Detecting illicit connections of IP cameras on the network layer

Sandino Moeniralam

October 2017

1 Abstract

This thesis demonstrates that by evaluating IP packets, unauthorized connections to an IP camera can be distinguished from authorized connections. This fundamental shift in focus on securing the application and network layers comes at a time when IoT devices are set to explode in global markets. Moreover, as our dependency on IP cameras increases, so too does the negative impact of hacked IoT devices. IoT devices are notorious for their lack of security, and certainly IP cameras are no exception. Studies reveal that the amount of data generated by IP cameras far exceeds that of any other IoT device, thus making the importance of security even more critical and challenging.
2 Acknowledgments

I would like to thank the OS3 team for introducing me into the world of cyber security. This thesis would not have been possible without the mentorship and patient guidance of my supervisor, Dr. Chaudhry. I would also like to thank my constant and unwavering supporters - my parents, for supporting me with this pursuit. Special thanks goes out to Tariq for helping me with the python script and Myra for improving the overall readability. Finally, I thank my classmates for cherished memories, inspiration and encouragement.
3 Introduction

When they first emerged, IP cameras were regarded as being relatively expensive, cumbersome and difficult to set up, while offering a limited range of capabilities. Only in the past few years have IP cameras grown to become omnipresent. In fact, vendors have specifically specialized in IP camera products where each device has a range of features. Whereas IP cameras were initially used by companies for security purposes, today their greatest market share and target audience are private individuals and end-users. In 2014, the market for IP cameras far surpassed the market for CCTV cameras[9].

The exponential growth of the IP camera has mainly been driven by expanding broadband connections, which are essential for a stable live video feed. Furthermore, smart phones allow end-users to connect to their IP camera remotely. With the range of devices on the market, it comes as no surprise that with this rate of growth, the potential vulnerabilities and threat targets of IP cameras have also drastically increased[44]. IoT devices, such as IP cameras, are generally resource strained. They have little memory and processing power available, which is mostly used up by default processes. Availability of even basic security features cannot be assumed in IP cameras. They usually do not have any security features built in, such as anti-virus software, or automatic firmware updates. It is for this reason that there has been a shift in focus of security on the application layer, to the network layer. This is done by shielding (parts) of a local area network from the internet, that may have several insecure IoT devices attached to it.

3.1 Research Questions

The purpose of this research is to discover, identify and analyze unauthorized connections to an IP camera. A working framework will be provided to demonstrate the efficacy and feasibility of interrogating the network layer. In this research, we hypothesize a scenario where an IP camera has been hacked. Using novel approaches, the network layer will be interrogated to identify illicit connections. Thus, the central research question is:

Can illicit connections to IP cameras be detected on the network layer?

Furthermore, to delve deeper into the inner workings of the different network layer protocols, additional research questions include:

- What protocols are used when connecting to IP cameras?
- Which variables can help to isolate illicit connections?
- Is automatic detection possible?
3.2 Related work

The security of IoT devices is a burgeoning area of research. Challenges and solutions related to IoT differ significantly from traditional internet connected devices, such as smart phones, laptops and desktop computers. Moreover, due to inherent resource limitations that IoT devices have, combined with the sheer range of devices on the market with restricted lifespans, a more holistic approach is essential to evaluate security risks[44].

Due to the sensitive nature of data that they capture, IP cameras are particularly more vulnerable targets. Unlike other IoT devices, IP cameras have usually been purchased with the intent of increasing physical security. Hacking these devices has the opposite result. Paradoxically, based upon the design of IP cameras, one may conclude that security is not a significant priority for these devices. Populated across the internet, numerous websites exist which maintain a public directory of unsecured IP cameras that are accessible to the general public - granted that default passwords in these cameras have not been changed[38].

One approach to secure IoT devices, such as IP cameras, is by securing the local network layer[40]. This can be achieved through the use of Software Defined Network switches[25]. However, due to the associated cost and expertise required to configure these devices, this seems to be a solution primarily for larger companies.

An emerging term is fog computing. This is somewhat akin to Software Defined Networking, although it differs as it offers many other options with regards to local data analysis and databases[17].

3.3 Relevance

The relevance of this research is substantiated by the global exponential growth of unsecured IP cameras[10]. In fact, of all the IoT devices on the market, more than half are IP cameras. The first priority towards establishing security is to develop a framework for detecting illicit connections. It is essential to discern illicit connections from authorized connections to improve the security of IP cameras, as well as for all other IoT devices. The mere fact that this research has not been previously conducted in academia or by vendors is unexpected. Little is known about why the security of IP cameras has long been overlooked. IP cameras with weak security thresholds have been used in botnets of considerable magnitude[14], which goes to demonstrate the danger - both direct and indirect, that IP cameras pose.
3.4 Scope

"As with any data network, providing that the correct security measures such as firewalls, VPNs and password protection are implemented, IP surveillance is just as secure as the millions of financial transactions made online every day. If this is compared to typical analog surveillance systems that have no encryption or authentication methods, then it is clear that bringing security into an existing IT security environment has many benefits."
- Anixer IP Video Surveillance Guide[37]

With this in mind, the scope of this research is to mitigate the security risks associated IP cameras, by resolving the ability to detect and identify unauthorized access. In this work, research will be provided regarding various forms of attacks, as well as scenarios where interrogation of the network layer will overcome the security vulnerabilities of IP cameras. Furthermore, a framework and recommendations will be provided for future research into this field.
4 Background

4.1 A short history of IP cameras

The first IP camera appeared on the market just two decades ago[1]. The Neteye 200 had limited capabilities, especially when compared to current standards. The functionality of the Neteye was to only capture frames or pictures twice daily to monitor the seas for oil spills. With its advent, the need for humans to perform physical surveillance was no longer required.

During that time, camera surveillance became ubiquitous by larger corporations which required constant monitoring. However, their cameras were all CCTV (Closed-circuit television) cameras. The drawback of this was that viewing data captured outside the local network was virtually impossible, if not impractical. IP cameras differ from CCTV cameras in the sense that they are directly connected to the internet. Consequently, data captured by IP cameras can be viewed from anywhere in the world.

Despite their limitations, these newly introduced cameras spurred the development of several successors. By introducing the low-level API that builds on the open standards of HTTP and RTSP, third party manufactures were encouraged to develop their own IP cameras[1].

The early 2000s saw a rapid increase in broadband internet, which allowed the masses with the ability to view videos online. This coincided with more IP cameras appearing on the market that offered live video recordings, instead of just sequential pictures captured every few seconds. In 2005, the first IP camera appeared with on board video content analytics (VCA), that enabled for the detection of certain events, such as the removal of objects it was filming.

