

**IOT Honeynet Framework**

*Timothy McCann*

*Jason Swords*

*Raigridas Bartkus*

Department of Informatics, School of Informatics and Engineering,

Technological University Dublin

Submitted to Technological University Dublin in partial fulfilment of the requirements for the degree of

*B.Sc. (Hons) Digital Forensics and Cyber Security*

Supervisor:

Mark Cummins

21/05/2019

# **Declaration**

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Degree of Honours B.Sc. in Digital Forensics and Cyber Security in the Technological University Dublin – Blanchardstown Campus, is entirely my own work except where otherwise stated, and has not been submitted for assessment for an academic purpose at this or any other academic institution other than in partial fulfilment of the requirements of that stated above.

Signed:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Dated: \_\_\_\_/\_\_\_\_\_/\_\_\_\_\_

Signed:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Dated: \_\_\_\_/\_\_\_\_\_/\_\_\_\_\_

Signed:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Dated: \_\_\_\_/\_\_\_\_\_/\_\_\_\_\_

# **Summary**

# **Acknowledgements**

# **Abstract**

**Contents**

[**Declaration** 2](#_Toc8129105)

[**Summary** 3](#_Toc8129106)

[**Acknowledgements** 4](#_Toc8129107)

[**Abstract** 5](#_Toc8129108)

[**List of Tables** 8](#_Toc8129109)

[**List of Figures** 8](#_Toc8129110)

[**Chapter 1: Introduction** 9](#_Toc8129111)

[**Chapter 2: Current Technologies** 10](#_Toc8129112)

[**2.1 The Modern Cyber-Threat Landscape** 10](#_Toc8129113)

[**2.1.1 Cyber Attacks** 10](#_Toc8129114)

[**2.1.1.1 Attack Factors** 10](#_Toc8129115)

[**2.1.1.2 Types of Attacks** 10](#_Toc8129116)

[**2.1.1.3 Syntactic Attacks** 12](#_Toc8129117)

[**2.1.1.4 Semantic Attacks** 12](#_Toc8129118)

[**2.1.1.5 Attackers** 12](#_Toc8129119)

[**Chapter 3: Problem Formulation** 12](#_Toc8129120)

[**Chapter 4: Design** 12](#_Toc8129121)

[**4.1 Hosting Service** 12](#_Toc8129122)

[**4.1.1 Understanding Amazon EC2 Features** 13](#_Toc8129123)

[**4.1.1.1 Setting Up Amazon EC2** 13](#_Toc8129124)

[**4.1.1.2 Virtual Private Cloud (VPC’s)** 16](#_Toc8129125)

[**4.1.1.3 AMI’s and Instances** 16](#_Toc8129126)

[**4.1.1.4 Regions and Availability Zones** 17](#_Toc8129127)

[**4.1.1.5 Amazon EC2 Key Pairs** 17](#_Toc8129128)

[**4.1.1.6 Security Groups** 18](#_Toc8129129)

[**4.1.2 VPC Flow Logs and CloudWatch** 18](#_Toc8129130)

[**4.1.2.1 VPC Flow Logs** 19](#_Toc8129131)

[**4.1.2.2 Amazon CloudWatch** 21](#_Toc8129132)

[**Chapter 5: Implementation** 22](#_Toc8129133)

[**5.1 Deploying AWS EC2 Instance** 22](#_Toc8129134)

[**5.1.1 Requirements and Setup** 23](#_Toc8129135)

[**5.1.2 Creating an EC2 Instance** 23](#_Toc8129136)

[**5.1.2.1 Creating a KeyPair** 25](#_Toc8129137)

[**5.1.2.2 Creating a Security Group** 26](#_Toc8129138)

[**5.1.2.3 Managing EC2 Instances** 27](#_Toc8129139)

[**AWS** 28](#_Toc8129140)

[**Bibliography** 28](#_Toc8129141)

# **List of Tables**

# **List of Figures**

# **Chapter 1: Introduction**

# **Chapter 2: Current Technologies**

# **2.1 The Modern Cyber-Threat Landscape**

## **2.1.1 Cyber Attacks**

### **2.1.1.1 Attack Factors**

### **2.1.1.2 Types of Attacks**

Cyber-attacks can be broken down into two categories, active and passive. Active and passive attacks can target a whole network or an individual host, depending on the target different attacks can be deployed in each scenario. An attacker can use a combination of both attack techniques to gain access to a system, network or data. Often a passive attack is launched prior to an active attack to perform a reconnaissance of the target and gather information that can reveal the vulnerabilities and weaknesses of the target.

1. **Active Attack**

An active attack is an attack that intercepts and modifies the information gathered during an attack phase. These types of attacks are often aggressive, and victims are usually aware when this attack is taking place. Active attacks are highly malicious in nature, often locking out users, destroying memory or files, or forcefully gaining access to a target system or network [25]. Syntactic attacks are examples of an active attack and are discussed further in 2.1.1.3. Other examples of an active attack include man-in-the-middle attacks, buffer overflows and Denial of Service attacks.

* **Man-in-the-middle**

A man-in-the-middle (MITM) attack also known as a Janus attack is an “active form of eavesdropping in which the attacker makes independent connections with victims and relays messages between them making them believe that they are in contact privately” [28].

* **Buffer overflow**

Buffer overflow is a well-known attack dating back as far as 1988 when it was accidentally discovered by a graduate student. A buffer overflow attacks work by overrunning a buffers boundary and overwriting the adjacent memory locations causing the system to crash or perform in an unpredictable way. “Overflow attacks exploit a lack of bounds checking on the size of input being stored in a buffer array” [23].

* **Denial of Service**

This attack focuses on crashing a system or making a system unusable or unavailable to legitimate users [28]. The attack exploits weaknesses in TCP/IP (Transmission Control Protocol/Internet Protocol) protocols and can be launched with minimum effort and can be very difficult to trace back to the attacker. Denial of Service attack can also be used to corrupt or in some cases delete data [26].

1. **Passive Attack**

“An attack in which an unauthorized attacker eavesdrops on the communication between two parties in order to steal information stored in a system by wiretapping or similar means” [28]. The eavesdropper however does not make any changes to the data gathered and it is this feature that separates a passive attack from an active attack. Passive attacks are often viewed as non-disruptive methods of gathering information about a victim or a company who most of the time are unaware that the passive attack is even taking place. The goal of a passive attack is to collect data while remaining anonymous and silent [25]. Examples of a passive attack include port scanning using tools like Nmap and key logging by installing some sort of malware on the victim’s system.

