UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Modeling and Recognizing Network Scanning Activities with Finite Mixture Models and Hidden Markov Models

THESIS

presented and publicly defended on the 20 December 2018

to obtain the

PhD from the University of Lorraine

(Computer Science mention)

by

Giulia De Santis

Composition of the jury

Reviewers:	Philippe Owezarski	Director of Research, LAAS, France
	Valérie Viet Triem Tong	Associate Professor, CentraleSupelec, France
Examiners:	Olivier Festor	Professor, Université de Lorraine, France
	Abdelkader Lahmadi	Associate Professor, Université de Lorraine, France
	Vincent Chevrier	Professor, Université de Lorraine, France
	Ghita Mezzour	Assistant Professor, International University of Rabat, Morocco

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Index

Chapter 1

General Introduction

Chapter 2

Introduction to Network Scanning Activities

Chapter 3

Techniques to detect Network Scanning Activities

Chapter 4

Stochastic models and algorithms

Chapter 5

Long Term Analyses of ZMap and Shodan

List of Figures

1

General Introduction

1.1 Motivations

The huge growth of Internet exposes its users to various threats. This widespread and general vulnerability has been intensified by the large deployment of new devices in addition to traditional computers, including web-cams, smart-phones, control devices and sensors, and already concerns daily life objects since the birth of the Internet of Things (IoT), introduced in 1999 and officially born between 2008 and 2009 as real interconnection between Internet and things. This spread and diffusion of Internet made the creation of a *cyberspace* (an interdependent network of Information Technology structures) possible [1]. Everybody and everything is daily connected: smart devices, printers, remotely controlled home heating, cars, webcams, control systems of traffic lights and industrial devices .... These devices and their softwares may be subject to *vulnerabilities*, i.e., weaknesses that could be exploited by malicious agents to tamper them or the services they provide, or the activities they are capable of doing. "Your most private secrets" may be no longer private. In other words, *vulnerabilities* can be exploited to sabotage either the hardware or the software of the device . . . or both. It is the case of famous cyber attacks, like Stuxnet that attacked the nuclear power plant in Natanz, Iran [2], and Carbanak, which caused a gain to the hacker group of more than 1 billion dollars, stolen from banks and private customers [3]. Internet is the most commonly attacked environment, with 25 billions intrusion attempts worldwide... daily [1]. The most widely used cyber weapons are botnets on compromised hosts [4]. Information about connected devices, services they can offer and their operating system (if any) is gathered through *Network Scanning (or Probing) Activities* that allow to discover vulnerable hosts and/or vulnerabilities on targeted hosts en masse [5]. Generally speaking, Network Scanning/Probing Activities are "reconnaissance techniques used to determine something about the targeted host". In other and more precise words, they are able to determine open communication ports and available services [6] and "to identify listening ports of the victims systems" [1, 7]. From now on, for sake of clearness, we will only use the expression "Network Scanning Activities" (NSAs). They are one of the most popular techniques to discover and map services that are listening on specific ports or hosts. They allow attackers to create a list of potential weaknesses and vulnerabilities in the open ports, leading to the exploitation and compromise of a remote host [7]. They are highly distributed scanning techniques with high coordination and composite stealth capabilities. These features make all the detection techniques available up to 2014 unfeasible [1]. They are thus nowadays widely used by attackers as a first step of exploitation attempts [8], which aim at taking control over the host [9], exploiting vulnerabilities [5, 8] and gathering information about the system which will be successively the target of a cyber attack. Among attacks that take place through (and thanks to) the cyberspace, one finds DDoS, APTs, Zero-day attacks, ..., each of them with its explosive charge.

- A *DDoS*, or *Distributed Denial of Service*, aims at making a (set of) host(s) or resources of a network unavailable. This is accomplished by directing enormous flood of Internet traffic toward a small set of targeted IP addresses [1], which causes the shut down, temporarily or definitely, of the services they provide.

- *APTs*, or *Advanced Persistent Threats* [1, 10] are attacks performed by a group of attackers with strong capabilities and the persistent and strong intent to efficiently target an entity, which can be a government institution, a company, an enterprise .... After leveraging highly stealthy techniques, the attack is strictuly tailored on the target. It clearly follows that APTs need to be operated by skilled, motivated, well-funded, equipped and organized attackers [1].

- *Zero-day attacks* refer to exploitations of freshly discovered and not yet patched vulnerabilities [1]. Once a vulnerability gets to be known, a vulnerability scan is launched to discover the machine that presents it, ready to be exploited by attackers, or to be patched. An example of vulnerability scanners is Nessus.

Cyber attacks can be deployed also within terroristic activities. In this case, one talks about *Cyber terrorism*. Attacks that fall into this label target governments and/or military facilities corporations, strategical infrastructures like power plants [1]. An example is Stuxnet, deployed during the Russian-Georgian war, which aimed at a sabotage of the Iranian nuclear power plant of Natanz. More precisely, it should have de-activated the spin cycles of the nuclear power plant, impeding the detection of dysfunctions and of Stuxnet itself. Though probably developed since 2005, it was uncover in 2010, and is believed to be a cyber weapon built by government agencies of United States of America and Israeli, respectively. Clearly, neither country has ever openly admitted its responsibility.

The need to prevent, protect from and mitigate attacks and to identify their sources is thus constantly rising [1, 7]. It can be done by *Network Forensics* experts, i.e. by monitoring network traffic in order to determine if the anomalies in traffic indicate an attack. Network Forensics apply forensics science to computer networks in order to discover sources of network crimes. The goal is to identify malicious activities from traffic logs, discover their details and assess the damage [7]. But, network traffic is never constant. Between 1995 and 2004 it increased quite regularly, and slowly started decreasing up to 2006. Within the same interval of time, traffic from Network Scanning Activities had a peak in 2001, and, since there, thousands of NSAs are probing every day [6]. This can be due to the diffuse and incessant usage of Network Scanning Activities ever since by researchers and security experts, but also by malicious agents. Probing traffic has *decreased* since 2004, after the activity of very aggressive scanning worms/bots between 2003 and 2004. This can be explained with the fact that both Network Scanners and Network Scanning Activities have been always refined over time, which let them gain efficiency and a low profile[1].

1.2 Contributions

The reason why within this PhD we decided to study and focus on Network Scanning Activities lies in the fact that in 2015 about 50% of all cyber attacks were preceded by a form of Network

[1] SMTP are scanned with a low-rate constant rate, whereas HTTP for example flare up and later die off.

Scanning Activities [8], which are widespread and regularly captured by monitoring infrastructures, but rarely studied [4]. In order to ensure efficient defense, targeted and advanced threats need to be mitigated at the earliest stage, which means during the Network Scanning Activities [9]. In order to hide their identity, and/or to gain scanning speed and stealthiness, Network Scanners, like Wireshark, NMap, ZMap, Masscan, Shodan ... are often used by attackers which aim at performing a Network Scanning Activity. It is for these reasons that our work aimed at modeling them from traffic logs collected by the hosted network. Inferring them helps security experts and network managers to be aware that a Network Scanner has been launched and that it is probing, or has recently probed, the hosted network. This helps them to prevent cyber attacks from being successful, and vulnerabilities from being exploited. Nonetheless, only few organizations detect Network Scanning Activities and block their probes. When it does happen, it is often by chance, since they are discovered during other maintenance operations [5]. Only limited work exists to detect distributed Network Scanning Activities, and statistical methods are the least exploited ones [1].

The work of this PhD aimed at providing an approach to recognize what Network Scanner is leveraged to probe the hosted network. In other words, once it is stated that a Network Scanning Activity has terminated or is probing the observed network, we are capable to identify what Network Scanner has been deployed. For this reason, since the only available information is the one that could be collected from the observed (and probed) network, the first stage of our work was to learn models of Network Scanners from the point of view of the probed network. This choice is motivated by the purpose of providing models and tools that can be almost straightaway leveraged by security experts, and that require only knowledge of information easily collected from logs of packets received by the probed network. More in detail, only one feature at once of Network Scanners has been modeled: intensity, spatial and temporal movements. *Intensity* refers to the number of scanning probes received by the targeted network within equally long time windows. *Spatial movements* refer to differences between IP addresses of the probed and hosted network that are probed one after the other. *Temporal movements* refer to differences of timestamps of consecutive probes collected by the hosted, and targeted, network. Despite the actual literature is rich in how to detect a Network Scanning Activity, it is poor in providing methods and tools that recognize the Network Scanner leveraged to probe the hosted network. We aimed at filling this gap, by providing a classification technique that leverages the obtained statistical models to recognize what Network Scanner originated the collected probes.

1.3 Manuscript organization

The manuscript is structured as follows. Chapter 2 provides an overview about *Network Scanning Activity* and details taxonomies and descriptions of various types of scanning. It concludes defining the term *Network Scanner* and detailing some of them (Nmap, Shodan, ZMap, Masscan).

Chapter 3 focuses on detection of Network Scanning Activities in the actual state ofthe art, leveraging various techniques. It claims that the actual state of the art is poor in Network Scanner recognition, providing the main motivation of the work done within this PhD.

Chapter 4 gives a detailed description of usage of statistical tools used in the methodologies presented in the following chapters to model Network Scanners and to recognize them.

Chapters 5 and 6 present two methodologies to model two NSs from the point of view of their target. The importance to have these statistical models from the point of view of the targeted system lies in the fact that they:

- leverage only data collected by the target itself. They are thus models that are easily customized to one's own system, being thus customized without requiring a customization of the methodology or its variations.

- do not need any knowledge of the way of functioning or peculiar behaviors of the NSAs or NSs. The proposed approach is then immediately applicable to all of them.

Chapter 6 goes a bit farther, exploiting the obtained models to detect the modeled NSs in real traces of traffic collected from a network telescope hostet at the High Security Lab at Inria Nancy - Grand Est, with a precision that reaches the 100%.

Conclusions follows and terminates the work.

2

Introduction to Network Scanning Activities

Contents

2.1 Introduction

The objective of this chapter is to provide a general overview of Network Scanning Activities (NSAs). It describes who are their performers and targets, distinguishes them between defensive and offensive, details their features, characterizations and types in Section 2.2. Section 2.3 goes a bit more into depth, detailing what protocols can be used by Network Scanning Activities. Considerations about distinctions between normal and probing traffic, the benign or malign nature of a Network Scanning Activity and about their detections are faced in Section 2.4. Section 2.5 proceeds defining the notion of Network Scanner and gives an overview on the most used ones. Conclusions of the chapter ad links with Chapter 3 are in Section 2.6, which concludes the chapter.

2.2 Overview and Taxonomies of Network Scanning Activities

The first thing to define is the concept of Network Scanning Activities. Throughout our work, we adopted the definition provided by Panjwani et al. [6], that we recall.

Definition 2.1. A *Network Scanning Activity* is a "reconnaissance technique able to determine open ports and services" [6].

There are various basic characteristics of Network Scanning Activities that help to have a deeper insight on them and to understand their intrinsic differences. Approaches adopted by Network Scanning Activities have been classified into various ways, according to their movements in the IPv4 space [11], their number of sources and targets, their stealth capabilities [12], etc. There exist various taxonomies (i.e., "hierarchical classifications of entities within a specific domain" [12]), which classify Network Scanning Activities into mutually exclusive categories. The following sub-sections define and present some of them, and show how they group Network Scanning Activities into disjoint classes.

2.2.1 Targets and Performers of Network Scanning Activities

The Definition 2.1 of Network Scanning Activities does not mention anything about their performer nor their targets. NSAs can be performed, for example, by an attacker using a local host to discover local vulnerabilities, i.e. weaknesses within the network that hosts its machine(s). The *performer* of the Network Scanning Activity is *local* if it leverages hosts of the local system or network, or *remote* otherwise. The term "local" here is with respect to a theoretical observer of the Network Scanning Activity. If the performer of the NSA leverages hosts within the network where the observer lies, then it is labeled as "local". It is instead "remote", if it leverages hosts outside the network of the observer. Equally, *targets* can be *local* if they are within the network hosted by the observer, *remote* otherwise. In this latter case, the network hosted by the observer may be included, but it is *not* the only part of the IPv4 space to be probed. The combinations of these definitions regarding the performer and the targets of NSAs produce four disjoint classes of Network Scanning Activities, summarized in Table 2.1 and detailed below [1].

		target	
		local	remote
performer	local	local-to-local	local-to-remote
	remote	remote-to-local	remote-to-remote

Table 2.1: Classification of NSAs with respect to their performers and targets

local-to-local Network Scanning Activities that fall in this category are performed through a host (or more) within the network there the observer is located, host(s) that scans within the boundary of the network in which it resides, together with the observer. Both the targets and the performer of the Network Scanning Activity are "local" with respect to the observer..

local-to-remote A local-to-remote Network Scanning Activity is deployed by a performer that leverages a host or more located within the network of the observer, and that probes outside the boundary of the network that hosts the scanning host(s) and the observer.

remote-to-local The performer of a remote-to-local Network Scanning Activity leverages one or more hosts located outside the boundaries of the network hosted by the observer (i.e. he/she is "remote" with respect to the observer) and probes only the network there the observer is located, which is, thus labeled as "local".

remote-to-remote Remote-to-remote Network Scanning Activities are performed leveraging hosts outside the network of the observer of the NSA, and probe parts of the IPv4 space that does not include the observer's network. Both the performer and the probed targets are "remote" with respect to the observer.

The classes of Network Scanning Activities whose target is labeled as *remote* include, but are not limited to, Internet-wide scanning campaigns, [1], also known as *Distributed cyber scanning campaigns*, *world-wide Network Scanning Activities*, *coordinated Internet-wide Network Scanning Activities* [13]. The target of these Network Scanning Activities is inevitably remote with respect to the observer, because the probed space may include the observer's network, but it is not limited to it. Internet-wide Network Scanning Activities have generally stealth features and slow motion [1]. More precisely, they consist of a large number of Scanning Hosts (bots) that participate at the probing campaign and which are well orchestrated to maximize the coverage, minimize their overlap of targets and redundancy and probe the whole IPv4 space. Internet-wide NSAs are generally performed by propagation of worms, which then launch an attack as soon as an open port is detected on one of the targeted systems [1].

When the target is specific, the Network Scanning Activity is usually *slow* and with a *low profile*. This means that the targeted network is probed slowly, in order to avoid detection. Network Scanning Activities with specific targets can be performed by botnets, i.e. by a set of compromised systems [1].

Let us focus on the *performer* of the Network Scanning Activity. It can leverage a single Scanning Host, meaning that a single source machine is used to probe the targets, or multiple systems or hosts that act in a common strategy to probe the targets of interest. A central node usually collects all the results from all the probing systems or hosts. In the latter case, the probability for the Network Scanning Activity to be detected is lower [1].

2.2.2 Offensive or Defensive Network Scanning Activities

The Definition 2.1 of Network Scanning Activities does have neither a malevolent nor a benevolent significance: in fact, NSAs are leveraged both by attackers and security experts or researchers. They can be thus referred as [14]:

- *offensive* if they are leveraged with malicious purposes, thus by attackers, who perform them before launching an attack or to discover vulnerabilities before they are patched [5];
- *defensive*, if they are deployed to acquire benign knowledge on one's own network or on the Internet. Defensive Network Scanning Activities are performed either by researchers or security experts, with benign or valuable research purposes. They aim at [15]:
  - investigating what weaknesses could have been discovered by Network Scanning Activities that targeted the system;

  - preventing future and undesired Network Scanning Activities;
  - modeling probing attempts.

Defensive Network Scanning Activities have been generally deployed to study cryptographic ecosystems, i.e. services like HTTPS and SSH. ZMap, for example, a Network Scanner developed by the University of Michigan, is continuously launched by researchers of the institution with the goal to probe HTTPS hosts [5].

2.2.3 Boundaries of Targets of Network Scanning Activities

Let us preliminarily recall that communication ports of the TCP/IP slack are generally divided into three intervals:

- $[0, 2^{10} - 1]$: well known/system ports. They are targeted more frequently than ports of other intervals;
- $[2^{10}, 2^{14} * 3 - 1]$: user or registered ports or ports used by vendors for applications;
- $[2^{14} * 3, 2^{16} - 1]$: dynamic/private/ephemeral ports.

The boundary of a Network Scanning Activity can be defined as [1, 11]:

- *horizontal*, when multiple IP addresses are tested on a single port/service;
- *vertical*, when a single IP address is tested on multiple ports/services;
- *block scan*, which is a combination of the two mentioned above.

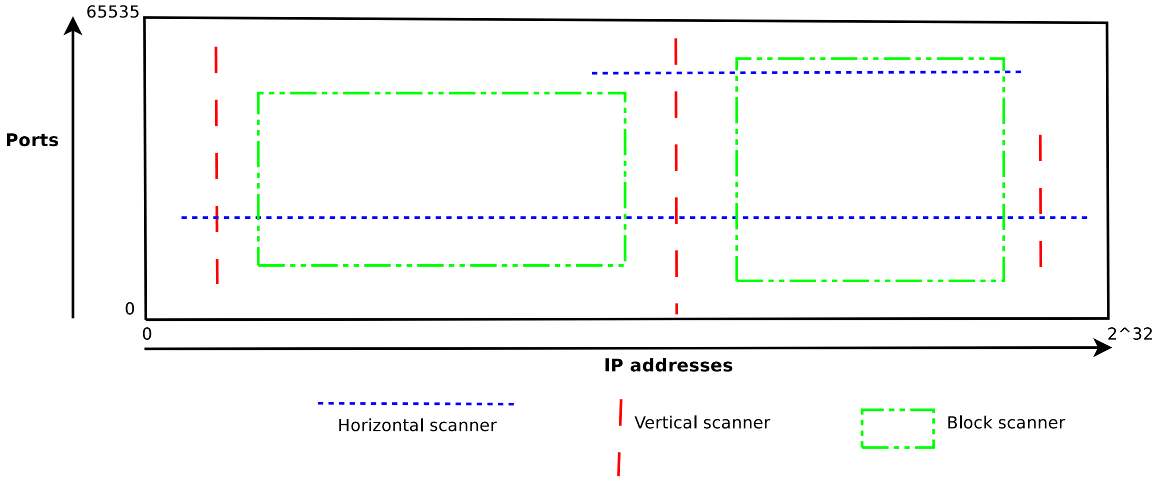


Figure 2.1: Graphical representation of the IPv4 space and of boundaries of NSAs

This classification is motivated by the graphical representation of the IPv4 space as a rectangular grid, where the 2^{32} IP addresses are located in its basis and the 2^{16} ports in its height, see 2.1. A Network Scanning Activity which targets one single IP over multiple ports is here represented as a dashed red vertical line, whereas one which targets multiple IP addresses over a single port by a dotted blue horizontal line. A combination of the two is thus a block delimited by dotted and dashed green lines.