The real innovation for IP cameras was the arrival of smart phones. The costs of IP cameras had plummeted dramatically, while the availability of mobile bandwidth had increased enormously. This development led to the possibility for individuals to own their own IP camera for a whole range of purposes. Vast numbers of IP cameras were produced by different vendors – but only now are we able to question the security of these devices. The threats and potential impact of security breaches have also increased exponentially as IP cameras continue to be redesigned to offer even more features and options. Unlike other IoT devices, the data captured by IP cameras is highly confidential and can cause significant damage to user-vendor relations.

With the sheer numbers of vendors actively releasing new products on an ongoing basis, the caveat remains that little attention has been paid to the security of these devices. The lack of security of IP camera is finally beginning to emerge and surface[38].

In short, IP cameras have revolutionized the concepts of physical surveillance for businesses and corporations, as well as for private individuals. At the crux
of this revolution lies business interaction with security and modern-day surveillance. Although paradoxically, these devices are expected to have the opposite effect, given their vulnerability to security breaches[1].

4.2 IoT insecurity

The Internet-of-Things, or the “proposed development of the internet in which everyday objects have network connectivity, allowing them to receive and send data”[12] has faced fierce criticism for being insecure. In fact, within the security community, the acronym IoT has been referred to as the “Insecurity of Things” and the “Internet of Threats”[31]. Since 2008, more devices are connected to the internet than people, which goes to say that “Anything that can be connected, will be connected”[24]. In the future, their numbers are expected to further increase as there seems to be no plateauing of the amount and diversity of IoT devices in the market.

The Internet-of-Things faces two significant challenges:

1. There is a vast number of IoT devices in the market, all of which are slightly different and lack standardization[12].

2. The resource restraint that most devices have prohibits them from implementing even the most basic security features, such as up-to-date anti-malware scanners, firewalls or even just limiting the amount of times a user can enter a password (which makes allowance for limitless brute-force attacks).

4.3 Top threats

The main factors related to the vulnerability of IoT devices are the heterogeneity and the large assortment of devices[44]. Despite the large assortment of devices, standardized testbeds are being developed which would include vulnerability scans, isolating and identifying insecure protocol versions, investigation of firmware updates, authentication issues and privacy violations[3]. Cipher suites are used for network connections that with TLS or SSL protocols. Albeit, IoT devices have severe resource constraints. This forces developers to implement weaker cipher suites, as well as omitting to install a robust anti-malware scanner[2]. One common feature that is often overlooked is the ongoing background communication to and from cloud servers. By monitoring idle behaviour, DNS queries made during these connections, and the query patterns revealed, sufficient metadata can be collected to uniquely identify specific IoT devices[2].

Two policies define the security of an IoT device. The data collection policy describes what data is collected, when and with which permissions[44]. The data anonymization policy ensures the anonymity of data through encryption and concealment[44].
Several solutions currently exist to secure IoT. However, most are currently under development and are rarely implemented. The most promising solution is to implement Software Defined Networking (SDN), and to implement security rules based on investigating the network layer[41]. The main advantages here are that the security of the network layer can be offered by a specialized company. This consists of adding an additional level of security, on a layer where vulnerabilities are more readily identified. A rule of thumb in security is that the higher one goes up in the OSI model, the more complicated and common security vulnerabilities are. SDN based solution builds upon the years of experience, and maturity of Intrusion Detection and Prevention Systems. Research has been conducted to implement Wireshark to investigate packets at the network layer, while even going further by combining IDPSs with SDNs[39]. This thesis further extends these findings by examining data packets in IP cameras, specifically.

Another solution that is still being evaluated is the impact of detecting a change in performance of IP cameras. IP cameras have a limited memory capacity, which causes the device to slow down considerably when other processes - such as malicious attacks, are running[23]. By mapping standard usage, any behaviour that seems to be out of the ordinary can be flagged as potential misuse or breach of the device.

In this research, all treats researched and discovered thus far have been categorized, according to the following categories: unencrypted traffic, weak credentials, weak connection, chain of trust, weak access point and firmware issues.

**Unencrypted traffic**

The use of a weak encryption algorithm (such as MD5 or SHA-1), or the complete lack of encryption algorithm allows anyone with the ability to eavesdrop on any traffic between the IP camera back-end server.

1. Idle network traffic[2]
2. Unencrypted data streams to and from the back end server[20]
3. Lack of Lightweight Cryptosystems and Security Protocols (using symmetric instead of asymmetric keys)[44]
4. Private Wi-Fi security keys are sent in clear text[20]

**Weak credentials**

Without a robust password policy, IP cameras can be connected to simply by brute forcing the passwords, or using other unsecured points of entry.

1. Allowing weak passwords[3]
2. Hard-coded credentials[19]
3. Insecure default credentials[3][19]
4. Allowing brute force attacks[3]
5. Bluetooth enabled by default allowing buffer overflow, crashing and reboot[35]
6. The configuration backup file is also protected by hard-coded credentials which, if accessed, can allow a hacker with the ability to decrypt the configuration file[7]
7. Browser vulnerabilities
8. Lack of authentication
9. Cross-Site Scripting (XSS) (OWASP Top 10)

**Weak connection**
By allowing IP cameras to connect with networks that have weak security, any security measure taken by the vendors can be voided.

1. No IDS available on IP cameras[2]
2. No anti-malware[44]
3. Cross-site request forgery (CSRF)[5]
4. Remote command injections[35][23][18][19][5]
5. Hidden and undocumented TELNET service[19][5]
6. Automatic connection to evil twins
7. HTTP response splitting[4][5]
8. Persistent cross-site scripting[19]
10. UPnP on by default[34]

**Chain of trust**
Security requires an unbroken chain of trust. Any failure in validating even a single component along this chain can generate an unsecure point of entry.

1. Weak cipher suites[3]
2. Vulnerabilities in the mobile app that connects to the IP camera[3]
3. Security of the cloud server that controls/orchestrates the device[3]
4. Security of the data itself[3]
5. Backdoors included by the manufacturer for testing purposes[2]
7. No DNSSEC employed, allowing for DNS cache poisoning and MITM attacks[2]
8. Lack of a global root certificate authority
9. Configured to used vendor’s own dynamic DNS services, making scanning the address space feasible[8][29]
10. Incorrect permissions assigned to programming scripts[19]

**Weak access point**

It is often forgotten that IP cameras are just small computers, with all the inherent security vulnerabilities of other computers. However, unlike desktop computers, IP cameras are rarely patched or even fully documented.

1. Open ports[3]
2. Accepted protocols[3]
3. Insecure admin panel
4. Unsecure web interface (OWASP IoT top 10)
5. Setting up port-forwarding rules in routers. Due to these rules, many devices are exposed to the Internet and can be attacked remotely[4]

**Firmware issues**

Any device connected to the internet should be regularly updated. Offering automatic firmware updates is less common than one would expect. Furthermore, vendors who do offer firmware updates often do not sign nor encrypt them. This can allow an attacker to add malicious snippets of code to the firmware, allowing them to gain full access to the device.