* **Wiretapping**

Wiretapping or passive wiretapping refers to the monitoring or recording of data as its being transmitted over a communication medium, without altering or changing that data [24].

* **Port Scanning**

Is a type of a Reconnaissance attack in which an attacker probes a network or a host to learn which ports are available and the services associated with the network or the host [28]. Ports found can be both closed or open and the goal of a port scan if to find an open port that is vulnerable to an exploit. A common tool used to perform port scanning is Nmap or a GUI version of this known as Zenmap.

* **Key Logging**

Key logging represents a serious threat to the privacy and security of today’s systems. A key logger is a malicious program that runs stealthily in a background on a user’s computer and collects the sensitive information about that user such as the user passwords, credit card details and any other personal information. Many anti-virus software fails to detect a key logger running on a user’s system and a user has no way to determine if their input on the keyboard is being recorded often resulting in the user becoming a victim of identity theft and fraud [22].

### **2.1.1.3 Syntactic Attacks**

“Syntactic attacks use virus-type software to disrupt or damage a computer system or network” [29]. Syntactic attacks are commonly referred to as “malware” because they are considered as malicious software. These attacks may include viruses, worms and Trojan horses.

* **Viruses**

Viruses are malicious executable files that can copy themselves and infect a computer through a removable medium like a USB key or through email files. However, viruses need user intervention to execute. A virus can hide in locations least accessed by a user in the memory of a computer system to establish persistence and can attach itself to whatever file it wants too insert its own code into that program [30].

* **Worms**

Worms are self-sustaining and replicating programs that can independently exploit vulnerabilities in a network and requires no user intervention to execute. A worm’s main goal is to replicate itself across the whole system and establish persistence and a worm often uses network vulnerabilities to achieve this. All worms are harmful and have the ability to corrupt, delete or encrypt files creating the possibility for further attacks such as ransomware, if a worm is not configured with a payload it can still caused damage by consuming the bandwidth of a network. An example of this is the Morris worm [30].

* **Trojan Horses**

These malicious programs are malware that appear legitimate and mislead the user into thinking that it will perform a desirable function but in fact facilitate unauthorised access to the user’s computer. Trojan horses often aim at exploiting authorization and granting the attacker access to the victim’s computer where the attacker can then install additional malware on the victim’s system. Trojan horses are unlike viruses and worms as they do not try to insert themselves into files, and don’t focus on propagation. Nevertheless, a Trojan horse has some similarities to the previously discussed malware, for example it disguises itself as a useful program in hopes that it will be executed by a user and therefore requires user intervention which makes it similar to viruses [30].

### **2.1.1.4 Semantic Attacks**

### **2.1.1.5 Attackers**

# **Chapter 3: Problem Formulation**

# **Chapter 4: Design**

## **4.1 Hosting Service**

Amazon Web Services (AWS) is a subsidiary of Amazon Inc. and describes itself as a “secure cloud services platform, offering compute power, database storage, content delivery and other functionality” [2]. Amazon EC2 describes itself as being able to provide “scalable computing capacity in the Amazon Web Services (AWS) cloud” [3]. Amazon EC2 allows developers to deploy as many or as few virtual servers as is needed, configure their networking and security, and manage their storage.

Amazon EC2 was chosen due to its flexibility, making it possible to deploy and manage virtual servers on demand. Other hosting providers such as Digiweb, Microsoft Azure and DigitalOcean were considered but Amazon EC2’s flexibility and prestige as being an industry leader in providing cloud hosting services resulted in it being chosen for this project.

Another reason for choosing Amazon for this project was that Amazon has its own SDK called boto3 which allows system administrators to automate a lot of tasks. The boto3 library is used to aid in providing a holistic approach for the user by allowing them to scan a device, deploy it as a honeypot and then manage these honeypots all from within the IOT Honeynet Framework.

### **4.1.1 Understanding Amazon EC2 Features**

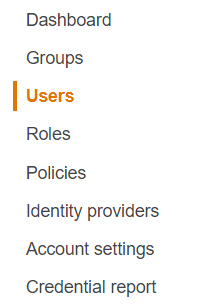
As previously mentioned, Amazon EC2 describes itself as being able to provide scalable computing capacity in the Amazon Web Services (AWS) cloud. Amazon EC2 instances can be provisioned under the AWS 12-month free tier to facilitate the frameworks need to deploy multiple instances. The features that make up Amazon EC2 and their relevance to the project are explored in further detail in the following sections.

It is important to understand the different features provided by Amazon EC2 to comprehend the design of the IOT Honeynet Framework and all of its associated features which help researchers and users to seamlessly scan devices and deploy them as honeypots quickly and efficiently. There are a number of requirements that must be satisfied in order to allow such an IOT Honeynet Framework to function properly. It is for this reason, that the features of Amazon EC2 must be explored and understood.

#### **4.1.1.1 Setting Up Amazon EC2**

Using a free tier EC2 instance first involves setting up an AWS user account. Once created, an account has access to all AWS services automatically including Amazon EC2 [1]. With an account now setup, the next step is to create an IAM user. Services in AWS, like Amazon EC2, require users to provide credentials when accessing them so it can be determined whether a user has permission to access that service or not [1]. It is not recommended that an AWS service be accessed using the default AWS account login credentials but rather by using the AWS Identity and Access Management instead, which is necessary to allow services to accessed programmatically [1].

Adding a user involves a few simple steps. Once logged into an AWS account, launch the Identity and Access Management console. Click *“Users”* on the navigation menu on the left of the screen as seen in the below figure.