2.2.4 Features of Network Scanning Activities

The characteristics or definitions provided in the previous sub-sections describe specific features of Network Scanning Activities, which are mutually disjointed. NSAs can also be profiled by evaluating other features, not necessarily disjointed, like *services or ports they probe*, their *scanning patterns* and the *Network Scanner they leverage* [14].

Probed Services

Not all the services are probed with equal manner, intensity or speed. Some of them are constantly probed, like FTP (port 21), SSH (port 22), HTTP (port 80), HTTPS (port 443), SMB (port 445) [14], SMTP (port 587) [16], MS SQL Server (port 1433) and HTTP alternative (port 8080) [14]. SSH (port 22) was, up to 2014, the most probed service by large Network Scanning Activities, but only the seventh by small NSAs [5]. Other ports are instead probed with an interest that quickly arises, and then slowly decays. This could be possibly explained with a discovery of some vulnerabilities, which motivates the sudden interest to the service, followed by some disinfection activities or patches to the vulnerability itself, which causes a slow decrease of interest to the service [14]. An exception to this behavior has been observed in March 2014, when a sudden surge in probing Telnet (port 23) took place and targeted Telnet-enabled Internet-of-Things (IoT) devices, as cameras. The exception lies in the fact that the surge has not been followed by any decrease. This could be motivate by a lack of security updates on IoT devices, and with the fact that new vulnerable devices are constantly connected to the Internet [14].

Scanning Patterns

Network Scanning Activities are labeled as *random* if they spread the probe load uniformly across the Internet or the range of targeted IP addresses. In this case, detection inside subnets of the IPv4 space, or of the range of probed IP addresses, is difficult, since only a small number of probes are received by the sub-network in a limited time window. Thus, randomization of probes is a powerful mean in the hands of malicious entities, and its usage increased over time. If, back to November 2006, only 10/15% of Network Scanning Activities were random, in 2013 their percentage reached 57% [14].

Network Scanners

Network Scanning Activities can deploy one or more *Network Scanners*, also known as mass-scanning tools. They send packets to numerous destinations and analyze the corresponding answers, or lack thereof, to acquire knowledge on remote hosts or networks [14], and can deploy multiple *Scanning Hosts*.

2.2.5 Active and Passive Network Scanning Activities

Network Scanning Activities can be *active* or *passive*.

Active Scanning *Active Network Scanning Activities* identify network services in a proactive manner: they transmit probes toward hosts or devices, and monitor their responses. The probes that are sent can be *generic* or can target a *specific protocol or application*. Active scanning is often used to look for *vulnerabilities* [1].

The main advantage of this type of scanning is its probing speed [1]. Among the disadvantages, one finds that [1]:

- it is intrusive, since it asks for a response;
- the request of a response makes it easy to be detected;
- ports that are filtered by a firewall are not probed.

Passive Scanning *Passive Network Scanning Activities* scan network services by observing traffic generated by and between servers and clients, when it passes through an observation point. A widely used tool to accomplish this task is Wireshark. Passive Scanning is able to detect both TCP and UDP well-known services. More precisely, it only captures setup messages of TCP connections, and observes UDP traffic. Recall that UDP is a connectionless protocol. This means that:

- bidirectional traffic between host and server indicates the presence of a UDP service;
- unidirectional traffic *may* indicate:
  - a service, since the UDP protocol does not require any response;
  - undesired probe traffic.

Services that do not run on well-known ports cannot be identified [1], if the header and the targeted port are the only information available.

Passive Scanning has two *disadvantages*:

- only active services are detected;
- only packets passing through observation points are seen. Observation devices need thus to be wisely located by performers of Passive Scanning, in order to maximize the amount of the collected information.

Disadvantages are compensated with many *advantages*, since Passive Scanning [1]:

- is *not* intrusive;
- can by-pass firewall configurations;
- does not consume network resources;
- detects more efficiently services that run on transient hosts;
- is harder to be detected.

An example of Passive Scanning is provided by PADS: Passive Asset Detections System, which provides an accurate and current list of hosts and services offered on the network of interest [1].

2.2.6 Other characterizations of Network Scanning Activities

Network Scanning Activities can be defined as *complete*, if they scan all the (targeted subset of the) IPv4 space, and *partial*, otherwise [17]. They are defined as *good* if they achieve maximum politeness at remote networks and efficiently use resources in case the scan is complete, or extrapolate information with accuracy in case the scan is partial [17].

Network Scanning Activities can also be *coordinated* over a set of scanning networks, when various sources work in a synchronized manner [15, 18] to gain stealth, firepower and more data: more exploits are obtained in smaller time windows, multiple Scanning Hosts are leveraged, each scanning a small portion of the targeted network [18], a blockage of an IP address or a subnet is less effective, and it is possible to obtain data which is impossible to collect with a single source

IP scan. Although coordinated Network Scanning Activities may be very different one from each other, they all consist of two phases [18]:

- permutation and split of the targeted set of IP addresses, where:
    - *permutation* establishes the order in which IP addresses are scanned;
    - *split* phase assigns targeted IP addresses to several hosts. The rationale behind this lies in the fact that a source IP address sending packets faster than others is readily detected.
- schedule of the scanning activity.

In other words, permutation-scanning worms have a divide-and-conquer approach [19]: each host scans a subset of all addresses, which are permuted because a sequential scanning increases the risk of identification. These two phases are clearly visible in the algorithm of ZMap, see Section 5.3.

Network Scanning Activities can also be *stealthy*, when they adopt techniques to avoid detection. When considering their *probing speed*, NSAs can be defined as *Rapid*, *Medium* and *Slow* [12]. When considering their *scanning distribution*, instead, their classification is built "crossing" the number of sources performing the scanner, one or many, and the number of hosts targeted, one or many.

2.2.7 Types of Network Scanning Activities

According to the approaches they deploy, or the goal they want to achieve, there are various types of Network Scanning Activities, some of which listed below.

- *Decoy Scanning* [15] consists in sending several probes to the same target, all but one are sent with spoofed addresses. The source of this Network Scanning Activity is hard to detect because, even if probes could be detected, it is impossible to know what the IP address of the real performer of the Network Scanning Activity is. Side effects of this are that hosts used as decoys need to be reachable in order to keep hidden the identity of the source of the Scanning, and involuntary DoS attacks against the Target may result.
- *Proxy Scanning* [15] is a real security threat since it is hard to trace. Let us assume that S is the *Scanning host*, T the *target machine* and F the *ftp server* with the proxy option enabled and capable to establish connections with both S and T, S starts a ftp session with F, declares a selected port (p-T) on T as a passive port needed for a proxy transfer, and then starts a proxy data transfer to p-T. If p-T is listening, then the transfer is successful. If p-T is closed, then a "425 Can't open data connection" reply is obtained.
- *Security Scanning* [15] automatically detects security weaknesses on remote or local hosts, querying connection ports and recording responses. It also determines whether anonymous logins are supported, whether certain network services require authentication, and whether security holes are present.
- *Port Scanning* [15, 11] checks for open and/or closed ports and for used and/or unused services which may have vulnerabilities to be exploited. The protocol and port of the target can be *one* or *many*.
- *Vulnerability Scanming* [6] fingerprints the presence or absence of exploitable vulnerabilities, that differ one from each other. This implies that also techniques to fingerprint them differ, which makes the development of an algorithm to detect them difficult.

2.3 Protocol-based and Active Network Scanning Activities

An important classification of active Network Scanning Activities is based on the protocol they adopt [12]. There exist thus TCP/IP, UDP and ICMP Network Scanning Activities [15]. In this section, we focus mainly on TCP scanning activities, widely used by major Internet-wide Network Scanners like ZMap [20] or Shodan.

In order to better understand how Network Scanning Activities work, it is important to have a clear overview on how TCP connections are established [15]. A *connection* is *established* between two *sockets*, one on the sending machine and one on the receiving one. A *socket* is simply a pair (IP_address; port_number). Connections are thus identified by the pair (socket_sender, socket_receiver), i.e. by their two emphendpoints, that exchange *segments*, each composed by a *header* and optional data. A *header* includes six flag bits:

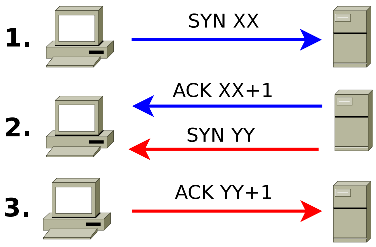


Figure 2.2: TCP Three-way handshake

SYN: initiates connection. If this flag is set, then the *Synchronize Sequence Number* is the first sent octet;

FIN: indicates that the sender finished sending data;

RST: resets the connection;

URG: states that urgent pointer is valid;

ACK: states that the *Acknowledgement Number* is valid;

PSH: indicates that the receiver should pass this data to the application as soon as possible.

A TCP connection needs to be established *before* two applications using TCP can exchange data. The TCP connection happens as follows. Two streams of data are present between the two sockets, and to each of them an *Initial Sequence Number* (ISN) is assigned. The *three-way handshake*, represented in Figure 2.2, consists of three steps, detailed below and visible in the aforementioned figure.

1. The *host* sends a *SYN request* to the *server*, with the SYN flag set to ON. The SYN request specifies:
   - the port number of the server that the host needs to connect to;
   - client's ISN (Initial Sequence Number), denoted by XX in Figure 2.2.
2. The *server* responds:
   - acknowledging the host's SYN by specifying that the next byte that the server expects is the one numbered XX+1;
   - with a SYN message with its ISN, labeled with YY in Figure 2.2;
3. the *host* acknowledges the SYN messages from the server, with the ISN equal to YY+1.

Only now data transfer may take place. Some implementation details are shown in Table 2.2

Port	Segment received	Action
closed	SYN	Segment dropped + RST sent
	FIN	Segment dropped + RST sent
	RST	Segment dropped
listening	RST	Segment dropped
	ACK	Segment dropped + RST sent
	SYN bit off	Segment dropped
	SYN	SYN/ACK
	FIN	Segment dropped

Table 2.2: Segments received according to the status of ports, and following actions

2.3.1 TCP based Network Scanning

TCP/IP based Network Scanning Activities exploit the TCP/IP stack and its TCP protocol to determine what TCP ports of one or more hosts have processes listening on them for possible connections [15]. At the end of 2016 they were the 56% of all the performed Network Scanning Activities [14]. Since the TCP/IP stack is dependent on the Operating System, their variations can be exploited by the attacker to fingerprint the Operating System of the target [6]. Some of the NSAs in this category are listed below.

- *Full TCP Connection Scanning* [15], also known as *Full-Open Scanning* [6] and *Vanilla Scanning* [1][2] tries to establish regular (full) connections, i.e. the three-way handshake as described above and represented in Figure 2.2, with chosen ports on the targeted machines [15, 1, 6]. It only verifies the availability to open a TCP connection, and not what services are supported by the connection itself [1]. If the connection attempt fails, then the port is closed [7] and a RST flag is received [1]. If the port is listening (open), instead, an ACK flag is received [1]. This type of scanning does not classify services that have *no* standard ports, or those services that dynamically assign ports [1]. Full TCP Connection Scanning is easy to detect, since logs of the probed networks report rapid sequences of connections and errors, which thus make it visible in system logs. It is a slow Network Scanning Activity [7], and implemented in NMap.

- *TCP SYN Scanning*, or *half-open Scanning*[3] [15, 1], sends a SYN segment to the chosen port on the targeted machine. The received response is analyzed without completing the TCP handshake process that is reset before its completion [1]:

  – if the received answer is an RST flag, then the probed port is *closed* and another one is scanned [15, 1, 7];

  – if the answer is a SYN/ACK flag, then the port is *listening* or *open*: a RST flag is then sent back [15, 1, 7].

  Traffic logs registered by probed networks show a large number of SYNs and RSTs. This type of Network Scanning Activities is able to probe thousands of ports per second on a fast network without restrictive firewalls, and owes its name to the fact that the TCP connection is not completed [7]. Half-open Scanning is more difficult to detect, since it is not logged by destination applications [1]. It is furthermore less stressful for the targeted

[2] Implemented in NMap

[3] This is the technique used by ZMap

network [1] and relatively stealthy. Up to 1999, this type of scanning was less applied than Full TCP Connection Scanning [15]. After 10 years, it turned out to be the mostly used method, thanks to its simplicity and performances [7], despite its disadvantages: it requires elevated privileges at the source (i.e. of the Scanning Host) and is easily detected by firewalls [1]. TCP SYN Scanning is implemented in ZMap [20].

Half open scanning was leveraged and deployed by *Version Detection Scanning*, that exploits open ports to probe software applications that run on remote services. If open ports are found, the *Version Detection Scanning* communicates with remote applications on open ports to uncover information like type, version, and status of the service, the Operating System and its version. Despite it is a revealing tool, it needs a significant computing power and an elevated monitoring bandwidth, and appears in application logs [1], which make it easy to detect.

- *Stealth scanning based on the TCP/IP protocol* [15] tries to avoid detection and filtering devices. It accomplishes it by employing sets of flags other than the SYN, in order to appear as legitimate traffic. The following are *Stealth Scanning* approaches.

  - *SYN/ACK Scanning* [1] is a modification of the *Half-open Scanning*, since it employs sets of flags that are different than the SYN required by the TCP/IP handshake (see Figure 2.2), in order to appear as legitimate traffic [1]. More precisely, the Scanning Host sends SYN and ACK flags to the target. If the probed port is *closed*, then the probed target replies with a RST flag. If the probed port is instead *open*, then the target sends no response, because the TCP protocol requires a sole SYN flag to initiate a connection [1]. This scanning is fast, but generates a high number of false positives [1].

  - *FIN Scanning* [1, 7], *XMasTree* and *Null Scanning* [1] operate as SYN/ACK scans, but send different flags to probe ports: FIN, URG and PUSH; and empty flags respectively. These Scanning leveraged the mentioned flags segments to scan ports, according to the implementation details summarized in Table 2.2. If the probed port is:

    - closed, then it sends a RST as reply;
    - open, then the FIN is ignored.

    Thus, if the probed port is *close*, only two packets are transfered. If the probed port is instead *open*, then only a single packet is transfered [1]. Since these segments may pass through packet filters only looking for SYN flags, they leave no traces in application logs. This makes these scanners very difficult to log. Traffic logs collected by the probed network show a large number of FINs, URGs, PUSHs and no flags without corresponding SYNs and ACKs [7]. Despite they fail with Windows95/NT and in general with Microsoft machines because their ports appear *closed* regardless of their real state, they succeed with Unix and other targets. Any device that shows open ports is *not* a Microsoft machine. [1].

  - *Idle Scanning* [1] or *Indirect Scanning* [15] gathers informations about ports by leveraging the so-called *zombie*, which is a spoofed IP address of a third host to disguise the real Scanning Host. The scanning process appears as it is performed by the IP address of the *zombie*, despite it is really performed by the Scanning Host. Three actors are thus involved: a *Target*, a *Scanning Host* and a *Silent* or *Zombie* machine. The latter needs to:

  - be idle, i.e. must not send any packet while the Scanning Host probes the Target, so that the IP identification frames remain consistent throughout the duration of the scanning process;
  - provide consistent and predictable IP identification values (IP-id).

  Indirect, or *Idle Scanning*, works as follows. The *Scanning Host* sends a SYN/ACK flag to the *Zombie* (*Silent*), and receives a RST flag as a response, which contains also the initial IP id. Then, the *Scanning Host* probes the *Target* host using the spoofed IP-id values of the *Zombie.* If the probed port of the *Target* is open, then the *Target* replies with a SYN/ACK flag to the *Zombie* host. Since it does not expect any SYN/ACK flag, it replies to the *Target* with a RST flag, causing an increment of 1 of its IP-id counter If, instead the probed port of the *Target* is *closed*, then it sends an RST flag to the *Zombie*, which drops it. Then, the *Scanning Host* resends the initial SYN/ACK probe to the *Zombie*: if its IP-id has been incremented, then the probed port on *Target* was open [15]. If its IP-id remained unchanged, then the probed port on the *Target* was closed [1].

  The main advantage of this scanning approach is that the IP address of the *Scanning Host* is never revealed to the Target. But, on the other hand, it:

  - requires a *Zombie* host;
  - does not prevent the *Scanning Host* from being identified if *Indirect Scanning* is used in a local subnet, because the *Scanning Host* MAC address on the local subnet is not spoofed;
  - requires privileges to create new packets.

- *ACK Scanning* [1, 7] finds out what ports are filtered by a firewall, since it does not distinguish between closed and open ports, but between *filtered* and *not filtered* ports. This, because it never connects to an application to confirm the state (open or closed) of a port [1]. Unfiltered ports appear as *not-reachable*, and filtered ones as *reachable*, and not between closed or open ports. This type of Scanning sends a TCP ACK flag to a remote port, and analyzes its response.

  - If an *ICMP destination unreachable message* or *no response* is returned, then the port is *filtered.*
  - If an RST packet is returned, then the port/service is not filtered.

  This type of scanning does not open any connection, nor any session. This means that the "conversation" between the Scanning Host and the remote target is simple, which makes the probe of one port almost invisible, and the ACK scanning difficult to detect [1]. Traffic logs of the probed system show a large number of ACKs without corresponding SYNs [7].

  The main disadvantage of ACK Scanning is that it cannot identify in a definite manner an open or closed port, because it never tries to connect to a remote device. A list of filtered and unfiltered ports is anyway useful as a reconnaissance method for Network Scanning Activities that focus on specific port numbers and successively on their vulnerabilities [1].

- *Window Scanning* leverages the fact that some TCP stacks return a predefined and known window size number when replying to an ACK packet. If a closed port is probed, then it replies with an RST flag and window size 0. Open ports reply with an RST flag and a non-zero window size. The main advantage of a Window Scanning is its absence in logs of the probed system, since it does not open any session. On the other hand, it does not work on all devices, and the number of devices on which it works is decreasing [1].

  - *Fragmented Packets Scanning* [15] or *TCP Fragmentation Scanning* [1] splits TCP packet headers up into several and smaller packets to avoid filters, making its detection by Intrusion Detection Systems (IDSs) harder. The disadvantage of this Scanning approach is that some destinations do not merge packets in a proper manner, causing the failure of the scanning itself [1].

- *Ident scanning*, also known as *reverse ident scanning* [15] retrieves the user-name (Userid) that owns the daemon (process) running on a specific port by attempting to connect to a TCP port. If full connection is established, then the ident request is sent out to the identd (Identification Protocol daemon) on TCP port 113 of the targeted host.

- Differences among TCP/IP implementations between Operating Systems (OSs) can be exploited by *Operating Systems detectors* [15] to detect the Operating System on the targeted host. It is possible, because the network stack of each Operating System, that is, the software that implements the TCP/IP protocol, varies the way it responds to some packets, and these variations depend on the Operating System itself [1]. *Operating system detectors* send packets to analyze responses, which have a signature of the remote Operating system. The signature is then compared against a database of signatures of known versions of Operating Systems [1]. Some examples are:

  - QueSO, the first OS detector to use a separate database for fingerprints;
  - NMap, that added OS detection techniques and defined a template structure to describe fingerprints;
  - Conner and Lin that do active probing: the TCP/IP implementation is considered as a black box, and its characteristics are deduced by studying how TCP responds to probes.