1. Unsigned or encrypted firmware[3]
2. Use of old versions of Network Time Protocol (NTP) for clock synchronization[2]
3. Lack of firmware updates, which if available, are rarely updated by users
4. Buffer overflow in the firmware[15][19][5]
5. Not encrypted or signed to stop uploads or tampering
6. Firewall leaking details about the validity of credentials[19]

**Uncategorized**

1. Data collection policy and data anonymization[44]
2. Lack of dynamic analysis (the semantic gap between real device and emulated system is an important issue to be addressed)[44]
3. Obtain a snapshot of the device’s memory. The memory dump will contain the administrator username and password in clear text, along with other sensitive information like Wi-Fi credentials or details about devices on the local network.

4. Poisoning the Web interface to remotely upload hosted piece of JavaScript code

5. Remote command injection vulnerability

6. Server-side request forgery

7. DDенial of service using requests on camera services

8. Manufacturer DDNS security issues

9. File disclosure using next_url parameter

10. Stored cross-site scripting in user name

11. Stored cross-site scripting in SSID

12. Open redirect using next_url parameter

13. Disable IP Cam and force a reset from factory

14. Sniffing the network using the WiFi driver

15. Bitcoin mining, password cracking, or any other abuse of distributed computing

16. Android app clear text authentication

17. Admin password disclosure to Eye4 API

18. Android app has malware

19. Information disclosure

20. Netwave IP remote exploit

21. Remote root access

22. Memory dump

### 4.4 Current problems and solutions

Security has always been an arms race between vendors and attackers. In the case of IP cameras, the attackers may have an upper hand. Currently available solutions place burden on the end user. For example, product manuals instruct users to change default passwords, but this is not enforced nor a requirement. Strong passwords are only recommended, but rarely enforced. The same goes for connecting to Wi-Fi where although WPA2 is advocated, most IP cameras allow wifi connections secured with weaker WEP, or no security at all. Other suggestions by vendors to enhance security are to secure one’s network and update firmware, as well as disconnecting the camera when not in use. It is easy to displace the blame of insecure environments solely to vendors. IP cameras are resource strained, have little memory, and most processing power is absorbed.
by default processes. This hinders vendors from implementing effective security measures such as anti-malware scanners and robust firewalls. One common characteristic that deters all IoT devices is that the higher one moves up the OSI model, the more vulnerabilities they encounter. Detecting vulnerabilities at the application layer is more difficult than at the network layer. The main problem with IP cameras is that they are more complex than most IoT devices. The amount of data that is sent to the back end far exceeds that of most IoT devices, which just send small packets over a regular interval. This considerable amount of data makes the detection of illicit connections even more difficult. Data volumes and network stress have even been mentioned as the top IoT concerns [28]. As a rule, the content of packets cannot be viewed on the network layer, given that these may be encrypted by a higher layer. Thus, the only way to distinguish between authorized and illicit connections are the meta data of network packets. Analyzing these for IP cameras is less trivial than for other IoT devices. Another important factor that makes the detection of a security breach even more crucial is the personal and confidential nature of data that is captured by IP cameras. Networked security cameras are the most likely to possess vulnerabilities [21]. IP cameras have also evolved to become important in court cases. As one can imagine, the validity of the photographic evidence can either make or break a legal case. With the advent of iPhone X and the facial recognition system that replaces passwords, the need to protect one’s facial photographs is even more important as photographs can be used in lieu of passwords. Lastly, consumers purchase IP cameras for ensuring security, but as discussed, threats and vulnerabilities paradoxically limit such perceived security.

4.4.1 Network layer security

Securing the network layer to shield attached physical devices is not a novel concept. IPSec was developed more than two decades ago, but is yet to be widely deployed. It has been argued that IPSec is too over-engineered and may be too complex for private individuals to implement themselves. With desktops and laptops having more secure Operating Systems such as Windows 7, 8 and 10, the need for securing the network layer diminished. However with the rise of IoT devices, the demand for a secure network layer has increased again [26]. Network-level security can be implemented across the entire range of IoT devices, rather than device-level security that is specific to any device. Be that as it may, IP packets have no inherent security. There is no inherent way to verify that the claimed sender is indeed the true sender, and that the data has not been modified or viewed by a third party as it is being transmitted. In short, there is a complete lack of nonrepudiation. Adjacent to that, the network layer has specific vulnerabilities of its own that must be overcome to secure connected devices. Vulnerabilities include Content Address Memory table exhaustion attack, ARP spoofing and DHCP starvation. IPSec counters these issues by authenticating, offering integrity and confidentiality. However, it is not the only solution to securing the network layer. Other options that are currently being evaluated are Software Defined Network Switches, advanced
firewalls, IDPS with integrated artificial intelligence and fog computing with a focus on security[30]. The first mention of protecting IP cameras by securing the network layer is from research published in 2016, whereby it was stated that “With limited storage and memory, it is seldom viable to embed anti-virus technology or advanced firewalls into the devices themselves, so the wider network needs to be robust enough to protect them from outsider threats”[42].

4.4.2 Shortcomings of current solutions

Presently, implementing anti-malware software, firewall and IDS/IPS is standard practice to secure a network. With IP cameras, each of these essential components suffers from drawbacks that overall diminish security. This research supports that anti-malware software is rarely present on the device itself. Moreover, when firewalls and IDPS are present on the network, they are still separated by a gap. An advanced security approach is required, as well as a more holistic approach that considers each element. Firewalls are insufficient, unless they are updated to detect and prevent latest vulnerabilities. On the other hand, IDPS altogether lack on private individuals. When they are present, signature-based IDS can easily detect known attacks, but fail to detect new attacks that don’t have a pattern available. Anomaly-based intrusion detection systems do offer potential, but require artificial intelligence to log and record normal behavior of IP cameras for an extended period before being able to effectively detect treats.

Even when implemented, most anti-malware scanners and firewalls can hardly stay abreast of all the data streaming. It costs too much CPU power and memory, so a trade off must be made.