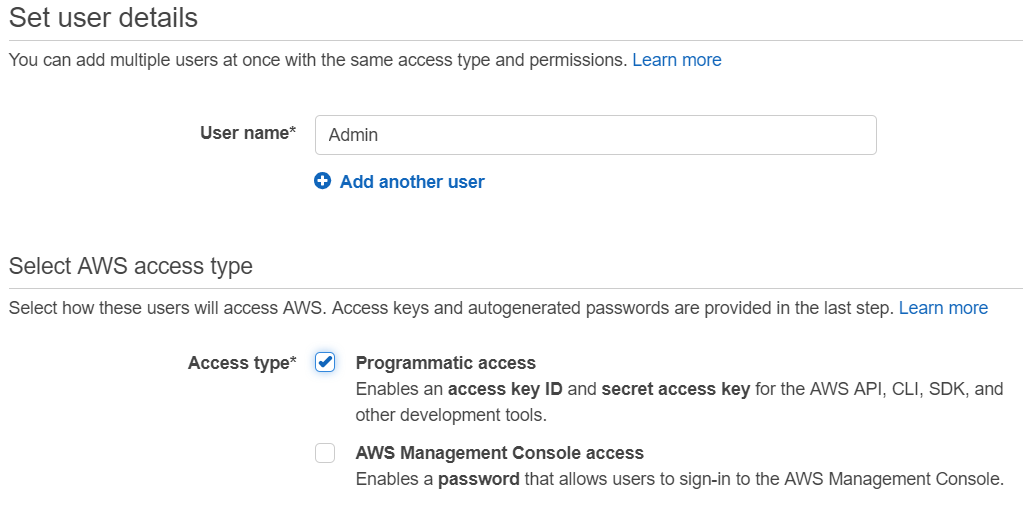
*Figure : IAM Navigation Menu.*

Next, in the popup window choose *“Add User”*.



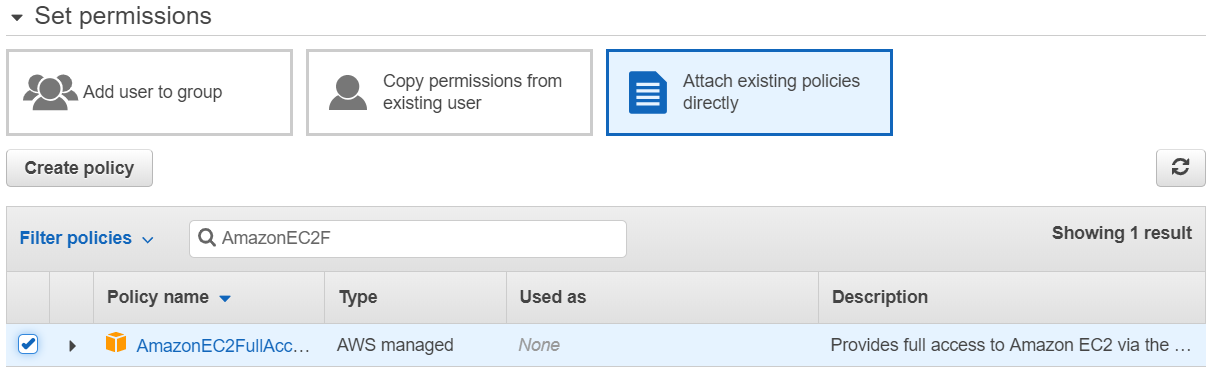
*Figure : IAM Add User.*

This will present a new window that requests a user name and the type of access this user should have. The type of access this user is allowed is important for when a user is working through the IOT Honeynet Framework. Programmatic access is essential to using the AWS CLI, a core feature of the IOT Honeynet Framework that will be explored in more detail later. For the purposes of this project it is necessary for the user with administrator privileges to have programmatic access, which is the option selected as can be seen in the figure below.



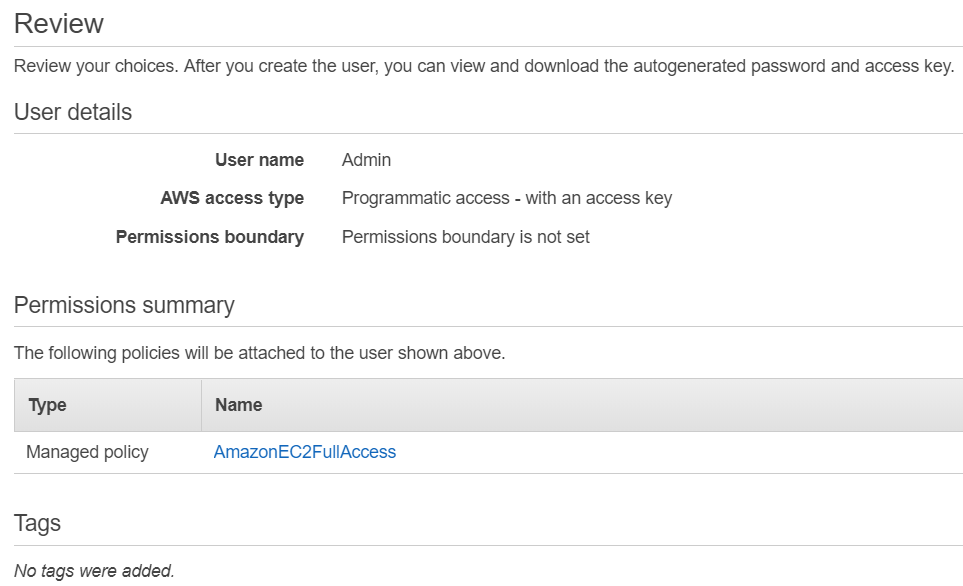
*Figure : IAM Username and Access Type.*

The next step is to determine what permissions the user should have. It is possible to create different groups to divide users based on what access privileges they are allowed to have. In order for a user to be able to fully interact with their instances and manage them, the iAM user they are setting up must have administrator level privileges. The policy titled *“AmazonEC2FullAccess”* was selectedand attached to the user which can be seen in the figure below.



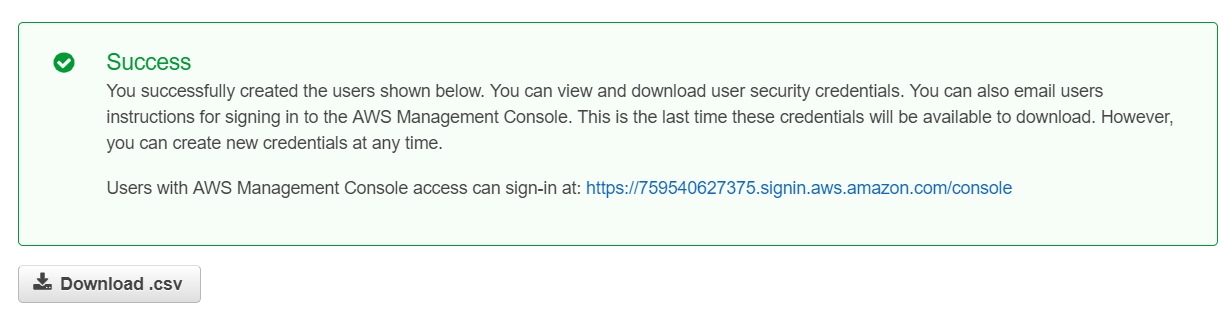
*Figure : IAM user permissions.*

Finally, the optional choice to add tags to the user is presented which can be used to assign metadata to help track, organize and control access for the user [2]. After this, the user details and permission levels can be reviewed before finally creating the user.