2.3.2 ICMP based Scanning

ICMP (or Ping) [15] or *Sweep Scanning* [1] uses information provided by ICMP control messages to check availability of a target machine [15] to identify as many active hosts as possible [1], and fingerprint the Operating System (OS). A single ping tells the sender whether one specific host computer exists on the network [15] and provides less information than UDP and TCP Scanning [6]. ICMP Scanning is accomplished using either *real pings*, i.e. sending ICMP echo requests and marking as *up* every responding host, or *TCP pings. Ping Scanning* is one of the oldest and slowest methods to scan a network. Some existing tools are [1]:

- fping, gping and NMap for UNIX systems;

- Pinger and PingSweep for Windows systems.

. The following Scanning approaches fall into this category.

- *ICMP (Internet Control Message Protocol) Echo Request Scanning* consists of ICMP (Internet Control Message Protocol) ECHO requests sent to multiple targeted hosts [15] and waits for their replies.

  - If the target replies with an *ICMP ECHO reply message*, then the address is alive, i.e. the target is active [1, 15].
  - If the Scanning Host receives a *destination unreachable reply*, or no answer is received within the *request time* or the target dropped the probing packet, i.e. it discarded the request, then the target is not active [1]. The main advantage of this approach is its independence from applications or open ports. On the other hand, it is the protocol that is the most filtered by enterprise networks.

- *ICMP Timestamp and Address Mask Scanning* [1] sends one of the *Timestamp* and *Address Mask ICMP Messages*, which are seldom used, and wait for responses. The drawback of this approach is that these messages are rare to appear, thus easily detected by the target. If the hardwares or the Operating System of the targets are updated, then the results of this scanning approach are not promising.

2.3.3 Miscellaneous Scanning

Scanning approaches that do not fall into any of the previous categories follow.

- *FTP Bounce Scanning* employs the FTP server as a proxy between the Scanning Host and the Target, and is effective against firewalls. It works as follows.
  - The Scanning Host connects to the FTP server;
  - the Scanning Host transmits a port command with the IP address and the port to probe;
  - the FTP server forwards the received request to the target, and replies to the Scanning Host with:
    - *no connection built* if the probed port on the target is closed;
    - *connection complete* if the probed port on the target is open.

  The main advantages of this scanning approach are its stealth capabilities, and the fact that the standard FTP communication is used by the majority of enterprises. On the other hand, it is successful only with TCP ports; it is slow; it can probe only one port at a time and it is easy to detect because the connection appears on logs of the target.

- *UDP Scanning* is used to probe some specific services which respond with success to a UDP probe packet [1]. More precisely, it sends 0 bytes UDP packets to selected ports on targeted machines and waits for possible replies. If *no response* is received, then the *port* is *open* [15], and the UDP service is present [1]. If the response is an *ICMP port unreachable error message*, then the *port* is *closed* [15], which means that no process is listening to the probed UDP port [1]. Its effectiveness is empowered by the absence of handshaking flags in packets [1]. Its drawback is the time to wait for a response.

- *IP protocol Scanning* [1] aims at identifying additional IP protocols used by the target. An unavailable IP protocol does not respond, whereas an available IP protocol provides a response specific to the protocol type. This scanning is easy to detect because protocols different than TCP and UDP are rarely used, thus easily individuated [1].

- *Remote Procedure Call (RPC) Scanning* sends RCP null messages to open ports that have been previously detected. If any RCP application runs on the target, its replies contain information like the name of the application, its version and status. This scanning approach establishes application sessions. The transition events thus appear in logs of the target, making thus this probing approach easily detected. The drawback of this scanning is that it relies on previously detected open ports.

2.4 Detection of Network Scanning Activities

Up to 2005 Network Scanning Activities and attacks were not strictly correlated [6]. Nowadays, instead, NSAs might be precursors of attacks, despite it is not always the case. This makes sense, since Network Scanning Activities are performed not only by *attackers*, but also by *researchers*, who perform them for security and investigation purposes. Some of the questions that arise, in an increasing order of abstraction, are:

1. How to know if behind a Network Scanning Activity there are attackers or researchers and security experts?
2. Are Network Scanning Activities correlated to attacks?
3. How to distinguish probing traffic between normal one?'
4. How to distinguish between types of Network Scanning Activities?
5. How to detect a "Scanning Host"?

In the following sub-sections we provide answers to them.

2.4.1 How to determine the performer of a Network Scanning Activity

This sub-section provides an answer to the first question: How to distinguish a Network Scanning Activity performed by an attacker from one performed by researchers or security experts? Network Scanning Activities performed by researchers are a really small percentage of the total. More precisely, in 2016 only 18 entities that perform them and make it publicly known have been documented [14], and the majority of them are security research entities. The percentage of Network Scanning Activities performed by them was 0.44% of the total, of which 22% of them used ZMap and 32% used Masscan. The percentage of IP addresses reserved for Network Scanning Activities executed by these entities was 0.1% of all the *Scanning Hosts*, i.e. the IP addresses leveraged for NSAs. Within them, 18% deployed ZMap, and 3.7% Masscan. Furthermore, it is possible to distinguish Network Scanning Activities performed by researchers or security experts for two reasons:

- their behavior profiles differ from the ones of the NSAs performed by attackers;
- they differ with respect to the probed services.

Despite Network Scanning Activities performed by benevolent entities should follow online documentation best practices, only 39% of them do [14]. "Online documentation best practices" include:

- stating the benign nature of probing traffic;
- stating the purpose of the probing campaign;
- divulging contacts of the entity performing the scanning campaign;
- providing the possibility to be excluded from further probing campaigns.

2.4.2 Correlation between Network Scanning Activities and Attacks

Let us face the second question of the above list: “Are Network Scanning Activities correlated to attacks?”. It can be split into two sub questions [6]:

“Are attacks preceded by Network Scanning Activities?” Despite security community frequently states that Network Scanning Activities are precursors to attacks, only few studies have been conducted to validate this hypothesis [6]. Up to 2005, only few of the detected Network Scanning Activities were followed by attacks. More precisely, none of the ICMP Scanning was followed by an attack, and only 4% of Port Scanning andd 21% of Vulnerability Scanning were followed by an attack [6]. But, if one focuses on combinations of Network Scanning Activities, then the situation changes. Combinations of ICMP and vulnerability Scanning, in 2005, were followed by an attack only in 3% of cases. If one adds also a Port Scanning, then an attack follows in 46% of cases. The combination of Port and Vulnerability Scanning turns out to be a good indicator or predictor of a future attack, since it follows 71% of the aforementioned combinations [6].

“Are Network Scanning Activities followed by attacks?” The answer to this question was, until 2006, negative in more than one case over 2: more than 50% of attacks were *not* preceded by any NSA. Of the rest, only 38% of them were preceded, in 2005, by vulnerability scanning, and only a small percentage, between 3 and 6%, was preceded by the combination of vulnerability and port scanning [6]. With time, the situation changed: in 2014, 50% of cyber attacks were preceded by a Network Scanning Activity [1]. Despite this, progresses in improving cyber security of network were limited [1].

2.4.3 Identification of types of Network Scanning Activities and Attacks

Let us now answer the third and the fourth questions of the above list: “How to distinguish probing traffic between normal one?” and “How to distinguish between types of Network Scanning Activities?”.

In order to properly distinguish between probing traffic and normal one, let us preliminarily define the *final state* of the connection between two sockets to be:

- *good*, if the connection is successful;
- *bad*, if the connection attempt is rejected;
- *unknown*, if it is impossible to label the final state of the connection as good or bad. 1% of connections are, in average, labeled as unknown [16].

Connection attempts that do not result in established connections, or in other words whose final state is *bad*, are possibly probes [16]. This helps to distinguish between probing and normal traffic, answering the third question of the list.

The distinction between types of Network Scanning Activities ans attacks is related to the number of packets per connection. If it is less than 5, then a *port Scanning Activity* is faced. More precisely, the number of packets per connection established for port scanning is 2 in 99.76% of cases; less or equal to 3 in 99.83% of cases and less or equal to 4 in 99.88% of cases. This explains that the threshold of 5 packets per connection can be used to characterize port scans. This assumption is confirmed also by the fact that the full TCP three-way handshake consists of three packets, with an optional fourth as a reset packet [6]. If the number of packets per connection is instead equal to 6, then three *half reverse port Scanning Activities* are faced. If it is between 5 and 12, external values included, then a *vulnerability Scanning Activity* is faced. These values are motivated by the fact that less than 0.05% of vulnerability scans have four or less packets per connection. But, many connections of six packets are half reversed port scans performed three times. 99.9% of vulnerability scans have their TCP connection built using between 5 and 12 packets each (besides the 6 packets per connection, as seen above), and only 0.03% have more than 12 packets per connection. Recall that the case in which one has 6 packets per connection, one faces three half reverse port Scanning Activities.

If more than 12 packets are sent per connection, then an attack is faced [6].

2.4.4 How to identify a "Scanning Host"?

Before answering the fifth and last question of the above list, the following definition is needed.

Definition 2.2. A *Scanning Host* is a machine performing a probing activity. More precisely, it is defined to be a service that sniffs around to look for open ports by sending *probes*, i.e. connection attempts, and analyzing their final state [16].

Yet, this definition remains abstract, and needs to be quantified and clarified. To do this, one looks at the number of pairs (target_IP, port) that a remote host attempts to access. If this number is small, then the host targets few services, or few (local) hosts on the same service, in both cases, the host can be labeled as *good*. Vice versa, if the aforementioned number is high, then the host probes a large number of ports, or the same port but on multiple hosts. In both cases, it can be identified as a *bad host*, or, using other words, as a *Scanning*, or *Probing*, *Host*. We are now able to provide a more detailed definition, that follows.

Definition 2.3. A *Scanning Host* can be defined as *a remote host that has bad service fanout of at least 4 and has bad service fanout of at least 2 times the good service fanout* [6].

Note that its connection attempts fail more than the ones of legitimate hosts [16][4]. A Scanning Host can be also defined as follows.

Definition 2.4. A *Scanning Host* can be also defined as "a source IP address that scans at least a fixed number of unique IP addresses of the hosted network on the same port and protocol at a minimum estimate Internet-wide probing rate of 10 packets per second" [5].

2.5 Network Scanners

The last thing that is left to clarify is the concept of *Network Scanner*. This section aims at answering two last questions:

1. What is a *Network Scanner*?
2. What are the most used Network Scanners?

Let us proceed with order. The first question is answered by the following definition.

Definition 2.5. A *Network Scanner* is a mass-scanning tool, that sends packets to various destinations and analyzes the corresponding responses, or lack thereof. Its goal is to acquire knowledge on remote hosts or networks and/or ports [14]. Multiple Scanning Hosts can be deployed by a Network Scanner.

[4] this could be leveraged to identify scanning hosts

Recall that a Scanning Host is a machine performing a probing activity (see Definitions 2.2, 2.3 and 2.4).

Let us now move to the second question. A widely known Network Scanner is Nmap, released in September 1999 and originally written by Gordon Lyon (also known as Fyodor Vorskovich). Its name is the short form for "Network Mapper", since it aims at discovering hosts and services on a network and building a map of them. Nmap is able to discover listening ports leveraging Full TCP Scanning [21], and to detect the Operation Systems of computers. Its recent versions include also NSE scripts, which make Nmap able to perform vulnerability checks against discovered services. It is thanks to it that port scanning, its importance and its links with vulnerabilities become known to the great public, thanks to its appearances in movies like The Matrix Reloaded (2003), The Listening (2006), Bourne Utimatum (2007) ... [22].

Other Network Scanners that are nowadays widely deployed are ZMap [20], released in August 2013, and Masscan [14], released in September 2013. Both use TCP, UDP and ICMP protocols, performing thus a wide variety of probing. ZMap is more prevalent than Masscan. They implement specific packet fingerprinting in the ID field of the IP header. This implies easy identification: if 95% of packets of an event matches the fingerprint of a Network Scanner, then the probing campaign (or, equivalently the Network Scanning Activity) is considered as being performed with the considered Network Scanner. Both of them, as Nmap [21, 23] are open source. This means that malicious entities can adopt and change their codes, to remove, for example, the fingerprinting mechanism or to make other changes [14]. Worm outbreaks are capable to probe and infect 90% of all vulnerable cyber hosts in less than 10 minutes [1].

Another Network Scanner that deserves attention is Shodan [24], launched in 2009 by John Matherly, that, leveraging several Scanning Hosts [25], aims at searching devices linked to the Internet [26] and owes its name to a malevolent character of the video game "System Shock II": Sentient Hyper-Optimized Data Access Network, or Shodan, indeed. More in detail, it is a search engine, reachable at `www.shodan.io`, that gives its users info about a wide range of devices connected to the IPv4, like security cameras, home heating systems, control systems of nuclear power plants, particle-accelerating cyclotrons ... [26]. The main issue with all these devices is not that they can be found by Shodan, but that they have little or no security at all: a web browser is all what is required to connect to them, which often have "admin" as their username and 1234 as password, or no password at all. This lack of security, which is totally unmotivated, allows anyone to sniff into bedrooms, homes, to mess up with traffic lights, power plants and nuclear reactors [26]. Shodan focuses on SCADA (Supervisor Control and Data Acquisition) systems, and collects data mostly on HTTP (ports 80, 8080), HTTPS (ports 443, 8443), FTP (port 21), SSH (port 22), Telnet (port 23), SNTP (port 161), IMAP (port 993), SMTP (port 25), SIP (port 5060), Real Time Streaming Protocol, or RTSP (port 554), the latter being exploited to take control over cameras. The website of Shodan gives back:

- 10 results to users who do not have an account;
- 50 results to users who have an account;
- more results to users who provide a reason and pay a fee.

Device data is stored in "Host Profiles", each of them providing:

- device summary, i.e. IP address, location latitude and longitude;
- available services, listen in the rder of most recent scans and port services associated with the service banner [25].

It is mostly used by cyber security professionals, researchers and law enforcement agencies, who have no problems to provide the aforementioned reason to have more than 50 results. Cyber criminals prefer to leverage botnets, which could furnish the same information without detection [27].

2.6 Conclusions

This chapter provided a detailed overview of Network Scanning Activities, in order to give comprehensive explanations about how they work, why they are deployed by security experts and researchers and by attackers and malevolent cyber criminals before moving to the next chapter, which will focus on their detection. The concept of *Network Scanning Activity* was initially defined, followed by some taxonomies which classify them according to a variety of features and characteristics. An analysis of network traffic follows, which shows how normal traffic can be distinguished from probing one. Analysis of transferred packets during a connection, and more precisely of their number, suffices to distinguish between types of Network Scanning Activities and attacks. The definitions of the terms *Scanning Hosts* and *Network Scanners* are then provided at the end of Sections 2.4 and 2.5, respectively, in order to have clarified all the actors involved during a probing campaign before moving to the following chapter. Chapter **??** will focus on detection of Network Scanning Activities, and output that the actual State of the Art is poor in providing tools, techniques and methods to recognize what Network Scanner is leveraged during a probing campaign.

3

Techniques to detect Network Scanning Activities

Contents

3.1 Introduction

Chapter 2 provided a global overview on Network Scanning Activities and Network Scanners 2.1. It also mentioned who performers of Network Scanning Activities are (see Section 2.4.1), how probing traffic may be identified and distinguished from normal one (see Section 2.4.3) and if Network Scanning Activities are correlated to attacks 2.4.2. In literature, multiple tools and techniques have been proposed for the detection of Network Scanning Activities, varying from approaches for detecting independent Scanning Hosts to approaches for detecting coordinated Network Scanning Activities. This chapter aims thus at detailing techniques of detection of Network Scanning Activities and currently available tools and approaches that accomplish this task. More precisely, since Network Scanning Activities can leverage one or multiple Scanning

Hosts as sources (see Section 2.2.6), also detection approaches reflect this bipartition. Only little has been done, so far, to recognize what Network Scanner is leveraged by performers of the detected Network Scanning Activity.

The structure of the chapter follows. Section 3.2 details approaches to detect Network Scanning Activities that leverage a single Scanning Host. Section 3.3 presents detection approaches for NSAs that deploy multiple Scanning Hosts as sources. Criteria to evaluate detection approaches are detailed in Section 3.4, whereas analyses are performed in Section 3.5.

3.2 Detection approaches for single Scanning Hosts

It is since 1990 that Intrusion Detection Systems are equipped with some sorts of approaches to detect Scanning Hosts. Approaches of detection of Scanning Hosts can be categorized into: *Algorithmic* (Section 3.2.1), *Threshold-based* (Section 3.2.2), *Soft computing based* (Section 3.2.3), *Ruled-based* (Section 3.2.4) and *Visual* (Section 3.2.5), all detailed in the remaining of this Section.

3.2.1 Algorithmic Detection Approaches

Approaches of Detection of Scanning Hosts that are defined as *Algorithmic* leverage tests of hypotheses and probabilistic models to detect Scanning Hosts. Only analyses of traffic packets collected by the probed network is required.

The first detection technique to be developed is GrIDS (*Graph-based Intrusion Detection System*) [28], which leverages information at packet level to represent communication patterns observed on the considered network with *graphs*: nodes are hosts and edges represent connections. GrIDS is then a graph-based technique to detect Scanning Hosts able to detect and analyze large scale attacks and attacks in individual hosts. *Activity graphs*, which require a notable amount of time to be built [29], are leveraged to aggregate the activity of the network of interest. The other Achilles' heel of GrIDS is its ineffectiveness against Denial of Service (DoS) attacks [29].

An attempt to improve GrIDS was proposed in 1999 and leverages evaluation of only connection attempts that result in a RST-ACK packet from the probed IP address [30]. This technique, despite it is able to identify hosts that probe more than 4 IP addresses that respond with RST-ACK packets, if they are active [30], is not able to detect non TCP-based scanning, and may miss probing of sparse networks [29].

Another tool that detects Scanning Hosts is EMERALD (Event Monitoring Enabling Responses to Anomalous Live Disturbances), presented in 1998. It builds statistical profiles for source IP addresses that communicate with the monitored network, so-called *subjects*, in order to match the short-term weighted profiles of the behavior of the considered subject with its long-term weighted profile [31]. If the short-term behavioral profile of the subject is within the tails of the distribution of the long-term profile, the subject is labeled as *suspicious.* Scanning Hosts are detected when the volume of SYN packets coming from a singular source IP address suddenly increases [29, 31].

A probabilistic approach, proposed in [32] in 2002, relies on the probability that the IP address d of the probed network is contacted by the IP address s (not within the probed network), $p(d|s)$, based on the probability that d is contacted by any other source IP address, $p(d)$, and the probability that port p receives a packet sent by the source IP address s is $p(p|s)$, based on the probability that p receives a packet from any source, $p(p)$, to detect Scanning Hosts [32]. Some limitations of this approach are that the aforementioned probabilities $p(d|s)$ and $p(p|s)$ may include both probing and normal traffic, and that the approach assumes that the IP addresses d of the probed network are accessed randomly, which is not always the case [29].