Implementing encryption also has drawbacks. Encrypted packets are not processed by most intrusion detection devices. Therefore, the encrypted packet can allow an intrusion into the network that remains undiscovered, until other significant network intrusions have occurred. Therefore, encrypted packets paradoxically make it impossible for IDPS to detect and reject illicit connections. The question ultimately is how to develop a system that non-technical users can also use and understand. Merely reviewing when an IP camera was connected, how long it was connected for, and which images were seen can help non-technical users identify the hacking of their IP cameras. Home network monitoring tools such as the CUJO Smart Internet Security Firewall offer security on their network layer for non-technical users[22]. However, the effectiveness of these tools is currently under debate. An advanced firewall may offer a solution to IDS evasion techniques, but is a topic for future research.
5 Methodology

5.1 Hardware and software used

All experiments will be done using the following hardware:

- MacBook Air with macOS Sierra, Version 10.12.6
- Foscam C1
- Thunderbolt to Gigabit Ethernet Adapter
- iPad Pro with iOS 11.0.3
- Samsung Galaxy Note 3 Neo with Android 5.1.1

The software needed for this research is:

- Wireshark for macOS, Version 2.4.0rc2
- The Foscam Android application
- The Foscam iOS application
- OS X Network Utility
- Virtual Box Version 5.1.28 r117968
- Ubuntu 16.04
- hydra

5.2 Setup

The Foscam C1 will be connected to the MacBook Air via the Thunderbolt to
Gigabit Ethernet Adapter, while Wireshark will be running on the MacBook
Air, scanning the Ethernet interface. In this manner, all the pcaps taken via
Wireshark will be free from any interference from other devices.
6 Experiments

6.1 Analyzing standard behaviour

To be able to distinguish between authorized and illicit connections, a clear analysis of normal behaviour had to be made. Events that can be considered part of normal usage by the clients were captured using Wireshark. The goal here was to analyze what normal behaviour looked like, in order to be able to detect anomalies later on. The pcaps were analyzed for the following variables:

- Source and destination Mac Addresses
- Source and destination IPv4 or IPv6 addresses
- Source and destination TCP/User Datagram Protocol (UDP) ports
- Packet length
- Average packet size

By comparing normal events with unknown events, based on the above mentioned variables, differences can be determined. Just as attackers use both passive as well as active fingerprinting to retrieve as much information as possible about their targets, so can system administrators use fingerprinting to detect and repel attacks. First off, a small database of known events had to be logged.
and analyzed. This research focuses on visualizing these differences, in order to understand that there are multiple ways in which data can be analyzed, categorized and visualized.

Normal behaviour was categorized as following:

**Connecting to the IP camera, via the Android app, using 3g**
- Failed to connect
- Live video without sound
- Live video with bi-directional sound
- Recording video and pictures, and storing these in the cloud
- Using a 3g connection, a delay of 5 to 10 seconds was noticed

**Connecting to the IP camera, via the Android app, using the same WiFi that was shared with the ethernet cable connected to the IP camera**
- Live video without sound
- Live video with bi-directional sound
- Login and logout of the Foscam app
- Deleting and adding a camera to the Foscam app
- Requesting information about the camera
- Changing the password and username of the camera
- Switching between day and night filters
- Idle traffic, with the android app still logged in, but not sending live video
- Idle traffic, with the android app logged out
- Time sync
- Turning the IP camera on and off
- Turning the WiFi on and off

**Connecting the IP camera, via the iPad app, using the same WiFi that was shared with the Ethernet cable connected to the IP camera**
- Failed to connect
- Recording video and pictures, and storing these in the cloud
- A connection with bi-directional sound
Alarm notifications

- Activity (movement) detected
- Sound detected

After having made an overview of normal, authorized behaviour, the 6 different categories of illicit connections were tested, after which the results were compared to the authorized connections. To summarize; of every event, a packet capture was taken using Wireshark. This packet capture was then analyzed for the following variables:

- Source and destination IP
- Source and destination Port
- Source and destination MAC address
- Protocols
- TTL
- IP Flags
- TCP Flags

After which two graphs were made to visualize the differences, and give a complete overview of a certain event. By having a different viewpoint, subtle differences between network traffic may appear, that otherwise would not be immediately visible. The first graph is to categorize the packets depending on their size in bytes. The following categories are distinguished:

1. 0 - 100 bytes
2. 100 - 500 bytes
3. 500 - 1,000 bytes
4. 1,000 - 1,500 bytes
5. 1,500+ bytes

On the left side of the graph, the total amount of packets is plotted and the corresponding percentages on the right side of the graph. A red line will be drawn as an example signature that could be stored for later comparison.

The second graph gives an overview of the packet sizes over time. On the left side of the graph, the actual packet size is plotted. At the bottom, the packet number is plotted. The red line is the average packet length of a certain event. Every event in total will have four images:

1. A screenshot of the Wireshark packet capture
2. A table named ‘details’ with the variables of that specific packet capture

3. A graph with the packets categorized by length

4. A graph showing the packet length over time

To keep the observation as brief as possible, only clearly visible differences will be put in the experiments section. Other interesting images will be put in the appendix. Due to the scope of this research, and an enormous amount of different events analyzed, only a select few are shown in the appendix. Further data can be requested from the author.

6.2 Unencrypted traffic

Unencrypted traffic can lead to eavesdroppers reading credentials in plain text. For this research, two types of idle traffic were analyzed. Idle traffic with the Android app logged in and connected, and idle traffic with the Android app logged out. During this research, no unencrypted traffic was found. Nor the ability to change the encryption algorithm, or the turn it on and off, as is the case with certain other types of IP camera.

Figure 3: Idle traffic with the Android phone on and the app logged in. Almost only small UDP packets were send across
Figure 4: Idle traffic with the Android phone on and app logged in. As seen above, the majority of packets have a length below 100 bytes.

Figure 5: Idle traffic with the Android phone on and app logged in. A repeating pattern can be recognized.
Figure 6: Idle traffic with the Android app logged out. The total amount of traffic drastically decreased.

![Length-Packet bar graph](image)

Figure 7: Idle traffic with the Android app logged out. The few packets that were sent across were of larger size than was the case with idle traffic with the Android app logged in.

21
Figure 8: Idle traffic with the Android app logged out. Until the app is logged in, the patterns remains the same, with almost no packets flowing across.

A clear distinction can be seen here between the two. The amount of traffic drops to almost zero, when the Android app is logged out.

6.3 Weak credentials

Most IP cameras do not limit the amount of password tries a user can try, here by allowing brute force attacks. In this experiment, two brute force attacks were performed. One with a textfile without the correct password, and one with. The packet size graph was exactly the same for either. But when the correct password was tried, a clear spike in packet size is visible in the last graph. The program used here was hydra, running on a virtualized Ubuntu environment, that did not recognize the response, and kept on trying other password combinations.
Figure 9: An unsuccessful brute force attack

Figure 10: Both brute force attacks produced the same graph
Figure 11: An unsuccessful brute force attack, visible as a repeating pattern.

Figure 12: A successful brute force attack. The spike coincides with the correct password being tried by hydra. Hydra, however does not recognize the response, and keeps on trying other passwords.
As can be seen here, the researched IP camera does indeed allow brute force attacks, and are clearly detectable on the network layer.

6.4 Weak connection

One vulnerability that can be found in several IP cameras is a hidden and undocumented TELNET connection.