*Figure : Review IAM User Details.*

When the user is created, a window is presented which shows the user’s access key id and access secret key. These are only available once, so it’s important to download and save them safely in a secure location.



*Figure : IAM User Created Successfully.*

Now that an IAM user is created, it is possible to programmatically access Amazon EC2 service, which is essential to allowing the IOT Honeynet Framework to remotely access and manage EC2 services. This will be explored more later in the Implementation Chapter.

#### **4.1.1.2 Virtual Private Cloud (VPC’s)**

Amazon Virtual Private Cloud (VPC) is defined as “creating a virtual network in your own logically isolated area within the AWS cloud, known as a virtual private cloud (VPC)” [9]. Amazon EC2 instances can be launched in a VPC which acts similarly to a traditional network but with the added bonus of being able to use scalable infrastructure from AWS [9]. When an AWS account is created, a default VPC is created in the region specified when setting up the AWS account, enabling users to instantly launch instances.

It’s important to understand the concepts of VPC as it is the networking layer for Amazon EC2 instances. Amazon kindly provides every user who sets up an account with a default VPC which is configured and ready for use. The default VPC conveniently includes an internet gateway, and each default subnet is a public subnet [10]. Each EC2 instance that is launched on a default subnet has a private IPv4 address and a public IPv4 address. These instances can communicate with the internet through the internet gateway. An internet gateway enables your instances to connect to the internet through the Amazon EC2 network edge [10].

It is possible to create nondefault VPC’s, but it was decided to work of the default VPC since it satisfies the requirements needed by the IOT Honeynet Framework, which is to allow honeypots access to the internet and to collect logging information, something that will be discussed in more detail later.

#### **4.1.1.3 AMI’s and Instances**

According to Amazon, an Amazon Machine Image (AMI) “is a template that contains a software configuration (for example, an operating system, an application server, and applications)” [4]. It is from this image that an instance is launched, which is a copy of a virtual server running in the cloud. There are multiple AMI’s that can be selected, with some being a part of the AWS 12-month free tier, while others being more expensive.

An important consideration to make when launching an instance is the type of instance that is being launched. The instance type that is specified determines the hardware of the host computer used for your instance, offering different compute, memory, and storage capabilities [6]. These instance types are often grouped based on their capabilities and can dramatically impact the performance of the application or software being deployed on the instance.

For the base operating system of each honeypot, it was decided to only use the free tier AMI’s and allow the user to select which operating system they would like each honeypot to run. This approach allows the user more options while also keeping in line with the AWS 12-month free tier contract. The different instance types that were allowed for a user to select from were kept to a small t2.micro general-purpose instance type. This was chosen based on the minimum resources that were required for each honeypot to operate in the Honeynet framework, meaning more honeypots could be deployed without unnecessarily using up resources.

#### **4.1.1.4 Regions and Availability Zones**

According to Amazon, Amazon EC2 can be hosted over multiple locations world-wide, with these locations being composed of Regions and Availability Zones [5]. Each Amazon region is designed to be completely isolated from other Amazon EC2 Regions. Each region is then comprised of isolated locations known as Availability Zones [5]. This is meant to achieve fault tolerance and stability, two very important requirements for the hosting service of a Honeynet as any downtime could result in valuable data being lost for researchers.

The choice of what region to deploy the Amazon EC2 instances could greatly impact the type of results and data gained by a researcher. It is an important consideration to make when determining what region to deploy Amazon EC2 services in. The goal of the IOT Honeynet Framework is not to make these decisions for the researcher but to rather support the ease of deploying and managing instances for the researcher. It is with this in mind that it was decided to allow the user to make this decision when initially setting up an AWS account.

#### **4.1.1.5 Amazon EC2 Key Pairs**

A key pair refers to the public and private key used to encrypt and decrypt data, which is used by Amazon EC2 to encrypt and decrypt login information [7]. This enables secure remote access into an EC2 instance. Amazon stores the public key while the private key is kept by the user which is why it is important to keep the private key secure as anyone with access to the key can decrypt the login information of any instance associated with that particularly private key. The keys that Amazon EC2 uses are 2048-bit SSH-2 RSA keys and it is possible to have up to five thousand key pairs per Region [7].

It is important to create a key pair to securely access a honeypot instance from a remote location once it has been created. Once created it is necessary to change the mode of the key pair file to read-only otherwise it will be denied access. It is possible to create key pairs in the IOT Honeynet Framework which are stored in a file with a .pem extension and then later used when creating the instance. This will be demonstrated programmatically in the Implementation Chapter, but it is necessary to understand the importance of the EC2 key pair for logging in securely to newly created instances and is instrumental for setting up the python server used for the honeypot.

#### **4.1.1.6 Security Groups**

Amazon describes how a security group “acts as a virtual firewall that controls the traffic for one or more instances” [8]. When an instance is launched, one or more security groups can be assigned to it. A security group contains rules which dictates what traffic is allowed to and from an instance. Security groups are associated with network interfaces and when changing an instance's security groups, it changes the security groups associated with the primary network interface (eth0) [8].

The security group rules dictate what inbound traffic can reach an instance and what outbound traffic can leave an instance. There are a number of key characteristics that a security group has which are important in acknowledging when deploying an instance:

1. Security Groups allow all outbound traffic [8].
2. It is not possible to create rules that deny access [8].
3. Security groups are stateful meaning it tracks the operating state and characteristics of network connections traversing it. The firewall is configured to distinguish legitimate packets for different types of connections [8].
4. Rules can be added and removed at any time [8].

A rule in a security group is created with several parameters. When the IOT Honeynet Framework creates the security group, the protocol type, source and destination port, and the source address parameters are used. This will be demonstrated further in the Implementation Chapter. It is essential to know how to configure a security group to open ports for the honeypot based on what scan results are provided by the user [8].

### **4.1.2 VPC Flow Logs and CloudWatch**

VPC Flow Logs and CloudWatch are separate topics from Amazon EC2 and deserve their own section so as to better elaborate on their importance to the IOT Honeynet Framework. Logging is an important and vital function of any Honeynet Framework service. For this thesis, it was evident that if Amazon EC2 was being used by the IOT Honeynet Framework then VPC Flow Logs and Amazon CloudWatch were going to be used for collecting log information from each honeypot. It is therefore important to understand the process to setup each feature and how both relate to each other. It is also essential to be able to understand how to read the information that is being stored, hence why it is necessary to provide a separate section for both VPC Flow Logs and CloudWatch.