Since 2003, Network return traffic can be leveraged to reconstruct connection sessions, and to flag any source host that contacts IP addresses which do not reply. An *anomaly score*, based on the number of contacted IP addresses that do not reply, is computed for each identified source host. This approach is able to view almost all traffic both ingoing and outgoing [33], but there is no warranty that it can be leveraged over large networks, due to asymmetric rating policies [29]. In case only traffic in one direction is available, *Peering center Surveillance Detection* (PSD) can be used to analyze traffic [33], but in this case a lack of response does not necessarily indicate a probing [29].

In the same year, MINDS - Minnesota INtrusion Detection System was developed. It analyzes Network traffic to detect Scanning Hosts. From Netflow data, MINDS generates characteristics of data and derives information, like number of connections:

- from a single source IP address;
- to a single destination IP address;
- from a single source IP address to the same destination port;
- from a single destination IP address to the same source port.

All these counters are counted over a time and a connection window, respectively [34], and leveraged, together with the flow data, to compute an anomaly score. MINDS is able to detect both fast and slow Scanning Hosts [29].

Since 2004, Scanning Hosts can be identified with a statistical approach that leverages *statistical tests* to analyze traffic rates, *detection* of anomalous packets and *models* of network traffic as a *marked point process* [29, 35]. Recall that a *marked point process* is composed of couples of locations and marks. This detection approach has been proposed and tested in [35], that detects malicious vertical probings with true positive rate higher than 90% and false positive rate less than 15%.

A conceptual model, proposed in 2005 and built on:

- source IP address;
- destination IP address;
- datagram size;
- transport layer protocol field;
- fragmented information;
- checksum [36];

is leveraged to analyze the probability that the monitored network is probed by a Scanning Host. The analysis is based on Network layer data. This approach reveals to be effective and robust in term of size of needed datasets and detection rates [29].

One year after the aforementioned conceptual approach, Statistical analyses of events are leveraged to detect Scanning Hosts (2006) [37]. More in detail, *events* are defined as bursts of network activities surrounded by periods of quiescence. They can be extracted from traffic data and for each source IP address. Successively, flows within each event are stored. For each event, six characteristics are extracted using statistical analyses of Scanning Hosts, and are then used as input variables to estimate a probability to predict whether the considered event contains a Network Scanning Activity performed by a single Scanning Host. The main disadvantage of this approach is that it is not in real-time, since it needs the whole event to extract its characteristics [29].

In the same year, an approach based on data mining was proposed. It is based on data of network traces, which are collected and transformed into feature datasets, containing also label information. Then, the fast rule classifier Ripper, able to learn rules from multi-model datasets, provides results in an easily interpretable manner [38]. The strong point of this approach is its ability to detect Port Scanning Activities at an early stage [29].

The last detection technique we present here is an improvement of SNORT: despite it is a threshold-based approach to detect Scanning Hosts and will thus be detailed later on in the predisposed section, its improvement leverages algorithmic approaches. For now, it suffices to know that SNORT looks for IP addresses that attempt more than a fixed amount of connections in a fixed interval of time [39]. Its improvement, proposed in [40], leverages heuristics written into firewalls and triggered immediately after a Scanning Host is identified. Packets whose source IP address is the one identified to be of a Scanning Host are automatically dropped for a predetermined period of time [40]. This approach has the advantage to be more user-friendly than SNORT [29].

3.2.2 Threshold-based Detection Approaches

Threshold-based approaches examine events of interest across intervals of time with fixed length, in order to detect Scanning Hosts: when the considered event (generated by a given source IP address) appears, within a time window, more than the fixed threshold, then a Scanning Host is detected. Despite the meaning of the term “event” varies according to the approach, it commonly refers to counters of unique IP addresses probed by a single host [29], which will be labeled as a *Scanning Host* if this counter is higher than a threshold.

Network Security Monitor (NSM), developed in 1990, is the first tool to detect Scanning Hosts leveraging a threshold approach [41]. It collects data, analyzes it and re-arrange it into various forms:

statistical data Network traffic, breakdown of protocol and distribution are aggregated;

session data Connection pairs and conversations between hosts are stored;

Full content data Every single bit of network traffic is stored;

Alert data Data collected by an Intrusion Detection System is stored.

A host is labeled as *anomalous* and *potentially malicious* (in other words, as Scanning Host) if all the following conditions are fulfilled:

- it contacts more than 15 IP addresses within a time window of predefined duration;
- it tries to contact an IP address that does not contain a responding host on the monitored network [29, 41].

A threshold approach is also leveraged by Bro, developed in 1998. It labels as Scanning Hosts single source IP addresses that contact a number of destination IP addresses, or a number of ports, higher than a given threshold [42]. Despite it is able to detect Scanning Hosts in real time, it produces a high number of false positives [29].

The aforementioned SNORT, developed in 1999 [39], is an Intrusion Detection System based on signatures able to extract Scanning Host. It leverages either a threshold approach (an event occurs more times than a fixed value within an interval of time), or invalid flag combinations of received packets. The preprocessor *portscan*, which identifies a Scanning Host by observing connections, is deployed. Snort raises an alarm if and only if it detects SYN packets that are sent, within the monitored network, to at least 5 different IP addresses or 20 different ports within 60 seconds [39]. The thresholds can be adapted, but keeping in mind that raising them makes slow scans with rates lower than the thresholds go undetected, despite the number of false positives would be reduced [29].

Another tool that leverages a threshold-based approach is proposed in 2000, *Flow-dscan*. It examines flows in order to identify floods and Scanning Hosts, where a *flood* identifies an excessive number of packets per flow. A *Scanning Host* is instead identified as a source IP address that contacts a number of IP addresses or of destination ports below 1024 and on a single host greater that a threshold [43, 29].

Four years later, *Threshold Random Walk* (TRW) has been proposed to detect Scanning Hosts. It leverages sequential hypothesis testing. When a request of connection is received, both the source and the destination IP addresses are stored into a list. If the destination IP address is already present, then the connection request data is ignored. Otherwise, a measure is leveraged to determine whether the connection request is a probing attempt, computed and added to the list entry. The source IP address of the connection attempt is flagged as a Scanning Host if the aforementioned measure exceeds the maximum threshold or is below the minimum threshold, respectively [44]. Benign activity rarely connect to unavailable hosts or services, whereas Scanning Hosts often do. The probability that a Scanning Host connects to a legitimate IP address within the monitored network depends on the density of the latter [29].

Another threshold-based approach is proposed in 2006 in [45]. Here, Scanning Hosts are detected in real time by leveraging counters of couples (destination_IP_Address - destination_port) that receive a connection attempt from a pair (source_IP_Address - source_port). If this counter is higher than a threshold, then a Scanning Host is detected [45], with high accuracy [29]. Scanning Hosts are then grouped together and reported immediately to defenders of the monitored network [29, 45].

In the case that the only information available is on unidirectional flow, the *Time based Flow size Distribution Sequential hypothesis testing* (TFDS) can be leveraged to detect Scanning Hosts probing high-speed transit networks [46]. Proposed in 2009, this approach leverages entropy and Flow Size Distribution (FSD) to identify access patterns of hosts sending probes to the monitored network. If these patterns are abnormal, then the host is labeled as a Scanning Host. TFDS is proved to enhance TAPS [47, 29]

3.2.3 Soft computing based Detection Approaches

Approaches of detection of Scanning Hosts that are based on *soft computing techniques* are able to handle real life ambiguous situations providing flexible information [29]. Tolerance for imprecision and uncertainty, approximate reasoning and partial truth are leveraged to achieve traceability. Solutions that they propose are low-cost and robust.

The first approach leveraging soft computing techniques was proposed in 2001 [48]. It detects low profile Scanning Hosts as hosts that attempt not more than 10 connections, or that wait more than 59 seconds between an attempt and the following [48]. They are identified through analyses and real time monitoring of packets and probes that transit over bidirectional network links, in order to read packets and probes in real time. Likelihood to find a particular connection, together with the assumption that legitimate connections appear more often that Denial of Service Attacks and Network Scanning Activities, is used to estimate an anomaly score. Connections are classified leveraging an *Artificial Neural Network* (ANN) [29, 49].

Another soft-computing based approach was proposed seven years later, in 2008, and is an enhancement of SNORT. More precisely, it enhances the detection of Scanning Hosts by adding a fuzzy logic controller to SNORT. Fuzzy logic comes into help for various motivations. Firstly, there does not exist clear and well defined boundaries between normal and abnormal events: fuzzy logic helps in smoothing the abrupt separation of normal and abnormal. Furthermore, it increases the accuracy with which bad traffic is determined, and it gives a rank for each type of Scanning Hosts [29, 50].

Another approach was proposed in the same year. It identifies, with high accuracy [29], flooding attacks and probing activities originated from a single Scanning Host leveraging a Naive Bayes Kernel Estimator (NBKE), after all known attacks are represented leveraging traffic features. The Naive Bayes Kernel Estimator is trained with instances of traffic that are identified by hand. The reason that lies behind the choice of leveraging the Naive Bayes Kernel Estimator is that it improves accuracy of 96.8% over the one reached with the simple Naive Bayes Estimator [51].

A comparison of detection approaches, published in 2008 [52], compares approaches based on Fuzzy Inference Systems, Neural Networks and Adaptive Neuro-Fuzzy Inference Systems (ANFISs). The latter is more complex than others, since it combines classical Fuzzy Inference Systems. On the other hand, this combination achieves excellent accuracy of classification [29].

3.2.4 Ruled based Detection Approaches

Behavior of intrusive traffic can be differentiated from normal traffic behavior through analysis of traffic data performed by Rule-based Intrusion Detection Systems [29]. They are able to detect anomalies within traffic and unauthorized activities, and take countermeasures.

The first detection approach to fall into this category, proposed in 2001, deploys the Packet Header Anomaly Detection (PHAD) to learn normal values for all the 33 fields in headers and to assign a score to each of them of each received packet. Fields whose value does not fail into learned intervals are flagged as *suspects*. An *aggregate anomaly score* is then computed for each packet, by summing up all the scores of the fields of its header. Packets whose score is one of the top n are labeled as *anomalous* [53], where the value of n varies in order to obtain ROC curves [29].

In case that the observed network is being probed by a slow Scanning Host, an Abnormal Traffic Control Framework, which leverages fuzzy logic rules, can come into help to detect the probing slow Scanning Hosts. It acts as an intrusion prevention system, since it disallows network traffic that appears suspicious by decreasing of the network bandwidth and discarding traffic [54].

3.2.5 Visual Detection Approaches

Visualization of data can come into help for the user that needs to recognize probing traffic and Scanning Hosts by the patterns the latter generate [29]. Security experts with this need had the first two available approaches only in 2004, when Network events could be detected through visualization approaches. The approach proposed by Conti [55] leverages packet level information and parallel coordinate plots to illustrate relationships amongst ports and IP addresses, and that different Network and Vulnerability Scanners (Nmap [23], SuperScan, Nessus ...) have different fingerprint patterns, which are identifiable in visual data [55].

The other visualization approach proposed in 2004 is NVisionIP [56], which is a visualization system based on bidirectional flow level data. It allows the user to quickly view all the connection activities on the monitored network, activities that appear as horizontal stripes [29, 56].

One year later other two approaches have been presented. One leverages visualization techniques of network traffic to recognize attacks and probing attempts in real time. Port based overviews come into help to detect and respond to malicious activities. In case of large networks, i.e., when there is a wide range of IP addresses, and of wide ranges of port numbers, visualizing finer details is necessary. It can be done through scaling techniques [57].

The other, instead, leverages large-scale commodity clusters in NVisionCC [58]: observed open ports are gathered into clusters that are correlated with the number and nature of active processes and the content of important system files [29].

The detection tool PortVis, proposed in 2006 [59], uses couples (source_IP_addresses, destination_IP_addresses) for each protocol and port and for each time window, whose size is specified by the user, summaries of network traffic including numbers of unique source IP addresses, destination IP addresses. It uses visualization to find horizontal or vertical Scanning Hosts within the monitored traffic [59], and needs then to be analyzed by hand [60].

Authors of [61] propose, in 2007 a software that incorporates projections of alert data in 3-dimensional displays, filtering, drill down and playback of alerts at variable speed, in order to enable, in an effective manner, visualization alerts, which are then classified into false and true by a decision tree algorithm [29].

The last visual approach we present is ScanViewer [62], proposed in 2008. It provides interactive visualizations that represent traffic activities. The system combines characteristics of Network Scanning with novel visual structures, and maps the collected datagrams to graphs that emphasizes their patterns [62]. It included a tool that captures large scale port information, called Localport. ScanView is able to detect Network and Port Scanning, coordinated Port Scanning Hosts and hidden Scanning Hosts [29].

3.3 Detection of Coordinated Network Scanning Activities

This section provides an overview of approaches and techniques that can be deployed to detect coordinated Network Scanning Activities. Four classes of approaches are available: *Algorithmic* (Section 3.3.1), *Clustering* (Section 3.3.2), *Soft Computing* (Section 3.3.3) and *Visual* (Section 3.3.4) approaches. Most of the techniques to detect coordinated Network Scanning Activities leverage packet level information, like source IP addresses, destination IP addresses, connection information, destination port and source port of probes. The main disadvantage of approaches to detect coordinated Network Scanning Activities is that they are not in real time. In order to choose what detection technique to use, an analysis of their performances comes into help. Recall that their performance is evaluated in terms of False Positive Rates (i.e. number of normal connection that are labeled as probing attempts) and Detection Rates (i.e., the number of cases that are correctly detected) [29], and is biased on the dataset used to evaluate the approaches. The remaining of the section details approaches that fall into each of the aforementioned categories.

3.3.1 Algorithmic approaches

Two approaches are detailed here. The first to be proposed, in 2006, builds models of potential performers leveraging information wished by performers of Network Scanning Activities [63].

The obtained models of performers of NSAs form the basis of a detection approach able to identify coordinated Network Scanning Activities, and allow comparisons of types of performers that could be identified by different approaches of detection of coordinated Network Scanning Activities [63, 29].

The other approach we detail, proposed in [64], detects both internal and external Network Scanning Activities within the boundaries of the network of an enterprise. It leverages different prototypes of detection tools, and provides low false positive and false negative rates. Local (i.e., performed within the boundaries of the enterprise network) Scanning Activities can be identified by just few scanning attempts, which suffice. Network Scanning Activities originated outside the enterprise network are detected through *Exposure maps*, which enable:

- options of active responses, in order to be safely focused *only* on systems that are a threat for the network of the enterprise;
- characterization and clustering of hosts within a network into different exposure profiles, based on the offered services;
- the ability to perform a Reconnaissance Activity Assessment (RAA), which determines what specific information was returned to the performer of the Network Scanning Activity [64].

3.3.2 Clustering approaches

Approaches to detect Coordinated Network Scanning Activities that fall into this category partition data into clusters of similar elements, with an unsupervised learning process.

In 2001, authors of [49] analyze information about connections, current sessions and alerts generated by probes, including type of received probe, the source IP address, the timestamp and the duration of the connection, are clustered into series of tables, to see whether any of them form a coordinated Network Scanning Activity [49, 29].

One year later, an *Intrusion Correlation Engine* [65] leverages anomalousness of packets and heuristics developed from real Network Scanning Activities to detect stealthy NSAs. More precisely, the aforementioned heuristics are leveraged to cluster anomalous packets into Coordinated NSAs. The Intrusion Correlation Engine is able to detect all the Network Scanning Activities that are detected by other detection techniques that were available up to 2002, and also many of them that leverage stealthy approaches [65], without too many false positives [29].

In 2003, *Scanner* [33] has been proposed. It is able to detect coordinated Network Scanning Activities that deploy Scanning Hosts that are located close to each other. The reasoning at the base of *Scanner* is that if an IP address is performing a probing of a network, other IP addresses near it probably probe the same network, too [33].

In the same year, authors of present a system that checks if different source IP addresses start and stop probing either at the same time, or with very similar temporal patterns [66]. It allows to detect Coordinated Network Scanning Activities that leverage various Scanning Hosts that target a particular destination IP address within a temporal window of one hour.

3.3.3 Soft Computing approaches

Since 2000, intelligent agents can be leveraged to detect coordinated Network Scanning Activities [67]. The *Master Analysis Agent* of the proposed intrusion response architecture is able to find a *confidence measure* based on observed false positive rates, and to determine whether the current

Network Scanning Activity is a continuation of a previous one, or a new one. It leverages characteristics of the Network Scanning Activity, which are time between reports of incidents; source IP address; destination IP address; username and program name [67].

A *distributed multi layer cooperative model*, which merges feature-based, scenario-based ans statistic-based detection approaches [68] has been proposed in 2008 and it detects common Network Scanning Activities and their variants, slow Network Scanning Activities, camouflage attacks and DoS attacks that leverage the TCP/IP protocol.

3.3.4 Visual Approaches

Visual approaches to detect coordinated Network Scanning Activities visualize network traffic, in order to individuate attack behavior of flows of network packets and to distinguish it from normal attitude [29]. Visual approaches to detect coordinated NSAs are the same described in Section 3.2.5 and able to detect Scanning Hosts.

3.4 Evaluation Criteria of detection approaches of Network Scanning Activities

An important question is hot to properly choose the detection approach to deploy to protect the monitored network. The approaches usually leverage *Receiver Operating Characteristics (ROC) curves*, completeness, correctness and *detection metrics*. The latter include *detection rates*, also known as *true positive rates*, *true negative rates*, *false positive rates* and *false negative rates*. All of them are affected by fallacy, which is related to the volume of normal traffic compared to the volume of malicious traffic: attacks are rare, when compared to the volume of traffic generated by normal activities and connection. Even when the false positive rate is low (i.e., around 1% or less), their number could be overwhelming for security and traffic analysts. The aforementioned metrics are leveraged to compute *effectiveness* and *efficiency* of the considered approach, as follows:

$$\text{Effectiveness} = \frac{|\text{true positive}|}{|\text{true positive}| + |\text{false negative}|} = \frac{|\text{detected scans}|}{|\text{all scans}|} \tag{3.1}$$

$$\text{Efficiency} = \frac{|\text{true positive}|}{|\text{true positive}| + |\text{false positive}|} = \frac{|\text{detected scans}|}{|\text{all cases flagged as scans}|} \tag{3.2}$$

Receiver Operating Characteristic (ROC) curves can be instead leveraged to evaluate the performance of a detector of anomalies. They take, as input, the *false positive rate*, and their output is the *true positive rate*. Their input and output values are thus included included in the interval [0; 1]. The same interval contains the likelihood score that detection systems return for each instance they analyze [29].

