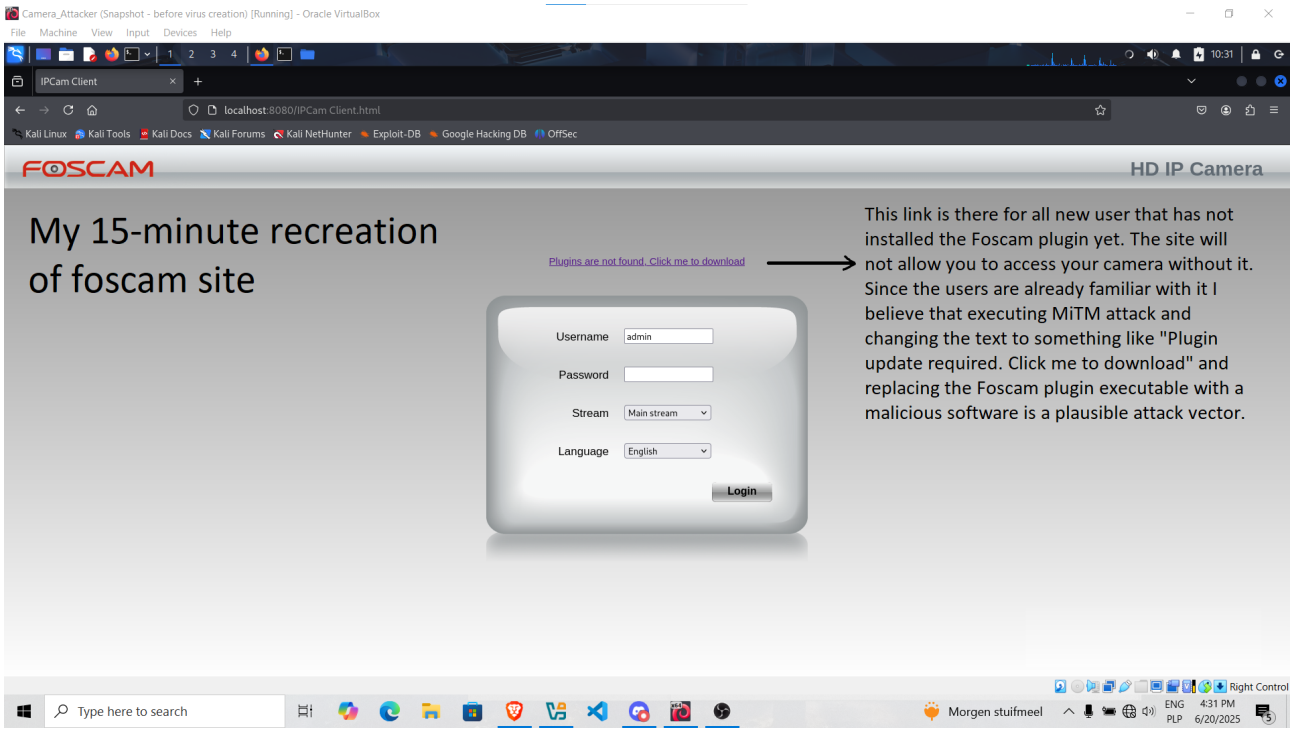

Figure 8: A possible MiTM attack showcasing a real plugin prompt that each new users sees when first entering the Foscam website.

7 Tool Capabilities

The tool is fully automatic and has two main modes:

- Offline analysis - will attempt to extract the credentials from a pcapg file.
- Live sniffing - will attempt to sniff and extract the credentials during a live authentication attempt on the current machine.