Figure 13: Using the OS X’s Network Utility, TELNET, SSH, SFTP and FTP were tried out. Of these, TELNET and FTP were successful
Figure 14: Trying out TELNET, SSH, SFTP and FTP

Figure 15: 4 recognizable drops in packet size, correlating to the TELNET, SSH, SFTP and FTP requests
As can be seen above, the Foscam C1 IP camera obviously allows the TELNET connection, despite not being documented in the manual.

6.5 Chain of trust

The Foscam C1 IP camera has been researched before, revealing several weaknesses. One weakness that has been dealt with in the latest firmware update is connecting to the camera via FTP, using port 50021[5].

![Figure 16: After connecting using FTP, files could be uploaded to the camera as well as removed](image-url)
Figure 17: The tell tale signs of a FTP connection

Figure 18: Using the terminal to connect to the IP camera using FTP
Figure 19: Besides uploading and downloading files, many other FTP commands were available.

6.6 Weak access point

Full port scans are a common way to detect open ports, after which different protocols can be used to establish a connection. When a port scan is detected, an alarm should be set, and refusing any other requests of the IP in question.
**Figure 20:** A port scan via the terminal is clearly visible here

**Figure 21:** Typical for port scans is the small packet size, as can be seen here, almost all of the packets are under 100 bytes
Figure 22: Only a couple of the almost 130,000 packets were larger than 100 bytes

6.7 Firmware issues

Firmware updates are necessary for updating the options of the device, as well as for security purposes. After the firmware upgrade, no notification was sent to the user on the Android app. This means that a potential hacker can upload new firmware, without any clear indication that this has taken place.
Figure 23: During the firmware upgrade, the packets were unusually large.

Figure 24: The firmware update was the only event where packets appeared larger than 1500 bytes.
Figure 25: The true overhead size of the firmware update

Figure 26: Current firmware installed

Figure 27: A firmware upgrade is available

Figure 28: Starting the firmware upgrade
Figure 29: No firmware upgrade available

Figure 30: Up to date firmware
7 Proposed solutions

At the moment several solutions are being researched. To sum up the results of the experiments:

What protocols are used when connecting to IP cameras?
The most used protocols are UDP, TCP, FTP, ICMP, DHCP, ARP.

Which variables can help to isolate illicit connections?
Of all the variables researched, the following seem sufficient enough in distinguishing different types of connections:

- Source and destination MAC address
- Source and destination IPv4 or IPv6 addresses
- Type of service
- Round trip time
- Packet length

Is automatic detection possible?
Automation should be possible with more advanced machine learning algorithms than are currently implemented in IDPS's. Here is a short comparison, taking into account the above-mentioned considerations.

Software Defined Network switch
As discussed, SDN switches allow the detection of malicious packets, and the refusal of requests for certain IP addresses. However, these devices are usually complex in usage, and not meant for private individuals[43].

Advanced Firewalls for IoT devices
Firewalls are rarely installed or updated on IoT devices. Physical devices, through which all IoT data flows are starting to appear on the market[22]. These devices can log when a connection is made to which device, but give no details about the type of connection made[33]. Further research will have to show how effective these advanced firewalls actually are, and whether they offer a solution to the broad array of challenges researchers face in terms of general system security, network security, and application security[16].

Cloud cameras
IP cameras have gained popularity over CCTV cameras, mainly because of their flexibility, and low cost. They are not, however, a complete substitution for CCTV cameras. Cloud cameras come closer to this analogy. Cloud cameras differ from IP cameras in the sense that they offer seamless integration with their back end. They are designed from the start with simplicity and security in mind[11]. Because the back end is part of the product, vendors are directly responsible for the security of the data stored, as well as securing the connections between the cloud and the cameras. This makes it easier for them to secure,
find weaknesses and update the system as a whole, hereby avoiding the dreaded single-point-of-failure. The costs may be significantly larger than for IP cameras, but the added benefits make this a good alternative for mid-size companies.

**Not connecting the device to the internet, but only locally**

Not connecting one’s device on the internet, but only on a local network is a viable alternative, without added costs. However, the flexibility IP cameras offer, mainly that they can be reached from anywhere in the world, is undone in this setup.

**Fog Computing**

Fog computing is a relatively new concept, where most computation is done on the data locally before it is sent out into the cloud. It extends cloud computing and could prove vital for maintaining applications that require short response times[17]. Fog computing promises to do more than just securing IoT devices. Coupled with Artificial Intelligence, it can make the internet of things actually smarter, and more responsive to the direct environment[13].

**802.1X**

One solution that is available at this moment, but has not yet been researched for usage of IoT devices, is the IEEE 802.1X standard for port-based Network Access Control. It helps to authenticate and secure both wireless and wired LANs. 802.1X passes the Extensible Authentication Protocol (EAP), which includes the user credentials, over the wire by packaging them in Ethernet frames instead of using Point-to-Point Protocol (PPP). PPP offers a lot of functionality but at the same time a lot of overhead that is not needed with IoT devices. 802.1X uses a RADIUS server as an authentication server. This server authenticates the supplicant, which is attached to an authenticator. The main advantage of 802.1X is that the authenticator, in this case, the IP camera, can be very simple in design. Little memory and processing power is needed on the authenticators side[32].

**8 Discussion**

This research shows that it is possible to detect illicit connections to IP cameras by looking at the network layer. The network protocol analyzer Wireshark shows many different protocols being used when connecting to IP cameras. It depends on the type of events that are taking place. Firmware upgrade happen for instance over TCP, while password changes happen over UDP. Besides the protocols, there are a lot of variables that can help to isolate illicit connections. (average) packet lengths, the amount of packets, source and destination IP are just a few of many variables that could potentially be used. With limited time and resources, only a couple variables were actively used in visualizing the packet captures. And still, almost trivial, differences were clearly visible. One can only imagine what previously undetected patterns will be able to reveal if
analyzed correctly. There are countless ways in which the data can be plotted and compared. There seems to be a gap in the security currently offered by Intrusion Detection and Prevention Systems. Despite their sophistication, intrusion detection system evasion techniques are still effective. Firewalls are rarely installed on IoT devices themselves, and IDPS not at all. Several solutions exist, none of which is currently fully functional. With increasing amounts of data to monitor, analyze and filter, and security threats being an every day threat, traditional security products fall behind. Being able to analyze network data over longer periods of time, and distinguish every event, will prove valuable. Automatic detection should be possible, but would require some form of artificial intelligence that is more advanced than what current firewalls and IDPS offer. Using machine learning algorithms is no new concept. It is a field of security that has seen increased attention in the last years[27]. The main drawback of this research is that normal behaviour has to be mapped meticulously, and understood well by a potentially usable and useful machine learning algorithm. Another shortcoming is that each graph results in a subjective interpretation. For practical usage these interpretations have to be expressed in numbers that programs can use to compare.