3.5 Analysis and lacks of detection approaches

To increase the network security, firewalls are often deployed thanks to their ability to block TCP, SYN and SYN/ACK scanning. Blocking other types of scanning is, on the other hand, still an open issue. What is wanted, and unfortunately still missing, is a detection tool that is able to detect both known and yet unknown scanning approaches and which leverages both header and payload information from packet analysis. Recall that most detection approaches leverage packet level information, like connection information and packet payload. After having presented all the categories of detection approaches, the spontaneous question is what class of methods is, in some way, the "best". Threshold methods turn out to be the most *effective*, despite they are highly sensitive to the input parameters (the thresholds imposed by their user) and depend on the network scenario, The development of a generic threshold-based detection approach across different and various network scenarios, and less dependent on the inputs, is still ongoing [29]. Non-adaptive Intrusion Detection Systems should be, instead, discarded since they are not able to handle either known or unknown attacks or intrusion attempts, because of the evolving nature of technology and the increasing effort of attackers and intruders. Furthermore, actual approaches of detection of Network Scanning Activities cause too many false alarms, which need to be reduced.

An open question that security experts need to face, and that is strictly dependent on the monitored network, is where to place modules of detection of port scanning and their best configurations for usage within a multi-stage environment. These models have to to build profiles of normal network behavior, which is a complicated task for various reasons. First of all, because of the huge size of network traffic data that needs to be handled and analyzed. Then, because traffic patterns are always updating and changing. Finally, because of the presence of noise within data. Machine learning approaches or soft computing methods that are available up to date are not yet adequate.

3.6 Summary and conclusions

This chapter provided an excursus over existing approaches and techniques to detect a Network Scanning Activity targeting the monitored network. Table 3.1 provides a visual summary of them. The line of the reference [33] deserves some explanation: the • refers to the algorithmic approach to detect a Scanning Host, whereas the ⊙ indicates the clustering approach to detect coordinated Network Scanning Activities.

But, once security experts know that the network they survey is being probed, the next question that arise is what Network Scanner has been being leveraged by opponents. The literature, at the best of our knowledge, seems not to be really interested in addressing it, since the only work that answer this question is[9]. Despite its merit of being interested in the recognition of the Network Scanner that originated the probing packets, its approach is naive. Precisely, authors are able to originate what Network Scanner sent the analyzed packet by checking its IPid or the TCP sequence number. They claim that probes are originated by *Masscan* if its IPid field of the header is the value obtained by applying bit a bit the logic operator XOR to the header fields dstIP, dstPort and tcpSeq (TCP sequence number). The monitored packet has been instead originated by ZMap if its IPid header field is 54321. Authors also claim to recognize packets generated by NMap and Unicorn by analyzing their TCP sequence number, in a manner similar to the ones seen for ZMap and Masscan. This approach they propose is naive for the reason that the Network Scanner they consider are opensource: as they got the information about the monitored header fields, malicious users have access and modify it at will. In this case, the approach proposed in [9] is not able anymore to recognize the Network Scanner that originates the probes.

We filled this gap, by proposing a methodology that does leverage not the code of the Network Scanner, but information that can be collected by the monitored network. Hidden Markov Models reveal to be a manageable tool to recognize Network Scanners from the point of view of the monitored and probed network. The next chapter will provide all the details of Hidden Markov Models that will be then leveraged, in chapters 5 and 6, to model and recognize

Ref.	SH	NSA	Algorithmic	Threshold	Soft-computing	Ruled	Visual	Clustering
[28]	●		●					
[30]	●		●					
[31]	●		●					
[32]	●		●					
[33]	●	⊙	●					⊙
[34]	●		●					
[35]	●		●					
[36]	●		●					
[37]	●		●					
[38]	●		●					
[40]	●		●					
[41]	●			●				
[42]	●			●				
[39]	●			●				
[43]	●			●				
[44]	●			●				
[45]	●			●				
[46]	●			●				
[48]	●				●			
[50]	●				●			
[52]	●				●			
[53]	●					●		
[54]	●					●		
[55]	●	●					●	
[56]	●	●					●	
[57]	●	●					●	
[58]	●	●					●	
[59]	●	●					●	
[60]	●	●					●	
[61]	●	●					●	
[62]	●	●					●	
[63]		●	●					
[64]		●	●					
[49]		●						●
[65]		●						●
[66]		●						●
[67]		●			●			
[68]		●			●			

Table 3.1: Approaches of detection of Network Scanning Activities

Network Scanners.

4

Stochastic models and algorithms

Contents

4.1 Introduction

The first chapters of this manuscript aimed at positioning our work within the state of the art of Network Scanning Activities (NSAs) and their associated models. Chapter 1 gave an overview on Advanced Persistent Threats (APTs) and their common 7 phases. It also pointed out that the Reconnaissance phase, despite it is not harmful yet, is preliminary to a targeted attack. Network Scanning Activities (NSAs) are generally used by attackers to collect information on the system to attack. Strong defenses to reconnaissance techniques and tools are thus required. Our work aims to provide tools to model features of NSAs and to early detect them, in order to warn as soon as possible security teams, who will activate the needed defenses. The following three chapters show in detail the proposed approach (Chapter 4) and the obtained results (Chapters 5 and 6). More in detail, Chapter 4 provides an overview of the statistical modeling tools that we used to elaborate learning models to fingerprint network scanners. It recalls definitions of Finite Mixture Models (FMMs), explains their mathematical equations and provides the details of the Expectation-Maximization (EA) algorithm widely used to estimate the parameters of these models. It then describes the mathematical formulation of Hidden Markov Models and their associated algorithms (Forward-Backward, Viterbi, Baum-Welch) in Section 4.4.4, with respect to the problems that they solve.

The first issue to handle is to detect what situations require the existence of models of Network Scanning Activities, which are useful to learn and gather information about the process that originated the received packets, without its availability [69], and to characterize the observed outputs [70]. A *model* can be *deterministic*, if it exploits known and specific properties of the output which is fully determined by the parameters values and the initial conditions, or *statistical*, that instead characterizes the statistical properties of the output, and some of its parameters and initial conditions provide an ensemble of different outputs. The underlying assumptions are that the output can be characterized as a parametric random process, and that the parameters of the stochastic process can be estimated in a precise and well-defined manner [69]. Since the Network Scanning Activity is unknown from the scanned system's point of view, and is only observed through logs, no specific properties of the observations are available. Statistical models can be thus used to characterize the collected logs and infer behavioral patterns of NSAs. NSAs can exploit a combination of one or more Network scanners, which could be executed one or multiple times. At best of our knowledge, there does not exist a model of Network Scanners developed by using only logs or information collected by the targeted network or system. Our work aims at filling this gap, by developing models of two Network Scanners from logs of packets received by the hosted and targeted network. Hidden Markov Models (HMMs) are well suited to accomplish this goal. They are characterized by the fact that the distribution generating an observation depends on the value of an underlying but unobserved Markov process or chain [71]. Each value (the so-called *state*) of the hidden Markov process is provided with a probability. In other words, the unobserved Markov chain of the Hidden Markov Model can assume a (finite) number of values, called *states*, described by an initial probability distribution which is, in fact, the *mixture distribution* of a *Finite Mixture Model.*

The chapter is structured as follows. Section 4.2 provides an excursus on Finite Models. Details of a Markov Processes are presented in Section 4.3. The chapter ends with Section 4.4 on Hidden Markov Models, providing their definitions, parameters, and three main problems.

4.2 Finite Mixture Models (FMMs)

Finite Mixture Models (FMMs) are used when data cannot be well fitted by only one single distribution, but by a finite number of them. An example is presented in Figure 4.1. Data, within the range $[-10; 30]$, have a mean of 5.143 and a variance of 26.176. The Gaußian distribution defined with these two parameters, unfortunately, does not well fit the data. The reasons of this lies in the fact that observations may be clustered into unobserved clusters, each of them with its own probability distribution for the observed variable. In case of the example in Figure 4.1, data are clustered into three clusters, each with its own probability and fitted by a Gaußian distribution. Each of them has a variance equal to 5.146 and is summarized in Table

Component i	**Probability** p_i	**Mean** μ_i
1	0.237	-1.32
2	0.635	5.69
3	1.128	14.39

Table 4.1: Details of components of the FMM in Figure 4.1

If the selection of one cluster is independent from the previous choice, then the FMM is said to be *Independent* [71]. FMMs aim thus at finding:

1. the number of unobserved clusters (3 in Figure 4.1) into which data are divided;

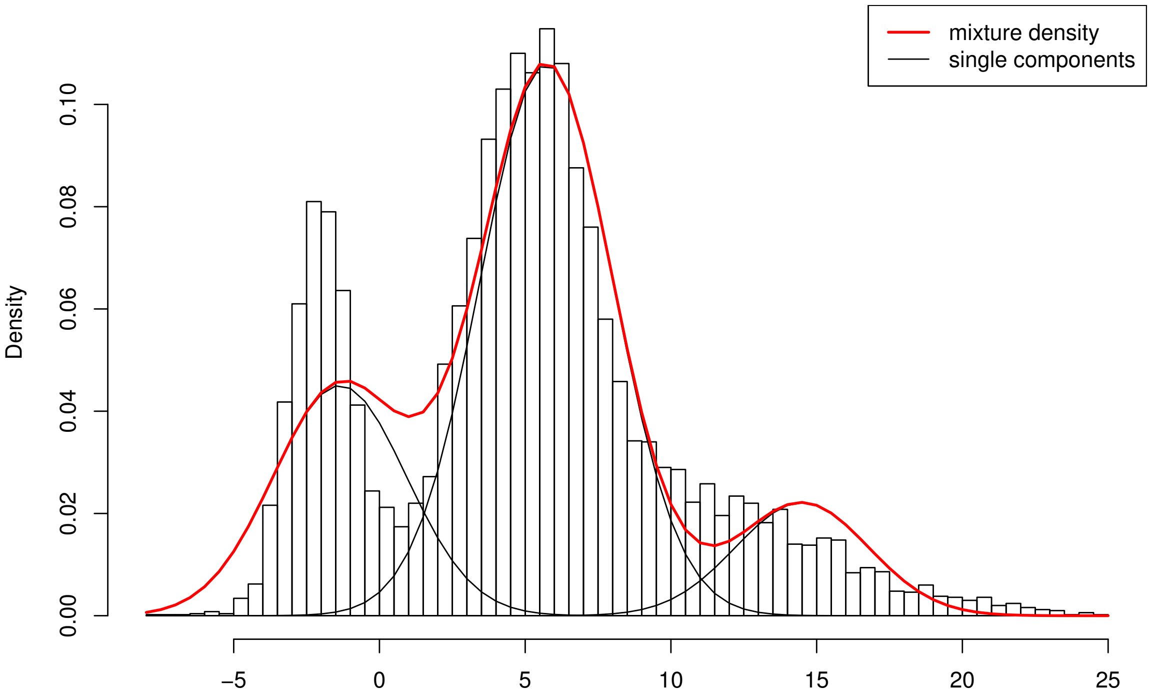


Figure 4.1: Fitting observations with 3 Gaußian distributions

2. the probability of each cluster (0.237, 0.635, 0.128);

3. the probability distribution of each cluster, the so-called “component” of the FMM (three Gaußian distributions).

Distributions of clusters don’t need to be of the same type: it means that an FMM can have a Gaussian component, a Bernoulli one, a Poisson one, etc.

FMMs consist of m components , i.e., m distributions $p_1, p_2, \ldots, p_m$ (continuous or discrete) each associated to one unobserved cluster of observations, and a *mixing distribution* $\boldsymbol{\delta} = (\delta_1, \delta_2, \ldots, \delta_m)$ that selects one of the m components and whose elements are the probabilities of each component. This means that

$$\sum_{i=1}^{m} \delta_i = 1, \qquad \delta_i \geq 0 \quad \forall i = 1, \ldots, m. \tag{4.1}$$

The selection of the component is performed by a discrete random variable C, which assumes one of the m values in $S = \{S_1, S_2, \ldots, S_m\}$, called *states* (one for each component) and each with probability δ_i, $\forall i = 1, \ldots, m$. The value of C is not known: this means that the component that was active when the observation was performed is not known, In other words, nor the cluster which the observation comes from, nor its distribution $p_i, i = 1, 2, \ldots, m$ being active when the observation is done, are known. Given X the random variable whose values are the observations and whose distribution is the mixture of distributions, its probability density function is then given by

$$p(x, \boldsymbol{\delta}, \boldsymbol{\theta}) = \sum_{i=1}^{m} \delta_i p_i(x, \boldsymbol{\theta_i}) \tag{4.2}$$

where the vector $\boldsymbol{\delta}$ is the *mixing distribution vector*, the vector $\boldsymbol{\theta}$ contains the parameters of all the components of the mixture, and $\boldsymbol{\theta_i}$, $\forall i = 1, 2, \ldots, m$ the parameters of the distribution i. In the discrete case, Equation 4.2 can be rewritten as:

$$\Pr(X = x|\boldsymbol{\theta}) = \sum_{i=1}^{m} \Pr(X = x|C = i, \boldsymbol{\theta_i})\Pr(C = i|\boldsymbol{\theta_i}). \tag{4.3}$$

Called $Y_i, i = 1, \ldots, m$ the random variable with probability function p_i, it follows that:

- the *expectation of the mixture* is

$$\mathrm{E}(X|\boldsymbol{\delta}, \boldsymbol{\theta}) = \sum_{i=1}^{m} \Pr(C = i|\boldsymbol{\theta_i})\mathrm{E}(X|C = i, \boldsymbol{\theta_i}) = \sum_{i=1}^{m} \delta_i \mathrm{E}(Y_i|\boldsymbol{\theta_i}) \tag{4.4}$$

- the k^{th} *moment of the mixture* is

$$\mathrm{E}(X^k|\boldsymbol{\theta}) = \sum_{i=1}^{m} \delta_i \mathrm{E}(Y_i^k|\boldsymbol{\theta_i}) \tag{4.5}$$

- the *variance of the mixture*, in the case of two components of the mixture, is

$$\mathrm{Var}(X|\boldsymbol{\theta}) = \sum_{i=1}^{2} \delta_i \mathrm{Var}(Y_i|\boldsymbol{\theta_i}) + \delta_1\delta_2\big(\mathrm{E}(Y_1|\boldsymbol{\theta_1}) - \mathrm{E}(Y_2|\boldsymbol{\theta_2})\big)^2 \tag{4.6}$$

Briefly, a Finite Mixture Model (FMM) is composed by a mixture distribution $\boldsymbol{\delta}$ and m components, each of them with its probability δ_i, $\forall i \leq m$, and consisting of a probability function and a cluster of observations. In order to define a FMM, one thus needs to define:

- the number m of components;
- the *mixing distribution vector* $\boldsymbol{\delta}$;
- the parameters of each component, i.e., the parameters of its distribution, stored in the vector $\boldsymbol{\theta_i}$.

The estimates of the latter two is detailed in Section 4.2.1, whereas the answer to the first point is given in Section 4.2.2.

4.2.1 Estimation of parameters

The estimation of the parameters of the distribution of the m components of the FMM is performed by maximizing the *likelihood* (ML), which is a function that assigns a number to each couple $(x, \boldsymbol{\theta})$ of observation x and vector of parameters $\boldsymbol{\theta}$. To have a clearer idea, let us consider the simplest case. Let us have a single observation x of a phenomenon described by a single probability density p depending on its parameters $\boldsymbol{\theta}$. Note that the type of the probability density (e.g. Gaussian, Bernoulli, etc.) is known. In this case, the likelihood of the observation x is

$$L(x, \boldsymbol{\theta}) = p(x, \boldsymbol{\theta}) \tag{4.7}$$

where $p(x, \boldsymbol{\theta})$ is a probability density in the continuous case, and a probability in the discrete case. Let us now extend the scenario, supposing to have T observations, x_j, $j \leq T$, of the phenomenon, that are divided into m clusters, each with a probability $\delta_i, i \leq m$, collected in the mixing vector $\boldsymbol{\delta}$. Equivalently, one can say to have a partition of the events space. Each cluster has its own probability density p_i with its own parameters vector $\boldsymbol{\theta}_i$, and the selection of one of them is performed by the random variable C as described above. Then, the likelihood of one of the observed x_j, $j \leq T$, which is its probability density, is given by

$$\begin{aligned} L(x_j, \boldsymbol{\delta}, \boldsymbol{\theta}) = p(x_j, \boldsymbol{\delta}, \boldsymbol{\theta}) &= \sum_{i=1}^{m} p(x, C = S_i, \boldsymbol{\delta}, \boldsymbol{\theta}) \\ &= \sum_{i=1}^{m} p(x_j | C = S_i, \boldsymbol{\theta}) p(C = S_i, \boldsymbol{\delta}) \\ &= \sum_{i=1}^{m} \delta_i p_i(x_j, \boldsymbol{\theta_i}). \end{aligned} \tag{4.8}$$

It follows that the likelihood of the sequence of observations $\boldsymbol{x} = (x_1, x_2, \ldots, x_T)$ is given by

$$\begin{aligned} L(\boldsymbol{x}, \boldsymbol{\delta}, \boldsymbol{\theta}) &= \prod_{j=1}^{T} p(x_j, \boldsymbol{\delta}, \boldsymbol{\theta}) \\ &= \prod_{j=1}^{T} \left[\sum_{i=1}^{m} \delta_i p_i(x_j, \boldsymbol{\theta_i}) \right] \end{aligned} \tag{4.9}$$

where, recall:

- $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_m$ are the vectors, one for each component of the mixture, containing *parameters* of the corresponding distribution i, $i \leq m$;

- $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_m)$ is the *mixing probability vector*, i.e., its elements are the probabilities of each component and thus satisfy the conditions $\forall i = 1, \ldots, m, \delta_i \geq 0 \wedge \sum_{i=1}^{m} \delta_i = 1$;

- $\boldsymbol{x} = (x_1, \ldots, x_T)$ is the vector containing the T observations.

Recall that the problem to be solved here is how to choose or correctly select the parameters of the FMM. One way to proceed is to maximize the likelihood function (4.9) and to store the parameters corresponding to its maximum. Note that the $p_i(x_j, \boldsymbol{\theta_i})$, $\forall i \leq m, \forall j \leq T$ appearing in Equation 4.9 are probabilities in the discrete case, or probability densities in the continuous case. In both scenarios, our focus lies in the *parameters* that maximize the probability of the observed sequence $\boldsymbol{x}$, which are also equivalently found in the continuous case by maximizing the probability density. It thus follows that the Maximum Likelihood (ML) is given by:

$$\overline{L} = \max_{\substack{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_m \\ \boldsymbol{\delta}}} \Big[L(\boldsymbol{x}, \boldsymbol{\delta}, \boldsymbol{\theta}) \Big]. \tag{4.10}$$

The parameters corresponding to the maximum of the likelihood $\overline{L}$,

$$(\boldsymbol{\delta}^*, \boldsymbol{\theta}^*) = \arg\max_{\boldsymbol{\delta},\boldsymbol{\theta}} L(\boldsymbol{x}, \boldsymbol{\delta}, \boldsymbol{\theta}), \tag{4.11}$$

are found through an iterative procedure, called *Expectation - Maximization* (EM) algorithm [72], presented in detail in the following sub-section.