#### **4.1.2.1 VPC Flow Logs**

VPC Flow Logs is a feature that captures information about the IP traffic going to and from network interfaces on a VPC. This data is then published to either Amazon CloudWatch Logs or Amazon S3 [11]. Setting up a flow log involves specifying a resource to capture data from, the type of traffic to capture and where to send the flow log data once it is collected.

An important point to note is that if a flow log is setup for a VPC with multiple instances on it then each network interface on the VPC is monitored and stored in that flow log [11]. This data that is collected is stored in what is called flow log records which consists of fields that describe the traffic being monitored on a network interface.

The structure of the flow log is quite detailed and so requires users to understand the different fields when examining a log record. A flow log record represents a network flow in your flow log and is a space-separated string that has the following format [11]:

***<version> <account-id> <interface-id> <srcaddr> <dstaddr> <srcport> <dstport> <protocol> <packets> <bytes> <start> <end> <action> <log-status>***

Amazon [11] provides a table containing the description for each field which is displayed below:

|  |  |
| --- | --- |
| Field | Description |
| version | The VPC Flow Logs version. |
| account-id | The AWS account ID for the flow log. |
| interface-id | The ID of the network interface for which the traffic is recorded. |
| srcaddr | The source IPv4 or IPv6 address. The IPv4 address of the network interface is always its private IPv4 address. |
| dstaddr | The destination IPv4 or IPv6 address. The IPv4 address of the network interface is always its private IPv4 address. |
| srcport | The source port of the traffic. |
| dstport | The destination port of the traffic. |
| protocol | The IANA protocol number of the traffic. |
| packets | The number of packets transferred during the capture window. |
| bytes | The number of bytes transferred during the capture window. |
| start | The time, in Unix seconds, of the start of the capture window. |
| end | The time, in Unix seconds, of the end of the capture window. |
| action | The action associated with the traffic:  **ACCEPT**: The recorded traffic was permitted by the security groups or network ACLs.  **REJECT**: The recorded traffic was not permitted by the security groups or network ACLs. |
| log-status | The logging status of the flow log:  **OK:** Data is logging normally to the chosen destinations.  **NODATA:** There was no network traffic to or from the network interface during the capture window.  **SKIPDATA:** Some flow log records were skipped during the capture window. This may be because of an internal capacity constraint, or an internal error. |

*Figure : Flow Log Record field Definitions Table.*

This format for a log record can appear to be quite complex and detailed at first but it provides a wealth of information to a user who wishes to see what traffic is being sent to their honeypots. A sample log that was stored for one of the test honeypots used during the development of the IOT Honeynet Framework can be used as an example to showcase what information is being gathered. Below we can see a sample log:



*Figure : Sample Flow Log Record.*

The first three fields relate to account information and displays the VPC Flow Logs version (2), the AWS account ID for the flow log (759540627375), and the ID of the network interface for which the traffic is recorded (eni-038a70d3b9da88ecd). This information doesn’t serve much purpose for researchers, but it is was deemed necessary to be aware of these fields in order to be thorough in explaining flow log records.

The following fields are of more interest to researchers, which contain the vital data that can be used later to determine what kind of attacks are being used and where these attacks are coming from. The source address in this flow log record is 46.209.123.106. A WHOIS lookup conducted using the website BigDomainData [12] showed that the address was from the geolocation of the Islamic republic of Iran. The next field shows the private IP address (172.31.38.66) of the honeypot that received this traffic. The next two field are the source port (4746) and destination port (445) respectively. The next field shows the IANA protocol number 6 which represents the type of protocol used, which in this case according to iana.org is the TCP protocol [13]. The following field represents the number of packets sent, which according to this log is one. This is followed by the byte size of the packet which was 52. The log also provides the start and end times of the capture window in Unix seconds. Finally, the action and log-status are recorded by the log which shows that the action taken against the packet was to reject it, and then store log successfully (OK) to its intended destination with the accompanying data.

For the IOT Honeynet Framework, it was debated on how to display the logs to the user. As an essential component of the IOT Honeynet Framework, it was decided to provide the log records as provided by Amazon Flow Logs to the user. This would allow users to observe and work with all provided information in a log. This avoided the possibility of hiding or withholding information that a researcher/user may want access to. However, this could also hinder a user’s ability to interpret the log information and cause relevant data to be obscured by irrelevant data. In the end the decision to provide all given information was taken to ensure all log information was provided to the user and could be used at their discretion.

#### **4.1.2.2 Amazon CloudWatch**

As mentioned early, when setting up VPC flow log there are a number of steps taken, one of which is to choose where to send the flow log data once it is collected. There are two options available when considering where to send the flow log data once it is collected which include Amazon Simple Storage Service (S3) and Amazon CloudWatch.

Amazon S3 is designed to allow a user to store and retrieve any amount of data using a web interface from anywhere on the web [14]. Amazon S3 provides a means for customers of any size in the industry to store data and then take advantage of easy-to-use management features to organize the stored data and finely tune access controls [14].

Amazon CloudWatch is promoted by Amazon as “a monitoring and management service built for developers, system operators, site reliability engineers (SRE), and IT managers” which “collects monitoring and operational data in the form of logs, metrics, and events, providing you with a unified view of AWS resources, applications and services that run on AWS, and on-premises servers” [15].

For this thesis, it was decided to use Amazon CloudWatch over Amazon S3 due to Amazon CloudWatch being designed to visualize logs, troubleshoot issues and analyse system metrics. This is in comparison to Amazon S3, which focuses on storing data from applications which can then be used in a variety of use cases such as backup and recoveries or big data analytics [14]. The process for implementing CloudWatch with VPC Flow Logs will be explored in more detail in the Implementation chapter.

# **Chapter 5: Implementation**

The IOT Honeynet Framework takes a holistic approach by taking the desired features which include scanning a device specified by the user, then making a clone of that device and finally deploying it as a honeypot. There are a lot of steps and elements at work in order to make these features operate correctly as intended. This chapter describes the steps and processes taken to achieve the desired features outlined above and follows the methodology and thought process used.