At present, of the several solutions discussed, two seem to be practical and offer at least partial protection. Cloud cameras seem to offer the greatest security, but unfortunately, come with a price tag that does not make them a viable option for most private individuals. The only viable security solution seems to be 802.1X, which could be offered as a service to anyone who owns an IP camera. The drawback here is that authentication will be done externally, coming with its own risks. The third party responsible for this service must be trustworthy in order for the end setup to be safe. Ultimately, security comes down to trust. And being able to verify this trust.

9 Conclusion

Detecting illicit connections to IP cameras can be done by looking at IP packets. In this research, analysis was done by humans but could be automated in some way. Advanced firewalls such as the CUJO, do offer similar security, but in a very limited way. Only whether a connection has been made to the device, not what type of connection. Analysis of metadata shows more on the past usage, and the types of connections made (video recording, only sound, firmware upgrade, password change) than logs made available to users by the vendors. The protocols used depend on the type of connection, usually either UDP or TCP. And the variable that this research mainly used to visualize differences was packet length. Automatic detection should be possible, but require and enormous amount of test data.
10 Future work

Advanced firewalls such as the CUJO may offer a solution to the total lack of security in most IP cameras, but do not offer protection from IDS evasion techniques. Future research should be done to analysis till what degree they offer security. Fog computing is a relatively new concept, and will surely be used for the security of IoT devices. Their complexity may make them hard to implement for individuals, but future work can be done to evaluate their effectiveness. One interesting aspect that promises to change the world of IT is artificial intelligence. Using machine learning to analyze the immense amount of different variables, can reveal patterns that have previously been undetected. Lastly, the implementation of 802.1X can be considered, and whether this indeed would offer adequate protection for all IP cameras. In previously done research into network layer security, traffic to and from different IoT devices has been analyzed. But none of IP camera traffic specifically, despite or perhaps because of the enormous amount of data IP cameras produce. By using a python script as a simple data mining tool, previously unseen differences became visible. Having shown basic differences, this research can serve as a stepping stone to advance the science of fingerprinting specific network behaviour.
11 Appendix

11.1 Pcaps and retrieved data of authorized traffic

Figure 31: The activity detection alarm going off

Figure 32: The activity detection alarm going off

39
Figure 33: Login and logout on the Android smartphone via wifi

Figure 34: Login and logout on the Android smartphone via wifi
Figure 35: Login and logout on the Android smartphone via wifi

Figure 36: As can be seen here, when the password and camera name are changed on the Android Foscam app, this is done over UDP
Figure 37: Change password and camera name on the Android app

Figure 38: The FTP connection also allows the removal of files and folders
Figure 39: Deleting a file via ftp

Figure 40: Deleting the camera from the Android app via wifi
Figure 41: Deleting the camera from the Android app via wifi

Figure 42: The sound activity alarm going off
Figure 43: The sound activity alarm going off

Figure 44: The sound activity alarm going off

45
Figure 45: A time sync of the IP camera

![Length Packet graph with average and actual length]

Figure 46: A time sync of the IP camera
Figure 47: Turning the camera on and off

Figure 48: Turning the camera on and off
Figure 49: Turning the wifi on and off

Figure 50: Turning the wifi on and off
11.2 Pcaps of retrieved data of illicit traffic

Figure 51: A lookup via Network Utility

Figure 52: A lookup via Network Utility
Figure 53: A performed traceroute

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Length</th>
<th>Source port</th>
<th>Destination port</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2 1:43:16</td>
<td>255.255.255.255</td>
<td>10.10.10.10</td>
<td>TCP</td>
<td>0</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>2</td>
<td>2 1:43:16</td>
<td>255.255.255.255</td>
<td>10.10.10.10</td>
<td>PPP</td>
<td>0</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>3</td>
<td>2 1:43:16</td>
<td>255.255.255.255</td>
<td>10.10.10.10</td>
<td>PPP</td>
<td>0</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>4</td>
<td>2 1:43:16</td>
<td>255.255.255.255</td>
<td>10.10.10.10</td>
<td>PPP</td>
<td>0</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>5</td>
<td>2 1:43:16</td>
<td>255.255.255.255</td>
<td>10.10.10.10</td>
<td>PPP</td>
<td>0</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>6</td>
<td>2 1:43:16</td>
<td>255.255.255.255</td>
<td>10.10.10.10</td>
<td>PPP</td>
<td>0</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>7</td>
<td>2 1:43:16</td>
<td>255.255.255.255</td>
<td>10.10.10.10</td>
<td>PPP</td>
<td>0</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>8</td>
<td>2 1:43:16</td>
<td>255.255.255.255</td>
<td>10.10.10.10</td>
<td>PPP</td>
<td>0</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
</tbody>
</table>

Figure 54: A performed traceroute

11.3 Python script for data retrieval

#!/usr/bin/python

import pyshark
import sys, os.path
import matplotlib.pyplot as plt
import matplotlib
import six
import numpy as np
from matplotlib.ticker import FuncFormatter

#global variable
win=1

def analyze_pcap(pcap):
    print()

    # returns ip or ipv6
    def find_src_ip(pkt):
        if hasattr(pkt, 'ip'):
            #print(pkt.ip.src)
            return pkt.ip.src
        elif hasattr(pkt, 'ipv6'):
            #print(pkt.ipv6.src)
            return pkt.ipv6.src
        else:
            #print("no ip")
            return None

    def find_dst_ip(pkt):
        if hasattr(pkt, 'ip'):
            #print(pkt.ip.dst)
            return pkt.ip.dst
        elif hasattr(pkt, 'ipv6'):
            #print(pkt.ipv6.dst)
            return pkt.ipv6.dst
        else:
            #print("no ip")
            return None

    def find_src_mac(pkt):
        if hasattr(pkt, 'eth'):
            #print(pkt.eth.src)
            return pkt.eth.src
        else:
#print("no mac")
return None

def find_dst_mac(pkt):
    if hasattr(pkt, 'eth'):
        #print(pkt.eth.dst)
        return pkt.eth.dst
    else:
        #print("no mac")
        return None

def find_transport_protocol(pkt):
    if hasattr(pkt, 'transport_layer'):
        protocol = pkt.transport_layer
        return protocol
    return None

def find_transport_src_port(pkt):
    protocol = find_transport_protocol(pkt)
    if protocol != None:
        return pkt[pkt.transport_layer].srcport
    else:
        return None

def find_transport_dst_port(pkt):
    protocol = find_transport_protocol(pkt)
    if protocol != None:
        return pkt[pkt.transport_layer].dstport
    else:
        return None

def find_tcp_port(pkt):
    if hasattr(pkt, 'tcp'):
        print(pkt.tcp.port)
        return pkt.tcp.port
    else:
        print("not tcp")
        return "not tcp"