Expectation - Maximization Algorithm

As mentioned above, the *Expectation - Maximization* (EM) algorithm is an iterative procedure that finds estimations of the parameters of the maximum likelihood when some data are missing. It finds local maximums of $\log\left[L(\boldsymbol{x}, \boldsymbol{\delta}, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_m)\right]$ and is based on the idea that the observations in the vector $\boldsymbol{x} = (x_1, \ldots, x_T)$ are incomplete data, and the missing part is the vector of T labels, $\boldsymbol{z} = \left(\boldsymbol{z^{(1)}}, \boldsymbol{z^{(2)}}, \ldots, \boldsymbol{z^{(T)}}\right)$. Each element $\boldsymbol{z^{(j)}}$, $\forall j = 0, 1 \ldots, T$ of the vector $\boldsymbol{z}$ is a binary vector, $\boldsymbol{z^{(j)}} = \left(z_1^{(j)}, z_2^{(j)}, \ldots, z_m^{(j)}\right)$, whose elements $z_i^{(j)}$, $\forall i = 1, 2, \ldots, m$ are all equal to 0 except the one corresponding to the cluster k that outputted the observation x_j:

$$z_i^{(j)} = \begin{cases} 0 \text{ if } j \neq k, i \leq m \\ 1 \text{ if } j = k \end{cases} . \tag{4.12}$$

The logarithm of the complete likelihood, which is the one used to estimate the parameters if the *complete* data $(\boldsymbol{x}, \boldsymbol{z})$ were observed, is then given by

$$\begin{aligned} \log\left(L(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\delta}, \boldsymbol{\theta})\right) &= \log\left(p(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\delta}, \boldsymbol{\theta})\right) \\ &= \log\left[\prod_{j=1}^{T}\sum_{i=1}^{m} \delta_i p_i(x_j, \boldsymbol{\theta_i})\right] \\ &= \sum_{j=1}^{T} \log\left[\sum_{i=1}^{m} \delta_i p_i(x_j, \boldsymbol{\theta_i})\right] \\ &= \sum_{j=1}^{T} \log\left[\sum_{i=1}^{m} z_i^{(j)} \delta_i p_i(x_j, \boldsymbol{\theta_i})\right]. \end{aligned} \tag{4.13}$$

Recall that the vector $\boldsymbol{z^{(j)}}$ is a binary vector with only one element equal to 1: it means that in the inner sum in the last line of Equation (4.13), the addend corresponding to the element of $\boldsymbol{z^{(j)}}$ equal to 1 is the only one that survives. This implies that Equation (4.13) can be rewritten as

$$\begin{aligned} \log\left(L(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{\delta}, \boldsymbol{\theta})\right) &= \sum_{j=1}^{T} \log\left[\sum_{i=1}^{m} z_i^{(j)} \delta_i p_i(x_j, \boldsymbol{\theta_i})\right] \\ &= \sum_{j=1}^{T}\sum_{i=1}^{m} z_i^{(j)} \log\left(\delta_i p_i(x_j, \boldsymbol{\theta_i})\right) \end{aligned} \tag{4.14}$$

since the only remaining $z_i^{(j)}$ is equal to 1.

The result of the EM algorithm is a sequence of estimates of parameters

$$\Bigg(\Big(\widehat{\boldsymbol{\delta}}(1),\widehat{\boldsymbol{\theta}}(1)\Big),\Big(\widehat{\boldsymbol{\delta}}(2),\widehat{\boldsymbol{\theta}}(2)\Big),\ldots,\Big(\widehat{\boldsymbol{\delta}}(u),\widehat{\boldsymbol{\theta}}(u)\Big),\Bigg), \qquad u = 1,2,\ldots,U$$

found by alternating its two steps [72], that are repeated until a convergence criterion is satisfied. The resulting couple $\Big(\widehat{\boldsymbol{\delta}}(u),\widehat{\boldsymbol{\theta}}(u)\Big) = \Big(\widehat{\boldsymbol{\delta}},\widehat{\boldsymbol{\theta}}\Big)$ is then a stationary point of the likelihood of the observed data [71].

Estimation step (E - step) The conditional expectations of the missing data given the observations vector $\boldsymbol{x}$ and the current estimates of the parameters $\big(\widehat{\boldsymbol{\delta}}(u),\widehat{\boldsymbol{\theta}}(u)\big)$,

$$\boldsymbol{w} := E\Big(\boldsymbol{z}|\boldsymbol{x},\widehat{\boldsymbol{\delta}}(u),\widehat{\boldsymbol{\theta}}(u)\Big) \tag{4.15}$$

are computed [71, 72]. Since the vectors v $\boldsymbol{z^{(j)}}$ elements of $\boldsymbol{z}$ are binary, their conditional expectations are

$$\begin{aligned} w_i^{(j)} :=& E\big[z_i^{(j)}|\boldsymbol{x},\widehat{\boldsymbol{\delta}}(u)\widehat{\boldsymbol{\theta}}(u)\big] \\ &= 0\mathrm{Pr}(z_i^{(j)} = 0|\boldsymbol{x},\widehat{\boldsymbol{\delta}}(u)\widehat{\boldsymbol{\theta}}(u)) + 1\mathrm{Pr}(z_i^{(j)} = 1|\boldsymbol{x},\widehat{\boldsymbol{\delta}}(u)\widehat{\boldsymbol{\theta}}(u)) \\ &= \frac{\mathrm{Pr}(z_i^{(j)} = 1,\boldsymbol{x}|\widehat{\boldsymbol{\delta}}(u)\widehat{\boldsymbol{\theta}}(u))}{\mathrm{Pr}(\boldsymbol{x}|\widehat{\boldsymbol{\delta}}(u)\widehat{\boldsymbol{\theta}}(u))} \\ &= \frac{\widehat{\delta}_i(u)p_i(\boldsymbol{x}|\widehat{\boldsymbol{\theta}}(u))}{\sum_{j=1}^m \widehat{\delta}_j(u)p_i(\boldsymbol{x}|\widehat{\boldsymbol{\theta}}(u))}. \end{aligned} \tag{4.16}$$

The last equality holds for the Bayes's law. Since log $\Big(L(\boldsymbol{x},\boldsymbol{z},\boldsymbol{\delta}),\boldsymbol{\theta}\Big)$ is linear with respect to $\boldsymbol{z}$, it is sufficient to compute the conditional expectations $\boldsymbol{w} = E\Big(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{\theta}(t)\Big)$, and put it into log $\Big(L(\boldsymbol{x},\boldsymbol{z},\boldsymbol{\delta},\boldsymbol{\theta})\Big)$. The obtained function is called *Q-function*:

$$Q\Big(\big(\boldsymbol{\delta},\boldsymbol{\theta}\big),\big(\widehat{\boldsymbol{\delta}},\widehat{\boldsymbol{\theta}}\big)\Big) := E\Big[log\Big(p(\boldsymbol{x},\boldsymbol{z},\boldsymbol{\delta},\boldsymbol{\theta})\Big)|\boldsymbol{x},\widehat{\boldsymbol{\delta}}(t),\widehat{\boldsymbol{\theta}}(t)\Big]. \tag{4.17}$$

Maximization step (M - step) Estimates of parameters are updated as follows:

$$\Big(\widehat{\boldsymbol{\delta}}(t+1),\widehat{\boldsymbol{\theta}}(t+1)\Big) = \arg\max_{(\boldsymbol{\delta},\boldsymbol{\theta})} Q\Bigg(\Big(\boldsymbol{\delta},\boldsymbol{\theta}\Big),\Big(\widehat{\boldsymbol{\delta}}(t),\widehat{\boldsymbol{\theta}}(t)\Big)\Bigg). \tag{4.18}$$

The Expectation - Maximization algorithm has been thus just used to find the vectors $\boldsymbol{\delta} = (\delta_1,\delta_2,\ldots,\delta_m)$ and $\boldsymbol{\theta_i}$, $\forall i = 1,2,\ldots,m$. The last question to answer is "What is the number of components of the searched FMM, m?. The issue is targeted in the next session.

4.2.2 Choice of the number m of components

Let us suppose to have a Finite Mixture Model, with a given $\overline{m}$ number of components, to describe the observation sequence $\boldsymbol{O},= (O_0,O_1,\ldots,O_T)$. It is worth to wonder if it is the best model for the given sequence of observations $\boldsymbol{O}$. In other words, we still need to find the number m of components of the FMM, in order for it to well model the sequence $\boldsymbol{O}$. The answer is found by using the AIC (Akaike's Information Criterion) [73] and BIC values (Bayesian Information Criterion) [74]. Both are methods to evaluate, compare and select statistical models, and aim to reduce model overfitting that could be caused by the increasing of the likelihood due to adding new parameters. The AIC was developed by Hirotsugu Akaike in 1971 and proposed to the mathematical community in 1974 [73], and is expressed as

$$\mathrm{AIC} = 2k - 2\ln(\overline{L}) \tag{4.19}$$

where:

- k is the number of parameters of the model;
- $\overline{L}$ is the maximized likelihood defined in Equation 4.10.

The BIC instead has been proposed by Gideon E. Schwarz in 1978 [74] and is also known as *Schwarz criterion* (SBC or SBIC). It differs from the AIC for a stronger penalization for additional parameters, being defined as:

$$\mathrm{BIC} = -2\ln(\overline{L}) + k\ln(T) \tag{4.20}$$

where:

- $\overline{L}$ is the maximized likelihood defined in Equation 4.10;
- k is the number of parameters of the model;
- T is the total number of observations (i.e., the number of elements of the sequence of observations $\boldsymbol{O}$).

The preferred models are the ones with lowest AIC and BIC values.

Also the Cumulative Distribution Function (CDF) comes into help to choose a proper number of distributions m. It is sufficient to compare the CDF of the sequence of observations $\boldsymbol{O}$ with the ones of each considered mixture of distributions. The model whose CDF is close to overlap the CDF of observations is to be chosen. The final FMM thus is selected by possibly using all the three criteria.

This section provided a complete overview of FMMs, including the algorithms to find their parameters, and the procedure to select their number of components, m, such that the FMM well describes the observations $\boldsymbol{O} = (O_0, O_1, \ldots, O_T)$. A FMM is devoid of transition probabilities between its components. They could be provided by Hidden Markov Models (HMMs). In order to build and define a HMM, Markov Chains are needed, which are thus presented in detail in Section 4.3.

4.3 Markov Chains

Section 4.2 shows how Finite Mixture Models come into help when observations $\boldsymbol{O} = (O_0, O_1, \ldots, O_T)$ consist of unobserved clusters, and provide them with their corresponding probabilities. Transitions between clusters are performed by a *Markov Chain*, which is a random variable that satisfies the Markov Property. The *Markov Property*, also called *limited horizon assumption* [75], is the first relaxation of the hypothesis of independence of random variables [71],. Given a random variable C that can assume, at each time t, one of the m values in the set of states

$S = \{S_1, S_2, \ldots, S_m\}$, the *Markov Property* states that the value of C at time t depends only on its value at time $(t-1)$ [69, 71]:

$$\Pr(C_{t+1}|C_t, C_{t-1}, \ldots, C_1) = \Pr(C_{t+1}|C_t) \tag{4.21}$$

or, in shorter form, defined $\boldsymbol{C}^{(t)} = (C_1, C_2, \ldots, C_t)$,

$$\Pr(C_{t+1}|\boldsymbol{C}^{(t)}) = \Pr(C_{t+1}|C_t). \tag{4.22}$$

The random variable C that satisfies the Markov Property is called *Markov Chain.* It can assume, at each time t, one of the m different values in the set S. The so-called *transitional probabilities* are simply the conditional probabilities to move to state i at time $s+t$ after being in state j at time s [71]:

$$\Pr(C_{s+t} = S_j|C_s = S_i), \qquad \forall S_i, S_j \in S = \{S_1, S_2, \ldots, S_m\}; \quad \forall s, t \geq 0 \tag{4.23}$$

or, setting $s = t-1$ and $t = 1$ in the previous expression [69]:

$$\Pr(C_t = S_j|C_{t-1} = S_i), \qquad \forall S_i, S_j \in S = \{S_1, S_2, \ldots, S_m\}; \quad \forall t \geq 0 \tag{4.24}$$

If expressions in Equations (4.23) and (4.24) do not depend on the “initial instant” s and $t-1$ respectively, then C is *homogeneous.* Throughout the following, when not explicitly mentioned, we assume that C is homogeneous. In this case, from Expression (4.23), transition probabilities can be denoted as follows:

$$\gamma_{ij}(t) = \Pr(C_{s+t} = S_j|C_s = S_i), \qquad \forall S_i, S_j \in S = \{S_1, S_2, \ldots, S_m\}; \quad \forall s, t \geq 0 \tag{4.25}$$

and are elements of the matrix $\boldsymbol{\Gamma}(t)$:

$$\boldsymbol{\Gamma}(t) = \begin{bmatrix} \gamma_{11}(t) & \gamma_{12}(t) & \cdots & \gamma_{1m}(t) \\ \gamma_{21}(t) & \gamma_{22}(t) & \cdots & \gamma_{2m}(t) \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_{m1}(t) & \gamma_{m2}(t) & \cdots & \gamma_{mm}(t) \end{bmatrix}. \tag{4.26}$$

From expression (4.24), instead, equivalently:

$$\gamma_{ij}(1) = \gamma_{ij} = \Pr(C_t = S_j|C_{t-1} = S_i), \qquad \forall S_i, S_j \in S = \{S_1, S_2, \ldots, S_m\}; \quad \forall t \geq 0 \tag{4.27}$$

and $\gamma_{ij} := \gamma_{ij}(1) \quad \forall S_i, S_j \in S = \{S_1, S_2, \ldots, S_m\}$ are elements of the matrix $\boldsymbol{\Gamma} := \boldsymbol{\Gamma}(1)$, called *transition probabilities matrix* (TPM) that contains one- step transition probabilities:

$$\boldsymbol{\Gamma}(1) := \boldsymbol{\Gamma} = \begin{bmatrix} \gamma_{11} & \gamma_{12} & \cdots & \gamma_{1m} \\ \gamma_{21} & \gamma_{22} & \cdots & \gamma_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_{m1} & \gamma_{m2} & \cdots & \gamma_{mm} \end{bmatrix}. \tag{4.28}$$

It implies that

- $\gamma_{ij} \geq 0 \quad \forall i, j = 1, \ldots, m$
- $\sum_{k=1}^{m} \gamma_{ik} = 1 \quad \forall i = 1, 2, \ldots, m$. In other words, elements of row i are the probabilities to move from state S_i to any other state $S_j, \forall S_j \in S = \{S_1, S_2, \ldots, S_m\}$.

It is easy to prove that the matrix $\mathbf{\Gamma}(t)$, defined in (4.27), satisfies the *Chapman-Kolmogorov equations*:

$$\mathbf{\Gamma}(t+u) = \mathbf{\Gamma}(t)\mathbf{\Gamma}(u) \tag{4.29}$$

by recalling that, given a partition $\{B_1, B_2, \ldots, B_n\}$ of the event space Ω, then every event A can be expressed as

$$A = \bigcup_{k=1}^{n} B_k \tag{4.30}$$

and its probability as

$$\Pr(A) = \sum_{k=1}^{n} \Pr(A|B_k)\Pr(B_k). \tag{4.31}$$

If:

- $A = (C_{s+t+u} = j | C_s = i)$;
- $B_k = (C_{s+t} = k | C_s = i)$;
- $A|B_k = (C_{s+t+u} = j | C_{s+t} = k)$;

then,

$$\begin{aligned}
\mathbf{\Gamma}(t+u) &= \gamma_{ij}(t+u) \text{ by definition of } \mathbf{\Gamma}(t) \\
&= \Pr(C_{s+t+u} = j | C_s = i) \text{ by definition of } \gamma_{ij}(t) \\
&= \Pr(A) \text{ by definition of } A \\
&= \sum_{k=1}^{n} \Pr(B_k)\Pr(A|B_k) \text{ by Equation (4.31)} \\
&= \sum_{k=1}^{n} \Pr(C_{s+t} = k | C_s = i)\Pr(C_{s+t+u} = j | C_{s+t} = k) \\
&= \sum_{k=1}^{n} \gamma_{ik}(t)\gamma_{kj}(u) \\
&= \mathbf{\Gamma}(t)\mathbf{\Gamma}(u).
\end{aligned} \tag{4.32}$$

By the Chapman-Kolmogorov Equations (4.29), it follows that

$$\mathbf{\Gamma}(t) = \mathbf{\Gamma}(1)^t = \mathbf{\Gamma}^t. \tag{4.33}$$

where $\mathbf{\Gamma}$ is the *transition probability matrix* defined in (4.28). The above equation above states that transition probabilities varying with time (or, in other words, transition probabilities at time t) are obtained through a power of the transition probabilities matrix (TPM). This means that no other parameters are needed to take trace of state transitions over time, but the TPM.

Let us suppose that the Markov Chain C is homogeneous. Then, the *unconditional probabilities* $\Pr(C_t = j)$, i.e., the probability that C is in state S_j at time t, are elements of the vector

$$\boldsymbol{u}(t) = \big(\Pr(C_t = S_1), \Pr(C_t = S_2), \ldots, \Pr(C_t = S_m)\big) \qquad \forall t \in \mathbb{N} \tag{4.34}$$

that satisfies the property

$$\boldsymbol{u}(t+1) = \boldsymbol{u}(t)\mathbf{\Gamma}. \tag{4.35}$$

The vector $\boldsymbol{u}(0)$ is referred as the *initial distribution* of the Markov Chain C, and is equal to the mixing probability vector $\boldsymbol{\delta}$ provided by the Finite Mixture Model built in the above sections:

$$\boldsymbol{u}(0) = \boldsymbol{\delta}. \tag{4.36}$$

A Markov Chain is said to be *stationary* if $\Pr(C_t|C_{t-1}) = \Pr(C_1|C_0) \quad \forall t = 1, 2, \ldots, T$ [75], or equivalently if it has a *stationary distribution* $\boldsymbol{u}(t) = \boldsymbol{u}(0) = \boldsymbol{\delta} = (\delta_1, \delta_2, \ldots, \delta_m) \quad \forall t \in \mathcal{N}$, whose elements, recall, are such that $\delta_i \geq 0 \quad \forall i = 1, 2, \ldots, m$ and $\sum_{i=1}^{m} \delta_i = 1$ (or, equivalently, $\boldsymbol{\delta}\mathbf{1}' = 1$ where $\mathbf{1}'$ is the transpose of the row unitary vector $\mathbf{1}$) and $\boldsymbol{\delta\Gamma} = \boldsymbol{\delta}$ [71]. The expression *stationary Markov Chain* is a consequence of the properties (4.35) of unconditional probabilities and of the stationary distribution $\boldsymbol{\delta}$: for every $t \geq 1$, the Markov Chain C continues to have the same distribution $\boldsymbol{\delta}$ that has at time $t = 0$. It can be proven by induction:

- for $t = 1$, $\boldsymbol{u}(1) = \boldsymbol{u}(0)\boldsymbol{\Gamma} = \boldsymbol{\delta\Gamma} = \boldsymbol{\delta}$ by definition of the stationary distribution;
- let us assume that the hypothesis holds for a given time t (i.e., that $\boldsymbol{u}(t) = \boldsymbol{\delta}$). Then, $\boldsymbol{u}(t+1) =$ by property (4.35), $\boldsymbol{u}(t)\boldsymbol{\Gamma} =$ by induction, $\boldsymbol{\delta\Gamma} =$ by definition of stationary distribution, $\boldsymbol{\delta}$.