## **5.1 Deploying AWS EC2 Instance**

Deploying an EC2 instance through an IOT Honeynet Framework requires a programmatic approach. This requires the framework to be able to securely connect to a user’s AWS account and deploy an EC2 instance with all the specified requirements requested by the user. Here, the requirements and the process of deploying an EC2 instance through a web framework will be discussed in detail.

### **5.1.1 Requirements and Setup**

To allow the IOT Honeynet Framework to create and manage EC2 instances, the Boto3 library in Python and the AWS Command Line Interface (CLI) tool need to be installed. After this, it’s necessary to create an IAM user using the steps described in the design chapter. This is important as the credentials for the user (access key and access id) are needed to allow the IOT Honeynet Framework to interact with a user’s AWS account. These requirements should be more clearly elaborated to better understand their role in enabling the framework to achieve the desired features outlined earlier [16].

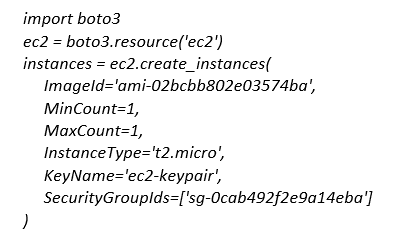
The Boto3 library in Python is, according to the boto3 documentation, “the Amazon Web Services (AWS) SDK for Python” [17]. It allows developers to create, configure and manage amazon services such as EC2 by providing an easy to use, object-oriented API, as well as low-level access to AWS services [17]. This library is used extensively throughout the IOT Honeynet Framework to provide many of the essential functions such as displaying logging information and creating new EC2 instances. The entirety of boto3’s contribution will be outlined in the coming sections, but it is important to note it’s integral part in the project.

Amazon outlines the AWS CLI as “an open source tool that enables you to interact with AWS services using commands in your command-line shell” [18]. Once a user with the necessary credentials (Access Key, Access ID) has been created, it is possible to configure the Python scripting environment with these credential in order for the IOT Honeynet Framework to manage EC2. The “*aws configure*” command is used to set the credentials and other configuration details which include the region and output format. Once those are provided, credentials are saved in a local file at path “*~/.aws/credentials*” and other the configurations for region and output format are stored in “*~/.aws/config*” [16].

In order for the IOT Honeynet Framework to operate correctly, the user must have an AWS account with the associated credentials setup. The python boto3 library and the AWS CLI must be installed for the IOT Honeynet Framework to be able to manage Amazon EC2 instances. Once the python environment is setup with the users AWS credentials, the IOT Honeynet Framework can operate as designed with all intended features.

### **5.1.2 Creating an EC2 Instance**

Creating an EC2 instance with boto3 library is a relatively straight forward process. The *create\_instances()* function in the boto3 library is used to create an instance. This function has many parameters some of which are required to create the instance while others are optional. Below is the script used by the IOT Honeynet Framework to create an instance.



*Figure : Python Script to Create EC2 instance.*

The script is broken down from top to bottom as follows. The first step is importing the boto3 library and then create a resource object which according to the boto3 documentation creates an object-oriented interface to Amazon Web Services (AWS). To use resources, you invoke the *resource()* method of a Session and pass in a service name which in this case is ‘ec2’ since the script is creating an EC2 instance [19].

Once the boto3 library is imported and a EC2 resource object for that session is created, the *create\_instances()* function can be used to deploy an instance. The first parameter in the function specifies the ID of the Amazon Machine Image (AMI) to be used when the instance is launched. This AMI, for the purposes of this project, indicates the operating system of the honeypot being deployed [20]. In the above script, the AMI image ID specified is the Amazon Linux 2 free tier AMI (64-bit x86).

The MinCount and MaxCount parameters specify the number of instances to deploy using an integer. In the above script the MinCount and MaxCount are set to one so only a single instance is deployed each time a script is used. Both parameters are required by the function in order to deploy an instance [20].

The instance type parameter defines the hardware of the EC2 instance being deployed. In this script the t2.micro instance type is specified. The decision to use this instance type was explained in the Design Chapter under the AMI and Instances heading (3.1.1.3) [20].

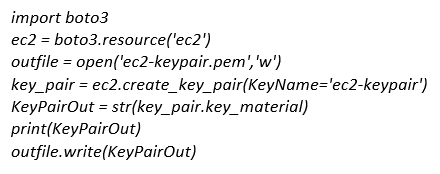
The parameter entitled KeyName refers to the name of the KeyPair that allows a user to access an instance once it is launched. It is important to note that if a KeyPair is not specified when an instance is launched then a user will not be able to remotely login to the instance unless the AMI used enables the user to login through other means [20]. This is an essential parameter that uses a resource that will be created in another script, which will be discussed below briefly in another subsection (5.1.2.1).

The final parameter labelled SecurityGroupIds indicates the security group to use when deploying the EC2 instance. It is worth noting that when a security group id is not specified, then a default security group is assigned to the instance being launched [20]. Like the KeyName parameter, this parameter uses a resource that is created using a different script which will be discussed in more detail below in another section (5.1.2.2).

With all the above parameters satisfied, an EC2 instance will be deployed. It is worth mentioning that there are a few rules that apply when deploying an EC2 instance. If an instance is launched without a subnet specified, then a default subnet is taken from the user’s default VPC. Each instance when launched has a primary private IP address that, if not specified, is chosen from the IPv4 range of the subnet assigned to the instance. Finally, if an AMI is selected for which a user has not subscribed to then the launch will fail [20].

#### **5.1.2.1 Creating a KeyPair**

According to the boto3 documentation the *create\_key\_pair()* is used to create a 2048-bit RSA key pair with the specified name [20]. Amazon EC2 stores the public key and displays the private key for a user to save to a file. The script below shows the process of creating a key pair and storing it locally in a file.



*Figure : Python Script to Create Key Pair.*

The script begins by importing the boto3 library and then creates a resource object used for the current session. A file called ‘ec2-keypair’ with the .pem file is created. Next the key pair is created in AWS with the *create\_key\_pair()* which uses the required parameter KeyName to assign a unique name to the key pair [20]. Finally, the key is written out to the ‘ec2-keypair.pem’ file created earlier.