def find_udp_port(pkt):
    if hasattr(pkt, 'udp'):
        print(pkt.udp.port)
        return pkt.udp.port
    else:
        print("not udp")
        return "not udp"
def find_udp_src_port(pkt):
    if hasattr(pkt, 'udp'):
        print(pkt.udp.srcport)
        return pkt.udp.srcport
    else:
        print("not udp")
        return "not udp"

def find_udp_dst_port(pkt):
    if hasattr(pkt, 'udp'):
        print(pkt.udp.dstport)
        return pkt.udp.dstport
    else:
        print("not udp")
        return "not udp"

def find_DSCP(pkt):
    if hasattr(pkt, 'tcp'):
        if hasattr(pkt.ip, 'dsfield_dscp'):
            print(pkt.ip.dsfield_dscp)
            return pkt.ip.dsfield_dscp
        else:
            print("not dsfield_dscp")
            return "not dsfield_dscp"
    else:
        print("not ip")
        return "not ip"

def find_ttl(pkt):
    if hasattr(pkt, 'ip'):
        print(pkt.ip.ttl)
        return pkt.ip.ttl
    else:
        print("not ip")
        return "not ip"

def exit():
    sys.exit()

def detect_ftp_attack(pkt):
    global ftp_pkt_list
    try:
        if pkt.tcp.srcport == "50021" or pkt.tcp.dstport == "50021":
            if pkt.ip.src == target_ip or pkt.ip.dst == target_ip:
                ftp_pkt_list.append(pkt)
except:
    pass

def print_conversation_header(pkt):
    try:
        protocol = pkt.transport_layer
        src_addr = pkt.ip.src
        src_port = pkt[pkt.transport_layer].srcport
        dst_addr = pkt.ip.dst
        dst_port = pkt[pkt.transport_layer].dstport
        Data = pkt.data.data.decode('hex')
        #Data=pkt.data.data.decode('hex')
        print ('%s %s:%s --> %s:%s' % (protocol, src_addr, src_port, dst_addr, dst_port))
        print (Data)
    except AttributeError as e:
        #ignore packets that aren't TCP/UDP or IPv4
        pass

def length_graph(pkt_list):
    global win
    length=[]
    pkt=[]
    avg_len=[]
    i=1
    total_len=0
    try:
        for p in pkt_list:
            l = int(p.length)
            total_len= l + total_len
            length.append(l)
            pkt.append(i)
            i=i+1
        except AttributeError as e:
            pass
    avg = total_len / (i - 1)
    for j in range(1, i):
        avg_len.append(avg)
    g=plt.figure(win)
    win = win + 1
    # plotting the points
    plt.plot(pkt, avg_len, label='average length', color='r')
    #print (length)
# print (pkt)
plt.plot(pkt, length, label='actual length', marker='.

# naming the x axis
plt.xlabel('Packets')
# naming the y axis
plt.ylabel('Length (bytes)')

plt.title('Length-Packet graph')
# show a legend on the plot
plt.legend(loc='upper left', bbox_to_anchor=(0.6, 1.05),
           ncol=3, fancybox=True, shadow=True)
g.show()
#raw_input()

def val_to_percent(y, position):
    # Ignore the passed in position. This has the effect of scaling the default
    # tick locations.
    s = str(y)
    return s + '%'

def length_hist(len_list):
    global win
    fig, ax1 = plt.subplots()
    n_groups = 5
    index = np.arange(n_groups)
    bar_width = 0.35

    opacity = 0.4
    error_config = {'ecolor': '0.3'}

    len_range=('0-100', '100-500', '500-1000', '1000-1500', '1500<')
    rects1 = ax1.bar(index + bar_width, len_list, bar_width,
                     alpha=opacity,
                     color='b',
                     error_kw=error_config
                     )

    ax1.set_xlabel('No. of Packets')
    ax1.set_ylabel('Length (bytes)')
    plt.title('Length-Packet bar graph')
    plt.xticks(index + bar_width + bar_width/2, len_range)
    per=[]

    total=float(sum(len_list))
# print (total)
for i in len_list:
    per.append(percentage(float(i), total))

ax2 = ax1.twinx()
ax2.plot(index + bar_width + bar_width/2, per, '*r-')
formatter = FuncFormatter(val_to_percent)
ax2.set_ylabel('Percentage', color='r')
ax2.tick_params('y', colors='r')
plt.gca().yaxis.set_major_formatter(formatter)
fig.tight_layout()
plt.show()

def summary_table(list1, list2, list3, list4, list5, list6, list7, list8, list9, list10):
    global win
t = plt.figure(win)
win = win + 1
rows = ['Source IP', 'Destination IP', 'Src Port', 'Dst Port', 'Protocol', 'Src MAC', 'Dst MAC', 'Details']

    cell_text = [[list_as_string(list1)], [list_as_string(list2)], [list_as_string(list3)],
                [list_as_string(list4)], [list_as_string(list5)], [list_as_string(list6)], [list_as_string(list7)],
                [list_as_string(list8)], [list_as_string(list9)], [list_as_string(list10)]

    the_table = plt.table(cellText=cell_text,
                          rowLabels=rows,
                          colLabels=cols,
                          cellLoc='left',
                          colLoc='center',
                          loc='upper center', bbox=(-0.05, 0, 1.14, 1))

    # plt.subplots_adjust(left=0.2, bottom=0.3)
    for k, cell in six.iteritems(the_table._cells):
        # cell.set_edgecolor(edge_color)
        # if k[0] == 0 or k[1] < header_columns:
            cell.set_text_props(color='b', wrap=True)
            cell.set_facecolor(header_color)
        # the_table._cells.set_text_props(weight='bold', color='w', wrap=True)

        cells = [key for key in the_table._cells]
    for cell in cells:
        # print key[1]
        # print the_table._cells[cell].PAD
        the_table._cells[cell].PAD = 0.05

    the_table.scale(1.3, 4)
```python
def list_as_string(list):
    str = ', '.join(list)
    return str

def append_unique(list, value):
    if list.count(value) == 0:
        list.append(value)
    return list

def percentage(n, total):
    return (n * 100) / total

# Main script
if(len(sys.argv) < 2) :
    print ('Usage : python intruder.py pcap-filename camera-ip')
    file = raw_input('Enter a pcap filename: ')  
    if file == '
':  
        exit()
    target_ip = raw_input('Enter the camera IP: ')  
    if target_ip == '
':  
        exit()
else:
    file = sys.argv[1]
    target_ip = sys.argv[2]

if not os.path.isfile(file):
    print (file,"Error: file not found")
    exit()

#target_ip = raw_input('Enter the camera IP: ')
packet_list=[]
src_ip_list=[]
dst_ip_list=[]
src_port_list=[]
dst_port_list=[]
protocol_list=[]
```

57
try:
cap = pyshark.FileCapture(file, keep_packets=False)
print ("Analyzing packets...")