A stationary Markov Chain, with stationary distribution $\boldsymbol{\delta}$ and transition probability matrix $\boldsymbol{\Gamma}$, can also be *reversible*, if

$$\delta_i \gamma_{ij} = \delta_j \gamma_{ji} \qquad \forall S_i, S_j \in S = \{S_1, S_2, \ldots, S_m\} \tag{4.37}$$

or, equivalently, if the probability to be in state S_i and to move to state S_j is the same to be in state S_j and to move to state S_i.

In case the Markov Chain is not stationary, it makes sense to talk about the so-called *initial probabilities* of states $S = \{S_1, S_2, \ldots, S_m\}$, which are the probability that, at time $t = 1$, the Markov Chain is in state S_i:

$$\pi_i = \Pr(C_0 = S_i) = u_i(0) = \delta_i \qquad \forall S_i \in S. \tag{4.38}$$

Initial probabilities are contained in the *initial probabilities vector*, or initial distribution, $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_m)$ and are provided by the Finite Mixture Model: each $\pi_i, \quad \forall i = 1, 2, \ldots, m$ is equal to δ_i, corresponding to the probability of the i^{th} component of the FMM: $\pi_i = \delta_i \quad \forall i = 1, 2, \ldots, m$. This is explained by the fact that the Markov Chain performs the transitions between clusters of observations $\boldsymbol{O} = (O_0, O_1, \ldots, O_T)$, that are found and defined by the FMM, which also provides their probabilities. Since FMMs do not depend over time, and that each state $S_i, \quad i = 1, 2, \ldots, m$ of the Markov Chain is associated to a component of the FMM, their mixing probabilities are the probabilities of states S_i of the Markov Chain at time $t = 0$. Thus, Equation (4.38) holds.

An interesting question to answer is the following [69]: given that the model is in a known state i at time $t = 0$, what is the probability to remain in state i for exactly d intervals of time?

$$\Pr(C_1 = S_i, C_2 = S_i, \ldots, C_d = S_i | C_1 = S_i) := p_i(d) = (\gamma_{ii})^{d-1}(1 - \gamma_{ii}), \tag{4.39}$$

which is the probability to remain in state i for $d - 1$ transitions, multiplied by the probability to change state at time $d + 1$. $p_i(d)$ is the discrete probability density function of duration d in state i. It follows that the expected number of times in which C is in a state i, given that $C_1 = i$, is

$$\mathrm{E}_i(d) = \sum_{d=1}^{+\infty} d p_i(d) = \sum_{d=1}^{+\infty} d(\gamma ii)^{d-1}(1 - \gamma_{ii}) = \frac{1}{1 - \gamma_{ii}}. \tag{4.40}$$

In order to be defined and described, a homogeneous Markov Chain thus needs:

- m states S_i, collected in $S = \{S_1, S_2, \dots, S_m\}$;
- the vector $\boldsymbol{\pi} = \boldsymbol{u}(0) = (\pi_1, \pi_2, \dots, \pi_m)$ of initial probabilities;
- the Transitions Probability Matrix, $\boldsymbol{\Gamma}$.

The first two are, as seen above, provided by Finite Mixture Models. It remains to find the transition probabilities. This is dealt in the following section 4.3.1.

4.3.1 How to estimate transition probabilities between states of a Markov Chain?

Let us suppose to have a Markov Chain C with m states in $\boldsymbol{S} = \{S_1, S_2, \dots, S_m\}$, and its realization $\boldsymbol{c} = (c_1, c_2, \dots, c_T)$. If its initial probability vector $\boldsymbol{\pi}$ and the transition probabilities matrix $\boldsymbol{\Gamma}$ are known, the probability of the realization $\boldsymbol{c}$ can be computed as follows [75]:

$$\Pr(C = \boldsymbol{c}) = \Pr(C_1 = c_1, C_2 = c_2, \dots, C_T = c_t) = \pi_{c_1} \prod_{t=2}^{T} \gamma_{c_{(t-1)} c_t}. \tag{4.41}$$

If $\boldsymbol{\Gamma}$ is unknown, estimates of its m^2 elements $\gamma_{ij}, \forall i, j = 1, \dots, m$ are given by [69]:

$$\widehat{\gamma}_{ij} = \frac{f_{ij}}{\sum_{k=1}^{m} f_{ik}} \qquad \forall i, j = 1, 2, \dots, m \tag{4.42}$$

where $f_{ij} \quad \forall i, j = 1, 2, \dots, m$ denotes the number of transitions observed in the realization $\boldsymbol{c} = (c_1, c_2, \dots, c_T)$ from state S_i to state S_j [71]. Expressions in Equation (4.42) are *empirical transition probabilities*, and follow from the fact that the likelihood of $\boldsymbol{c}$, conditioned on its first observation, is

$$L = \prod_{i=1}^{m} \prod_{j=1}^{m} \widehat{\gamma}_{ij}^{f_{ij}}. \tag{4.43}$$

Its log-likelihood is then

$$\begin{aligned} l = log(L) &= log\left(\prod_{i=1}^{m} \prod_{j=1}^{m} \widehat{\gamma}_{ij}^{f_{ij}} \right) \\ &= \sum_{i=1}^{m} \left(\sum_{j=1}^{m} f_{ij} log \widehat{\gamma}_{ij} \right) = \sum_{i=1}^{m} l_i. \end{aligned} \tag{4.44}$$

where we defined:

$$l_i := \sum_{j=1}^{m} f_{ij} log \widehat{\gamma}_{ij}. \tag{4.45}$$

In order to maximize the (log-)likelihood, one can then simply maximize each l_i separately. Note that $\widehat{\gamma}_{ii}$ can be expressed also as $\widehat{\gamma}_{ii} = 1 - \sum_{k \neq i} \widehat{\gamma}_{ik}$, reducing the numbers of parameters to estimate to $m^2 - m$. The partial derivative of l_i is

$$\frac{\partial l_i}{\partial \widehat{\gamma}_{in}} = \frac{f_{in}}{\widehat{\gamma}_{in}} + f_{ii} \left[\frac{-1}{1 - \sum_{k \neq i} \widehat{\gamma}_i k} \right] = \frac{f_{in}}{\widehat{\gamma}_{in}} - \left[\frac{f_{ii}}{1 - \sum_{k \neq i} \widehat{\gamma}_i k} \right] = \frac{f_{in}}{\widehat{\gamma}_{in}} - \frac{f_{ii}}{\widehat{\gamma}_{ii}} \tag{4.46}$$

that, if equated to 0, yields to

$$\frac{f_{in}}{\widehat{\gamma}_{in}} - \frac{f_{ii}}{\widehat{\gamma}_{ii}} = 0 \Rightarrow f_{in}\widehat{\gamma}_{ii} = f_{ii}\widehat{\gamma}_{in}. \tag{4.47}$$

When summing over all $j \leq m$, since $\sum_{n=1}^{m} \gamma_{in} = 1$ by definition of $\mathbf{\Gamma}$,

$$\sum_{n=1}^{m} f_{in}\widehat{\gamma}_{ii} = \widehat{\gamma}_{ii}\sum_{n=1}^{m} f_{in} = \widehat{\gamma}_{ii}\sum_{n=1}^{m} f_{in} = f_{ii}. \tag{4.48}$$

From the latter expression, it follows that:

$$\begin{aligned} \widehat{\gamma}_{ii} &= \frac{f_{ii}}{\sum_{j=1}^{m} f_{ij}} \\ \frac{\widehat{\gamma}_{ii}}{f_{ii}} &= \frac{1}{\sum_{n=1}^{m} f_{in}} \end{aligned} \tag{4.49}$$

A local maximum of the likelihood is then

$$\begin{aligned} \widehat{\gamma}_{ii} &= \frac{f_{ii}}{\sum_{j=1}^{m} f_{ij}} \\ \widehat{\gamma}_{ij} &= \frac{f_{ij}\widehat{\gamma}_{ii}}{f_{ii}} = \frac{f_{ij}}{\sum_{j=1}^{m} f_{ij}}. \end{aligned} \tag{4.50}$$

An equivalent proof of this can be found in [75], where the Lagrangian and the characteristic function are used. We now have all the tools needed to handle Hidden Markov Models, which are detailed in the following section.

4.4 Hidden Markov Models

A *Hidden Markov Model* (HMM) is a "doubly embedded stochastic process with an underlying stochastic process which is hidden, but that can be observed only through other stochastic processes that produce the sequence of observations" $\boldsymbol{O} = (O_0, O_1, \ldots, O_T)$ [69, 70]. A HMM has a series of observed and visible outputs $\boldsymbol{O}$, and a series of not observable states $\boldsymbol{c} = (c_1, c_2, \ldots, c_T)$, $c_j \in S = \{S_1, S_2, \ldots, S_m\}, \quad \forall j \leq T$. In other words, a HMM consists of two random variables, say C and O. The first one, C, is a Markov Chain (this class of models owes its name to this property) and is *hidden*, in the sense that the values it assumes are not known. The outputs of the latter, O, are observable and functions of the values of C [76]. A HMM is characterized by [69, 70]:

- m hidden states of a Markov Chain C, each with a peculiar significance, gathered in the set of states $S = \{S_1, S_2, \ldots, S_m\}$;

- for each state S_i, $i = 1, 2, \ldots, m$, N possible distinct outputs or symbols of the random variable O, collected in $V = \{v_1, v_2, \ldots, v_N\}$. Then, $O_t \in V \ \forall t \leq T$;

- the Transition Probability Matrix (TPM) of the Markov Chain C $\mathbf{\Gamma} = \{\gamma_{ij}\}_{i,j \leq m} \quad \forall i, j \in \{1, 2, \ldots, m\}$, where (recall) $\gamma_{ij} = \Pr(C_{t+1} = j | C_t = i)$;

- the matrix $\boldsymbol{B} = \{b_j(k)\}_{j \leq m, k \leq N}$, whose elements $b_j(k)$ are defined as the probability of the output v_k of the random variable O, known that the Markov Chain C at time t is in state S_i: $b_j(k) := \Pr(v_k | C_t = S_j)$, $\forall j \in \{1, 2, \ldots, m\}$, $k \in \{1, 2, \ldots, N\}$. It is assumed that this probability does depend nor on previous observations (i.e., values of $\boldsymbol{O}$) nor on the previous values of the Markov Chain C (*Output Independence Assumption* [75]). The elements of the matrix $\boldsymbol{B}$ are the probability that the hidden state S_j generates the output v_k [75];

- the vector of initial probabilities $\boldsymbol{\pi} = \big(\pi_1, \pi_2, \ldots, \pi_N\big)$, where $\pi_i := \Pr(C_0 = S_i) \forall i = 1, 2, \ldots, m$ are the so-called *initial probabilities* of the states of Markov Chain C defined in Equation (4.38);

A HMM is briefly indicated with

$$\boldsymbol{\lambda} = (\boldsymbol{\Gamma}, \boldsymbol{B}, \boldsymbol{\pi}). \tag{4.51}$$

A sequence of observations $\boldsymbol{O} = (O_0, O_1, \ldots, O_T)$ up to time T is generated as follows [69, 70]:

1. Choose $C_1 = S_i$, with $S_i \in S$ being selected according to the initial probabilities $\boldsymbol{\pi}$;

2. Set $t = 1$;

3. Choose $O_t = v_k$, according to probabilities $b_j(k)$;

4. Move to $C_{t+1} = S_j$ according to the transition probabilities γ_{ij}, $\forall j = 1, 2, \ldots, m$;

5. Set $t = t + 1$:

    - If $t < T$, then go back to step 3;
    - if $t = T$, then the sequence of observation is completed.

In other words, at each instant of time t (or, equivalently, at each step or clock time), C enters a new state, according to the transition probabilities γ_{ij}, say $C_t = S_j$, and an observation is outputted, say $O_t = v_l$, $l = 1, 2, \ldots, m$, according to the distribution $b_j(k), k \in \{1, 2, \ldots, m\}$ that depend only on the value of C at time t, $C_t = S_j$ [70]. It is important to correctly select the length T of the sequence $\boldsymbol{O}$, in order to be able to make reliable estimates of the parameters of the model.

This section provided a brief but exhaustive excursus over general features and components needed to build and define a HMM. The rest of the section proceeds as follows. Sub-section 4.4.1 shortly defines types of discrete Hidden Markov Models. Sub-section 4.4.2 answers the question “How to compare HMMs, in order to choose the one that best fits the collected observations?”. To build a HMM, three main problems need to be solved. This is the content of Sub-section 4.4.3. Algorithms and procedures capable to solve the three main problems here presented are detailed in Sub-section 4.4.4, followed by a Section 4.5 that concludes the chapter by providing its summary and explains briefly how it was used to obtain the results detailed in the following chapters.

4.4.1 Types of discrete Hidden Markov Models

Here, a short list of types of discrete Hidden Markov Models is presented.

In case each state can be reached in a single step from every other state (or, equivalently, if $\gamma_{ij} > 0 \ \forall i, j = 1, 2, \ldots, m$), the HMM is said to be *ergodic*, or *fully connected* [69]. In this case, the state transition matrix is *full*. This implies that any state will be revisited with probability equal to 1 [70], and such revisits do not need to take place at periodical intervals of time.

If, instead, it proceeds only "from left to right", i.e., $\pi_i = \begin{cases} 0 \text{ if } i \neq 1 \\ 1 \text{ if } i = 1 \end{cases}$ (i.e., the state sequence starts in state S_1) and $\gamma_{ij} = 0 \ \forall j < i$, the HMM is called *left-right model* or *Bakis model* [69]. In this case, its state transition matrix is upper triangular, and there is a kind of temporal order among the states: lower numbered states account for observations that occur for lower values of t [70]. There could be also additional constraints that avoid large state "jumps" or changes, expressed as $\gamma_{ij} = 0 \ \forall j > i + \Delta$. It easily follows that the last visited state needs to be S_m: $\gamma_{mm} = 1$ and $\gamma_{mi} = 0 \ \forall i < m$ [69]. An important issue is dealing with models comparison, in order to choose "a good one" to fit the observations. It is faced in the next Sub-section 4.4.2.

4.4.2 How to compare Hidden Markov Models?

Given two different Hidden Markov Models $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$, it can be reasonable to know if they are statistically equivalent. The answer is found by computing the following *distance measure*:

$$D(\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2) = \frac{1}{T}\Big[\log\Big(\Pr(\boldsymbol{O}^{(2)}|\boldsymbol{\lambda}_1)\Big) - \log\Big(\Pr(\boldsymbol{O}^{(2)}|\boldsymbol{\lambda}_2)\Big)\Big] \tag{4.52}$$

where $\mathbf{0}^{(2)} := (O_1^{(2)}, O_2^{(2)}, \dots, O_T^{(2)})$ is defined to be the sequence of observations generated by the model $\boldsymbol{\lambda}_2$. Since the measure D $D(\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2)$ is not symmetric, the following symmetric distance measure is often applied:

$$D_{symm}(\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2) = \frac{D(\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2) + D(\boldsymbol{\lambda}_2, \boldsymbol{\lambda}_1)}{2}. \tag{4.53}$$

The smaller $D_{symm}(\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2)$ is, the more statistically equivalent the models *boldsymbol*λ_1 and $\boldsymbol{\lambda}_2$ are.

Once we are able to compute the statistical difference between models, the last but not least issue to face is how to build a HMM.

4.4.3 Three main problems of HMMs

The above sections provided a range of definitions of Hidden Markov Models. More in detail, after showing how to formally define a HMM, Sub-section 4.4.1 provided a brief excursus over various kinds of discrete HMMs, whereas in Sub-section 4.4.2 one finds if two HMMs *boldsymbol*λ_1 and $\boldsymbol{\lambda}_2$ are statistically equivalent. Here, we deal with the last missing part about building a HMM, i.e., the three main problems related to them.

Evaluation Problem [69] Given the sequence of outputs $\boldsymbol{O} = (O_1, O_2, \dots, O_T)$ and the HMM $\boldsymbol{\lambda} = (\boldsymbol{\Gamma}, \boldsymbol{B}, \boldsymbol{\pi})$, how to efficiently compute the probability that the sequence of observations $\boldsymbol{O}$ has been produced by the model $\boldsymbol{\lambda}$? Or, equivalently, how well the model $\boldsymbol{\lambda}$ matches the observations $\boldsymbol{O}$? Both questions can be answered by computing the probability $\Pr(\boldsymbol{O}|\boldsymbol{\lambda})$. The second point of view gives the path to choose the HMM that best matches $\boldsymbol{O}$.

Estimation Problem [70] Given the sequence of outputs $\boldsymbol{O} = (O_1, O_2, \dots, O_T)$ and the HMM $\boldsymbol{\lambda} = (\boldsymbol{\Gamma}, \boldsymbol{B}, \boldsymbol{\pi})$, how to choose a corresponding state sequence $\boldsymbol{c} = (c_1, c_2, \dots, c_T)$, that is optimal in some meaningful sense and that best generates the sequence of observations $\boldsymbol{O}$? Since we are dealing with stochastic models, there does not exist a unique answer to the question. It can be answered using some optimality criterion.

Learning Problem [69] Given the sequence of outputs $\boldsymbol{O} = (O_1, O_2, \ldots, O_T)$ and the HMM $\boldsymbol{\lambda} = (\boldsymbol{\Gamma}, \boldsymbol{B}, \boldsymbol{\pi})$, how to adjust the parameters of $\boldsymbol{\lambda}$ (i.e., how to adjust $\boldsymbol{\Gamma}, \boldsymbol{B}$ and $\boldsymbol{\pi}$) to maximize the probability of $\boldsymbol{O}$, given the model? In other words, answering this question means maximizing $\Pr(O|\boldsymbol{\lambda})$.

The algorithms to solve them are detailed in Sub-section 4.4.4.