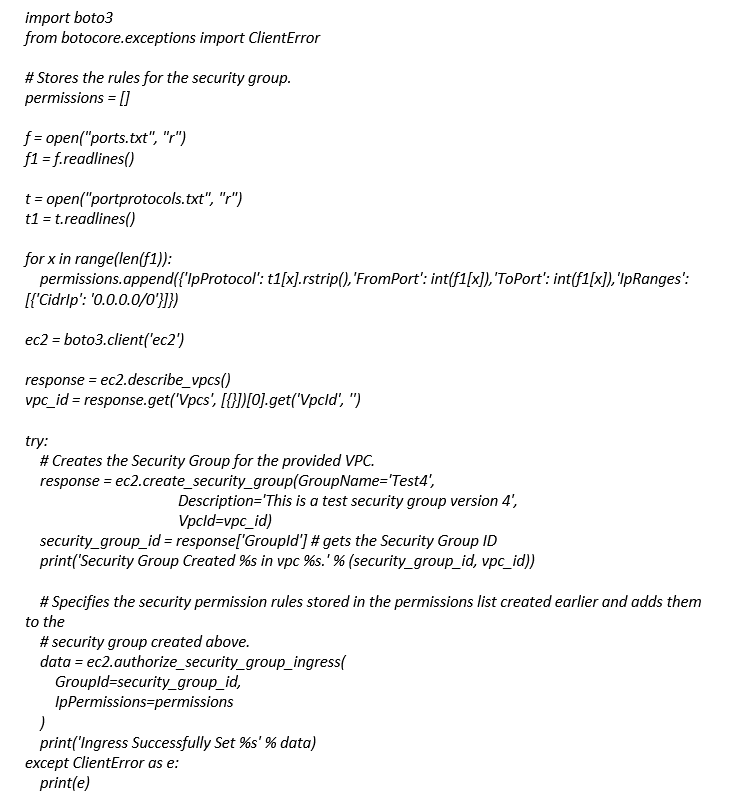
An important final step to take when the key pair is created is to change the mode of the key pair file to read-only otherwise it will be denied access. For Linux this accomplished by simply using the below command:

*chmod 400 <name of file with private key>*

This key pair can then be used in the above script that is used in creating instances, so that when an instance is created a user can remotely login to it.

#### **5.1.2.2 Creating a Security Group**

Creating a security group involves using two functions provided by the boto3 library. The first function called *create\_security\_group()* creates the security group and the second function called *authorize\_security\_group\_ingress()* adds one or more ingress rules to a security group [21]. As previously explained in the Design Chapter, a security group acts as a virtual firewall. When deploying a honeypot, it is necessary to ensure that the ports picked up by the scanning script are open on the EC2 instance in order to create a clone of the device the user scanned. The below python script uses the results from the scanning script and creates a security group with the desired ingress rules.



*Figure : Python Script to Create Security Group and Add Ingress Rules.*

The script starts by importing some libraries. A python list called permissions is created which stores the rules for the security group. Two files are open and read which contain port numbers and port protocol types associated with the port numbers, both of which were generated by the scanning script. These are used in conjunction with a for loop to dynamically create rules for the security group according to how many ports were picked up by the scanning group. The rules defined consist of the protocol type (TCP, UDP) associated with a port, the port range that the rule applies to which is specified by the FromPort and ToPort parameters and the CIDR range that the CIDR range applies to. In this script the CIDR range is set to allow any device to connect to these ports since a honeypot is designed to pick up as much traffic as possible. These new rules are then stored in the permissions list as python dictionaries.

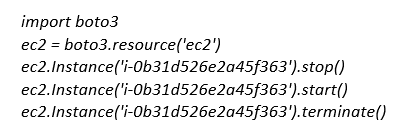
An EC2 resource project is then created for the session. Since the security group is being created for the user’s default VPC, it is necessary to specify the VPC ID when creating the security group. The *describe\_vpcs()* functionlists all the VPCs a user has. The first VPC is selected which is the default VPC unless the user has created a different VPC. Once the default VPC ID a try and except block is created, which contains the *create\_security\_group()* and *authorize\_security\_group\_ingress()* functions mentioned earlier.

The *create\_security\_group()* function takes in three parameters which are required according to the boto3 documentation [19]. The first parameter GroupName defines the name of the security group, the second parameter GroupDescription specifies a description for the security group and the VPCid parameter specifies the VPC the security group is being made for [19]. Once the security group is created, the security group id is retrieved and will be used in the *authorize\_security\_group\_ingress()*.

In the python script above the *authorize\_security\_group\_ingress()* function takes in two parameters which includes GroupId and IpPermissions parameters. The GroupId contains the security group ID retrieved earlier after the security group is created and the IpPermissions contains the list of permissions created earlier in the script. The IpPermissions parameter takes in a list of dictionaries hence why they are stored as such when created [19]. Once the script is completed, a security group with the desired ingress rules are created.

#### **5.1.2.3 Managing EC2 Instances**

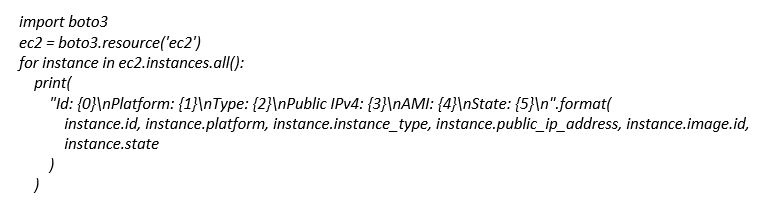
Once an EC2 instance is created, it is obviously desirable to allow the user to remotely manage their EC2 instances (honeypots) from the IOT Honeynet Framework, which is working in line with the holistic approach the IOT Honeynet Framework is designed to achieve. Achieving this in a programmatic way is quite simple when using the python boto3 library. The IOT Honeynet Framework uses this library to programmatically manage EC2 instances, allowing the user to stop, start, list and terminate instances.

**

*Figure : Python Script to Manage Instance States*

To change the state of an instance, the instance ID needs to be passed to the *instance()* function and then use the *stop(), start()* or *terminate()* method depending on the desired action the user wishes to take with an instance.

To list all EC2 instances (honeypots) currently created, an ec2 resource will be created. This EC2 resource is used to iterate through all EC2 instances, then access the individual properties of each virtual machine and print the list of all EC2 instances and their respective properties on the console.

**

*Figure : Python Script to List Honeypots.*

In the above script, the instance id, platform, instance type, public IP address, AMI and State properties are listed for each honeypot that a user has.