#cap.apply_on_packets(detect_ftp_attack)
if len(ftp_pkt_list) > 0:
    print ("FTP attack detected")
    for pkt in ftp_pkt_list:
        print_conversation_header(pkt)

#cap.apply_on_packets(print_conversation_header)

for pkt in cap:
    #neglect eth layer packet
    if not hasattr(pkt, 'ip'):
        continue

    src_ip = find_src_ip(pkt)
    dst_ip = find_dst_ip(pkt)
    if src_ip == target_ip or dst_ip == target_ip:
        packet_list.append(pkt)
        ttl_list = append_unique(ttl_list, pkt.ip.ttl)
        src_ip_list = append_unique(src_ip_list, src_ip)
        dst_ip_list = append_unique(dst_ip_list, dst_ip)
        src_mac = find_src_mac(pkt)
        if src_mac != None:
            src_mac_list = append_unique(src_mac_list, src_mac)
        dst_mac = find_dst_mac(pkt)
        if dst_mac != None:
            dst_mac_list = append_unique(dst_mac_list, dst_mac)

        protocol = find_transport_protocol(pkt)
        if protocol != None:
            protocol_list = append_unique(protocol_list, protocol )
            src_port = find_transport_src_port(pkt)
            src_port_list = append_unique(src_port_list, src_port)
            dst_port = find_transport_dst_port(pkt)
            dst_port_list = append_unique(dst_port_list, dst_port)

    else:
        protocol_list = append_unique(protocol_list, pkt.highest_layer)
#find IP flags
if pkt.ip.flags_rb == '1':
    ip_flags = append_unique(ip_flags, 'Reserved bit')
if pkt.ip.flags_mf == '1':
    ip_flags = append_unique(ip_flags, 'More fragments')
if pkt.ip.flags_df == '1':
    ip_flags = append_unique(ip_flags, 'Don’t fragment')

#find tcp flags
if hasattr(pkt, 'tcp'):
    if pkt.tcp.flags_fin == '1':
        tcp_flags = append_unique(tcp_flags, 'FIN')
    if pkt.tcp.flags_syn == '1':
        tcp_flags = append_unique(tcp_flags, 'SYN')
    if pkt.tcp.flags_reset == '1':
        tcp_flags = append_unique(tcp_flags, 'RESET')
    if pkt.tcp.flags_push == '1':
        tcp_flags = append_unique(tcp_flags, 'PSH')
    if pkt.tcp.flags_ack == '1':
        tcp_flags = append_unique(tcp_flags, 'ACK')
    if pkt.tcp.flags_urg == '1':
        tcp_flags = append_unique(tcp_flags, 'URG')
    if pkt.tcp.flags_ecn == '1':
        tcp_flags = append_unique(tcp_flags, 'ECN')
    if pkt.tcp.flags_cwr == '1':
        tcp_flags = append_unique(tcp_flags, 'CWR')
    if pkt.tcp.flags_ns == '1':
        tcp_flags = append_unique(tcp_flags, 'NS')
    if pkt.tcp.flags_res != '0':
        tcp_flags = append_unique(tcp_flags, 'RES')

l = int(pkt.length)
if l <= 100:
    length[0] += 1
elif l <= 500:
    length[1] += 1
elif l <= 1000:
    length[2] += 1
elif l <= 1500:
    length[3] += 1
elif l > 1500:
    length[4] += 1

#'' if pkt.ip.src == target_ip or pkt.ip.dst == target_ip:
protocol = pkt.transport_layer
print(protocol)
src_addr = pkt.ip.src
src_port = pkt[pkt.transport_layer].srcport
dst_addr = pkt.ip.dst
dst_port = pkt[pkt.transport_layer].dstport
if pkt.ip.src == target_ip or pkt.ip.dst == target_ip:
    if src_mac_list.count(pkt.eth.src) == 0:
        src_mac_list.append(pkt.eth.src)
    if dst_mac_list.count(pkt.eth.dst) == 0:
        dst_mac_list.append(pkt.eth.dst)
    if protocol_list.count(protocol) == 0:
        protocol_list.append(protocol)
    if dst_port_list.count(dst_port) == 0:
        dst_port_list.append(dst_port)
    if src_port_list.count(src_port) == 0:
        src_port_list.append(src_port)
    packet_list.append(pkt)
if dst_ip_list.count(pkt.ip.dst) == 0:
    dst_ip_list.append(pkt.ip.dst)
if src_ip_list.count(pkt.ip.src) == 0:
    src_ip_list.append(pkt.ip.src)'''

'''
    elif pkt.ip.dst == target_ip:
        if src_mac_list.count(pkt.eth.src) == 0:
            src_mac_list.append(pkt.eth.src)
        if dst_mac_list.count(pkt.eth.dst) == 0:
            dst_mac_list.append(pkt.eth.dst)
        if protocol_list.count(protocol) == 0:
            protocol_list.append(protocol)
        if dst_port_list.count(dst_port) == 0:
            dst_port_list.append(dst_port)
        if src_port_list.count(src_port) == 0:
            src_port_list.append(src_port)
        packet_list.append(pkt)
    if src_ip_list.count(pkt.ip.src) == 0:
        src_ip_list.append(pkt.ip.src)'''

input=len(packet_list)
if input == 0:
    print ("no packet detected with Src or Dst IP as target_ip:"),target_ip
    exit()
print('Total packets detected with either Src or Dst IP as target_ip:',input)
#summary_table(src_ip_list, dst_ip_list, src_port_list, dst_port_list,
#protocol_list,src_mac_list,dst_mac_list, ttl_list, ip_flags, tcp_flags)
length_graph(packet_list)
length_hist(length)

#press Enter to exit!

#raw_input()
'''
#commented
while input != "0":
    print("(0) Quit")
    print("(1) Print packets summary")
    print("(2) Print list of unique source IPs")
    print("(3) Print list of unique destination IPs")
    print("(4) Print list of unique source MACs")
    print("(5) Print list of unique destination MACs")

    input=raw_input('Enter a choice (index): ')
    if input == "1":
        #select=raw_input('a) All b) index: ')
        for pkt in packet_list:
            print(pkt)
    elif input == "2":
        for ip in src_ip_list:
            print(ip)
    elif input == "3":
        for ip in dst_ip_list:
            print(ip)
    elif input == "4":
        for ip in src_mac_list:
            print(ip)
    elif input == "5":
        for ip in dst_mac_list:
            print(ip)
    elif input == "0":
        print("bye bye")
    else:
        print("Wrong input!!!")
'''

except Exception, e:
    print("Some Exception",e)
12 Bibliography

References


[14] Lorenzo Franceschi-Bicchierai. How 1.5 million connected cameras were hijacked to make an unprecedented botnet. 2016.


[34] Brock Thompson. 3 ways to protect your foscam from hackers. 2014.