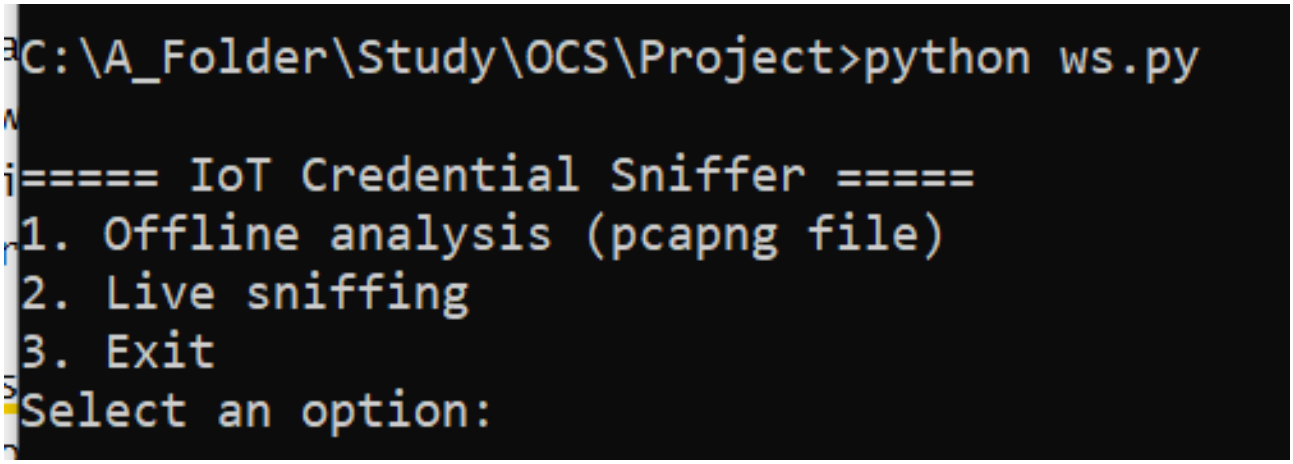

Figure 9: Main menu of the tool

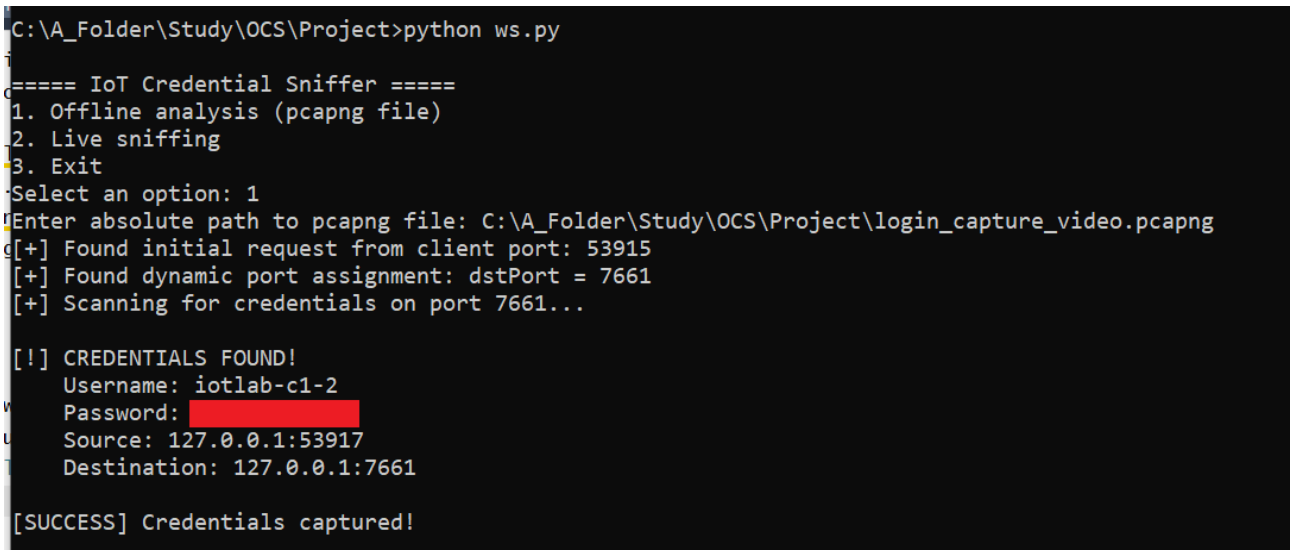

Figure 10: Possible output of the tool

8 Conclusions

The goal of this project was to gain access to the camera feed. It was achieved by gaining access to the camera itself which by extensions grants access to the video feed. This access was achieved by exploiting the fact that during a localhost websocket communications between the browser and FOSCAM plugin the credentials are being sent in plain text without any encryption. This packet is being sent between the two on dynamically assigned localhost ports. However the attack is able to reliably find the correct packet by extracting one of the dynamically assigned ports from the initial handshake between the plugin and the browser, by monitoring port 50000 which is the port the plugin is listening on initially. After that the tool switches to sniffing on the dynamically assigned port on which it intercepts the packet with plain text credentials. Because of that I believe that I have fulfilled the goal of the project.

9 Future Work

There are two potential ways in which the project could develop further:

- Delivery of the tool on potential 'victim's' machine and upgrading the tool to be able to send the credentials from the 'victim's' machine back to the attacker.
- Further exploring the packets containing the video data with the goal of trying to break the encryption and being able to reconstruct the camera feed by intercepting the packets sent between the camera and the user.

I believe that pursuing any of these potential paths would meaningfully expand the project.