## **AWS**

# **Bibliography**

1. Amazon.com. (2019). Setting Up with Amazon EC2 - Amazon Elastic Compute Cloud. [online] Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/get-set-up-for-amazon-ec2.html [Accessed 24 Apr. 2019].
2. Amazon Web Services, Inc. (2019). What is AWS? - Amazon Web Services. [online] Available at: https://aws.amazon.com/what-is-aws/ [Accessed 22 Apr. 2019].
3. Amazon.com. (2019). What Is Amazon EC2? - Amazon Elastic Compute Cloud. [online] Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html [Accessed 22 Apr. 2019].
4. Amazon.com. (2019). Instances and AMIs - Amazon Elastic Compute Cloud. [online] Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instances-and-amis.html [Accessed 24 Apr. 2019].
5. Amazon.com. (2019). Regions and Availability Zones - Amazon Elastic Compute Cloud. [online] Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html [Accessed 24 Apr. 2019].
6. Amazon.com. (2017). Instance Types - Amazon Elastic Compute Cloud. [online] Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html [Accessed 24 Apr. 2019].
7. Amazon.com. (2019). Amazon EC2 Key Pairs - Amazon Elastic Compute Cloud. [online] Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html [Accessed 24 Apr. 2019].
8. Amazon.com. (2019). Amazon EC2 Security Groups for Linux Instances - Amazon Elastic Compute Cloud. [online] Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-network-security.html [Accessed 24 Apr. 2019].
9. Amazon.com. (2013). Virtual Private Clouds - Amazon Elastic Compute Cloud. [online] Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-vpc.html [Accessed 24 Apr. 2019].
10. Amazon.com. (2013). What Is Amazon VPC? - Amazon Virtual Private Cloud. [online] Available at: https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html#what-is-connectivity [Accessed 24 Apr. 2019].
11. Amazon.com. (2019). VPC Flow Logs - Amazon Virtual Private Cloud. [online] Available at: https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs.html [Accessed 24 Apr. 2019].
12. domainbigdata.com (2015). DomainBigData.com - Online investigation tools. [online] Domainbigdata.com. Available at: https://domainbigdata.com/ [Accessed 26 Apr. 2019].
13. Iana.org. (2017). Protocol Numbers. [online] Available at: https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml [Accessed 26 Apr. 2019].
14. Amazon.com. (2019). What Is Amazon S3? - Amazon Simple Storage Service. [online] Available at: https://docs.aws.amazon.com/AmazonS3/latest/dev/Welcome.html [Accessed 26 Apr. 2019].
15. Amazon Web Services, Inc. (2018). Amazon CloudWatch - Application and Infrastructure Monitoring. [online] Available at: https://aws.amazon.com/cloudwatch/ [Accessed 26 Apr. 2019].
16. Singh, P. (2018). How to Create an AWS EC2 Instance with Python. [online] Ipswitch.com. Available at: https://blog.ipswitch.com/how-to-create-an-ec2-instance-with-python [Accessed 26 Apr. 2019].
17. Amazonaws.com. (2019). Boto 3 Documentation — Boto 3 Docs 1.9.136 documentation. [online] Available at: https://boto3.amazonaws.com/v1/documentation/api/latest/index.html [Accessed 26 Apr. 2019].
18. Amazon.com. (2019). What Is the AWS Command Line Interface? - AWS Command Line Interface. [online] Available at: https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-welcome.html [Accessed 26 Apr. 2019].
19. Amazonaws.com. (2019). Resources — Boto 3 Docs 1.9.137 documentation. [online] Available at: https://boto3.amazonaws.com/v1/documentation/api/latest/guide/resources.html [Accessed 27 Apr. 2019].
20. Amazon.com. (2019). Amazon Machine Images (AMI) - Amazon Elastic Compute Cloud. [online] Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html [Accessed 28 Apr. 2019].
21. Amazonaws.com. (2019). Working with Security Groups in Amazon EC2 — Boto 3 Docs 1.9.137 documentation. [online] Available at: https://boto3.amazonaws.com/v1/documentation/api/latest/guide/ec2-example-security-group.html [Accessed 28 Apr. 2019].
22. Al-Hammadi, Y. and Aickelin, U. (2008). Detecting Bots Based on Keylogging Activities. *SSRN Electronic Journal*. [online] Available at: https://arxiv.org/ftp/arxiv/papers/1002/1002.1200.pdf [Accessed 7 May 2019].
23. Cowan, C., Pu, C., Maier, D., Walpole, J., Beattie, S., Grier, A., Wagle, P. and Zhang, Q. (1998). *StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks*. [online] Available at: https://www.usenix.org/legacy/publications/library/proceedings/sec98/full\_papers/cowan/cowan.pdf [Accessed 2 May 2019].
24. Defined Term - A dictionary of defined terms for the legal profession. (2013). *Passive Wiretapping*. [online] Available at: https://definedterm.com/passive\_wiretapping/140598 [Accessed 2 May 2019].
25. DiGiacomo, J. (2017). *Active vs Passive Cyber Attacks Explained | Revision Legal*. [online] Revision Legal. Available at: https://revisionlegal.com/cyber-security/active-passive-cyber-attacks-explained/ [Accessed 2 May 2019].
26. Ieee.org. (2019). *IEEE Xplore Full-Text PDF:* [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=601338 [Accessed 2 May 2019].
27. Singh, P. (2018). How to Manage AWS EC2 Infrastructure with Python. [online] Ipswitch.com. Available at: https://blog.ipswitch.com/managing-aws-ec2-infrastructure-with-python [Accessed 28 Apr. 2019].
28. Uma, M. and Padmavathi, G. (2013). A Survey on Various Cyber Attacks and Their Classification. *International Journal of Network Security*, [online] 15(5), pp.390–396. Available at: <https://pdfs.semanticscholar.org/ba7b/234738e80b027240e9bfd837bfba61c13e17.pdf>.
29. Bhardwaj, M. and Singh, G.P. (2011). Types of Hacking Attack and their Counter Measure. *International Journal of Educational Planning & Administration*, [online] 1(1), pp.43–53. Available at: https://www.ripublication.com/ijepa/ijepav1n1\_7.pdf [Accessed 7 May 2019].
30. Stallings, W., Brown, L., Bauer, M., Howard, M., Columbus, B., New, I., San, Y., Upper, F., River, S., Cape, A., Dubai, T., Madrid, L., Munich, M., Montreal, P., Delhi, T., Sao, M., Sydney, P., Kong, H., Singapore, S. and Tokyo, T. (n.d.). *COMPUTER SECURITY PRINCIPLES AND PRACTICE Second Edition*. [online] Available at: https://412dnet.files.wordpress.com/2016/01/computer-security-principles-and-practice-2nd-edition.pdf [Accessed 7 May 2019].