Solution to the Evaluation Problem

Let us enumerate all the possible state sequences of length T, $\boldsymbol{c} = (c_1, c_2, \ldots, c_T)$. Given one of such state sequences, then recalling that the searched probability is $\Pr(\boldsymbol{O}|\boldsymbol{\lambda})$ that does not depend on $\boldsymbol{c}$, to compute it we need to know $\Pr(\boldsymbol{O}|\boldsymbol{c}, \boldsymbol{\lambda})$, $\Pr(\boldsymbol{c}|\boldsymbol{\lambda})$ and $\Pr(\boldsymbol{O}, \boldsymbol{c}|\boldsymbol{\lambda})$.

$$\begin{aligned}
\Pr(\boldsymbol{O}|\boldsymbol{c}, \boldsymbol{\lambda}) &= \prod_{t=1}^{T} \Pr(O_t|c_t, \boldsymbol{\lambda}) = b_{c_1}(O_1) b_{c_2}(O_2) \ldots b_{c_T}(O_T) = \prod_{t=1}^{T} b_{c_t}(O_t) \\
\Pr(\boldsymbol{c}|\boldsymbol{\lambda}) &= \pi_{c_1} \gamma_{c_1 c_2} \gamma_{c_2 c_3} \cdots \gamma_{c_{T-1} c_T} = \pi_{c_1} \prod_{t=2}^{T} \gamma_{c_{t-1} c_t} \\
\Pr(\boldsymbol{O}, \boldsymbol{c}|\boldsymbol{\lambda}) &= \Pr(\boldsymbol{O}|\boldsymbol{c}, \boldsymbol{\lambda}) \Pr(\boldsymbol{c}|\boldsymbol{\lambda}) = \prod_{t=1}^{T} b_{c_t}(O_t) \pi_{c_1} \prod_{t=2}^{T} \gamma_{c_{t-1} c_t} \\
&= \pi_{c_1} b_{c_1}(O_1) \prod_{t=2}^{T} \gamma_{c_{t-1} c_t} b_{c_t}(O_t)
\end{aligned} \tag{4.54}$$

$\Pr(\boldsymbol{O}|\boldsymbol{\lambda})$ is then computed as follows:

$$\begin{aligned}
\Pr(\boldsymbol{O}|\boldsymbol{\lambda}) &= \sum_{\boldsymbol{c}} \Pr(\boldsymbol{O}, \boldsymbol{c}|\boldsymbol{\lambda}) = \sum_{\boldsymbol{c}} \Pr(\boldsymbol{O}|\boldsymbol{c}, \boldsymbol{\lambda}) \Pr(\boldsymbol{c}|\boldsymbol{\lambda}) \\
&= \sum_{\boldsymbol{c}} \left(\pi_{c_1} \prod_{t=2}^{T} \gamma_{c_{t-1} c_t} \prod_{t=1}^{T} b_{c_t}(O_t) \right) \\
&= \sum_{\boldsymbol{c}} \pi_{c_1} b_{c_1}(O_1) \gamma_{c_1 c_2} b_{c_2}(O_2) \ldots \gamma_{c_{T-1} c_T} b_{c_T}(O_T).
\end{aligned} \tag{4.55}$$

The computation of $\Pr(\boldsymbol{O}|\boldsymbol{\lambda})$ requires a number of calculations of the order $\mathcal{O}(2T \cdot N^T)$ (more in detail, $(2T-1)N^T$ multiplications and $N^T - 1$ additions), too high when T grows. To drastically reduce this number, one can use the *forward-backward algorithm.* To solve the evaluation problem only its forward part is needed. The backward part come into help to solve the Learning problem, detailed in Section 4.4.4.

Solution to the Estimation Problem

The Estimation problem tries to reveal the state sequence $\overline{\boldsymbol{c}}$, which is the hidden part of the model [70]. There are various ways to solve this problem, since there exist several optimality criteria that come into help to choose the optimal state sequence $\overline{\boldsymbol{c}} = (\overline{c}_1, \overline{c}_2, \ldots, \overline{c}_T)$. One solution is to choose the sequence $\overline{\boldsymbol{c}}$ of states that are individually most likely [69, 70]. Let's define:

$$\gamma_t(i) := \Pr(c_t = S_i|\boldsymbol{O}, \boldsymbol{\lambda}). \tag{4.56}$$

Then, the state $\overline{c}_t$ that, at time t is most likely is given by

$$\overline{c_t} := \text{argmax}[\gamma_t(i)] \qquad \forall t = 1, 2, \ldots, T. \tag{4.57}$$

It is important to note that, theoretically, the sequence obtained in this way could be not feasible according to the model. In other words, the resulting sequence of states $\overline{\boldsymbol{c}}$ could be impossible according to the transition probabilities of the model $\gamma_{ij}, \forall i, j \in \{1, 2, \ldots, m\}$. To check if $\overline{\boldsymbol{c}}$ is valid according to the transition probabilities matrix $\boldsymbol{\Gamma}$, there are various ways: one can look for the state sequence that maximizes the correct pair (c_t, c_{t+1}), or the correct triple (c_t, c_{t+1}, c_{t+2}) [69]. Nonetheless, the most widely used criterion consists in finding the single best state sequence with highest probability, i.e., to look for the state sequence that maximizes $\Pr(\boldsymbol{c}|\boldsymbol{O}, \boldsymbol{\lambda})$ or, equivalently, $\Pr(\boldsymbol{c}, \boldsymbol{O}|\boldsymbol{\lambda})$ [75]:

$$\begin{aligned} \overline{\boldsymbol{c}} &= \underset{\boldsymbol{c}}{\text{argmax}}\Big(\Pr(\boldsymbol{c}|\boldsymbol{O}, \boldsymbol{\lambda})\Big) = \underset{\boldsymbol{c}}{\text{argmax}}\left(\frac{\Pr(\boldsymbol{c}, \boldsymbol{O}|\boldsymbol{\lambda})}{\Pr(\boldsymbol{O}|\boldsymbol{\lambda})}\right) \\ &= \underset{\boldsymbol{c}}{\text{argmax}}\left(\frac{\Pr(\boldsymbol{c}, \boldsymbol{O}|\boldsymbol{\lambda})}{\sum_{\boldsymbol{c}} \Pr(\boldsymbol{c}, \boldsymbol{O})}\right) = \underset{\boldsymbol{c}}{\text{argmax}}\Big(\Pr(\boldsymbol{c}, \boldsymbol{O}|\boldsymbol{\lambda})\Big). \end{aligned} \tag{4.58}$$

The last equality holds because $\sum_{\boldsymbol{c}} \Pr(\boldsymbol{c}, \boldsymbol{O})$, being a sum all over the $\boldsymbol{c}$, does not depend on $\boldsymbol{c}$. The naive way to find $\overline{\boldsymbol{c}} = (\overline{c_1}, \overline{c_2}, \ldots, \overline{c_T})$ requires to try every single possible assignment to $\boldsymbol{c}$ and then to take the one with highest probability. It requires $\mathcal{O}(m^T)$ operations *only* to enumerate the set of all possible assignments, which is clearly unfeasible. For this reason, $\overline{\boldsymbol{c}}$ is computed by the *Viterbi Algorithm*, which differs from the forward procedure only because it tracks the maximum probabilities and records its corresponding state sequence. It is presented in detail in Section 4.4.4.

Solution to the Learning Problem

Solving the Learning problems is a bit tricky, since there is no analytical nor optimal way to estimate parameters of the model. One way is to choose them (i.e., to choose the model) such that the probability $\Pr(\boldsymbol{O}|\boldsymbol{\lambda})$ is locally maximized, using an iterative procedure. The mostly used is the *Baum-Welch method* (or, equivalently, the *EM – expectation-maximization – algorithm*), or gradient techniques [69, 70] (see Section 4.4.4).

4.4.4 Algorithms to solve the three problems for HMMs

Once the three main problems about building a HMM have been explained, detailed and solved, the last thing to do is to is to provide the algorithms mentioned in the previous Sub-section 4.4.3, to solve them more rapidly. This section explains in detail the algorithms mentioned above to solve the three problems related to HMMs.

Forward-Backward Algorithm

The Forward-Backward Algorithm comes into help to solve the Evaluation Problem in a feasible time. Note that only the forward part is used for the problem 1. The Backward one will be used to solve the Problem 3 [69]. Defined the so-called *forward variable* [69, 70] as

$$\alpha_t(i) := \Pr(O_0, O_1, \ldots, O_t, c_t = S_i|\boldsymbol{\lambda}), \tag{4.59}$$

the *Forward algorithm* proceeds as follows [69, 70].

Step 1: Initialization

$$\alpha_0(i) = \pi_i b_i(O_0) \quad \forall i = 1, 2, \ldots, m. \tag{4.60}$$

The variable $\alpha_0(i)$ is initiated with the joint probability to be in state S_i at time $t = 0$ and to observe the output O_1 [70].

Step 2: Induction

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{m} \alpha_t(i)\gamma_{ij}\right] b_j(O_{t+1}) \quad \forall t = 1, 2, \ldots, T-1; \forall j = 1, 2, \ldots, m. \tag{4.61}$$

At time $t+1$, state S_j is reached from one of the m possible states S_i reached at time $t, \forall i = 1, 2, \ldots, m$. $\alpha_t(i)$ is the probability that the first $t+1$ outputs $(O_0, O_1, \ldots, O_t)$ are observed and to be in state S_i at time t: $\alpha_t(i)\gamma_{ij}$ is then the probability to observe the sequence of length t of observations $(O_0, O_1, \ldots, O_t)$ and to reach the state S_j at time $t+1$ after being in state S_i at time t. The sum of $\alpha_t(i)\gamma_{ij}$ over all $i = 1, 2, \ldots, m$ gives then the probability to be in state S_j at time $t+1$ with the observation sequence, up to time t, given by $(O_1, O_2, \ldots, O_t)$. $\alpha_{t+1}(j)$, which is the probability to be in state S_j at time $t+1$ and to observe the output sequence, up to time $t+1$, (O_0, O_1, O_{t+1}) is then obtained by multiplying the previously mentioned sum with $b_j(O_{t+1})$ [70].

Step 3: Termination

$$\Pr(\boldsymbol{O}|\boldsymbol{\lambda}) = \sum_{i=1}^{m} \alpha_T(i) \tag{4.62}$$

since, by definition, $\alpha_T(i) = \Pr(O_1, O_2, \ldots, O_T, c_T = S_i|\boldsymbol{\lambda})$.

The calculation of $\alpha_t(j), \forall t = 0, 1, \ldots, T; \forall j = 1, 2, \ldots, m$ requires $\mathcal{O}(m^2T)$ calculations (more in detail, $m(m+1)(T-1)$ multiplications and $m(m-1)(T-1)$ additions are necessary).

The *Backward procedure* is below [69], once defined the *backward variable* to be the probability to observe $(O_{t+1}, O_{t+2}, \ldots, O_T)$ given that $C_t = S_i$ [70].

$$\beta_t(i) := \Pr(O_{t+1}, O_{t+2}, \ldots, O_T|c_t = S_i, \boldsymbol{\lambda}) \tag{4.63}$$

Step 1: Initiation

$$\beta_T(i) = 1 \qquad \forall i = 1, 2, \ldots, m. \tag{4.64}$$

Step 2: Induction

$$\beta_t(i) = \sum_{j=1}^{m} \gamma_{ij} b_j(O_{t+1})\beta_{t+1}(j) \quad \forall t = T-1, \ldots, 1, \forall i = 1, \ldots, m. \tag{4.65}$$

To be in state S_i at time t and to observe $(O_{t+1}, O_{t+2}, \ldots, O_T)$, it is required a transition to each of the m possible states at time $t+1$ from state S_i reached at time t. Then, one accounts for the observation sequence $(O_{t+1}, O_{t+2}, \ldots, O_T)$ [70].

The computation of $\beta_t(i)$, for each $t = 1, 2, \ldots, T$ and for each $i = 1, 2, \ldots, m$, requires $\mathcal{O}(m^2T)$ calculations.

The *forward* and *backward variables* can be applied to re-write $\gamma_t(i)$ [69, 70], defined in Equation (4.56):

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\Pr(\boldsymbol{O}|\boldsymbol{\lambda})} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{m} \alpha_t(j)\beta_t(j)} \tag{4.66}$$

since:

- $\alpha_t(i)$ accounts for the probability of the partial observation sequence $(O_0, O_1, \dots, O_t)$ and of $C_t = c_t = S_i$;
- $\beta_t(i)$ accounts for the probability of the remaining part of the observation sequence $(O_{t+1}, O_{t+2}, \dots, O_T)$ given that $C_t = c_t = S_i$;
- $\Pr(\boldsymbol{O}|\boldsymbol{\lambda})$ is a normalizing factor, such that $\sum_{i=1}^{m} \gamma_t(i) = 1 \ \forall t = 0, 1, \dots, T$.

Viterbi Algorithm

The Viterbi Algorithm allows to find a solution to the Estimation Problem, and proceeds as follows. Given an observation sequence $\boldsymbol{O} = (O_0, O_1, \dots, O_T)$, the goal is to find the "best" state sequence $\overline{\boldsymbol{c}} = (\overline{c_0}, \overline{c_1}, \dots, \overline{c_t})$ for the observed sequence of outputs $\boldsymbol{O} = (O_0, O_1, \dots, O_T)$. Defined the highest probability of a sequence of observations $(O_0, O_1, \dots, O_t)$ up to time t and a sequence of states up to time t that is in state S_i at time t [69, 70] as

$$\delta_t(i) := \underset{\boldsymbol{c}=(c_1,\dots,c_T)}{max} \Big(\Pr\big((c_1, c_2, \dots, c_t = S_i), (O_1, O_2, \dots, O_t) | \boldsymbol{\lambda} \big) \Big) \tag{4.67}$$

by induction we have

$$\delta_{t+1}(j) = \Big[\underset{i \in \{1,\dots,m\}}{max} \delta_t(i) \gamma_{ij} \Big] b_j(O_{t+1}). \tag{4.68}$$

In order to find the optimal state sequence, one needs to keep track of the argument that maximizes $\delta_{t+1}(j), \forall j = 1, 2, \dots m$ and $\forall t = 1, 2, \dots, T$. This task is accomplished by the *Viterbi Algorithm* [69, 70]:

Step 1: Initialization

$$\begin{aligned} \delta_0(t) = \pi_i b_i(O_1) \qquad \forall i = 1, 2, \dots, m \\ \Psi_0(i) = 0 \qquad \forall i = 1, 2, \dots, m; \end{aligned} \tag{4.69}$$

Step 2: Induction

$$\begin{aligned} \delta_t(i) &= \Big[\underset{1 \le i \le m}{max} \, \delta_{t-1}(i) \gamma_{ij} \Big] b_j(O_t) \qquad \forall t = 1, 2, \dots, T, \quad \forall j = 1, 2, \dots, m \\ \Psi_t(j) &= \underset{1 \le i \le m}{argmax} \big[\delta_{t-1}(i) \gamma_{ij} \big] \qquad \forall t = 1, 2, \dots, T, \quad \forall j = 1, 2, \dots, m \end{aligned} \tag{4.70}$$

Step 3: Termination

$$\begin{aligned} \Pr{}^* &= \underset{1 \le i \le m}{max} \Big[\delta_T(i) \Big] \\ c_T^* &= \underset{1 \le i \le m}{argmax} \Big[\delta_T(i) \Big] \end{aligned} \tag{4.71}$$

Step 4: Path (state sequence) backtracking

$$c_t^* = \Psi_{t+1}(c_{t+1}^*) \qquad \forall t = T-1, T-2, \dots, 0 \tag{4.72}$$

The Viterbi Algorithm is, w.r.t. implementation, similar to the forward-backward algorithm (without the back-tracking steps): the sum is substituted by a maximization over states [70].

Baum - Welch Algorithm

Used to solve the Learning Problem, the *Baum - Welch Algorithm* proceeds as follows. Let us define the probability that the state sequence $\boldsymbol{c}$ is in state S_i at time t (i.e., $C_t = c_t = S_i$) and in state S_j at time $t+1$ (i.e., $C_{t+1} = c_{t+1} = S_j$) given $\boldsymbol{O}$ and $\boldsymbol{\lambda}$ as follows:

$$\xi_t(i,j) := \Pr(c_t = S_i, c_{t+1} = S_j | \boldsymbol{O}, \boldsymbol{\lambda}), \qquad i,j = 1,2,\dots,m \quad \forall t \leq T. \tag{4.73}$$

Then, $\gamma_t(i)$, that we recall to be the probability to be in state S_i at time t given the model and the sequence of observations defined in Equation (4.56), can be expressed as follows:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j). \tag{4.74}$$

The expected number of transition from state S_i, or equivalently the expected number of times that state S_i is visited, is expressed by

$$\sum_{t=1}^{T} \gamma_t(i) \tag{4.75}$$

whereas the expected number of transitions from state S_i to state S_j by

$$\sum_{t=1}^{T} \xi_t(i,j). \tag{4.76}$$

Using these expressions, a set of reasonable re-estimation formulas for the parameters $\boldsymbol{\pi}, \boldsymbol{\Gamma}$ and $\boldsymbol{B}$ are given by the *Baum-Welch re-estimation formulas* [69, 70]:

$$\begin{aligned}
\overline{\pi_i} &= \text{expected frequency (i.e., number of times) in state } S_i\text{Ramage2007HMMFundamentals at time } t = 0 \\
&= \gamma_0(i) \\
\overline{\gamma_{ij}} &= \frac{\text{expected number of transitions from } S_i \text{ to } S_j}{\text{expected number of transitions from } S_i} \\
&= \frac{\sum_{t=0}^{T-1} \xi_t(i,j)}{\sum_{t=0}^{T-1} \gamma_t(i)} \\
\overline{b_j(k)} &= \frac{\text{expected number of times in } S_j \text{ and observing } v_k}{\text{expected number of times in } S_j} \\
&= \frac{\sum_{\substack{t=0 \\ O_t = v_k}}^{T} \gamma_t(j)}{\sum_{t=0}^{T} \gamma_t(j)}.
\end{aligned} \tag{4.77}$$

A proof of how they are obtained can be found in [75]. Indicated with $\boldsymbol{\lambda} = (\boldsymbol{\Gamma}, \boldsymbol{B}, \boldsymbol{\pi})$ the initial model (used to compute the above formulas), and with $\overline{\boldsymbol{\lambda}} = (\overline{\boldsymbol{\Gamma}}, \overline{\boldsymbol{B}}, \overline{\boldsymbol{\pi}})$ the re-estimated one, it can be proven that either $\boldsymbol{\lambda}$ is a critical point for the likelihood function, which leads to $\boldsymbol{\lambda} = \overline{\boldsymbol{\lambda}}$, or $\Pr(\boldsymbol{O}|\overline{\boldsymbol{\lambda}}) > \Pr(\boldsymbol{O}|\boldsymbol{\lambda})$ [69] as follows. Defined the *Kullback-Leibler divergence* of two distributions p_1 and p_2 [76],

$$D(p_1, p_2) := \sum_{\omega} p_1(\omega) \log\left(\frac{p_1(\omega)}{p_2(\omega)}\right) = \mathrm{E}_1\left[\log\left(\frac{p_1(\omega)}{p_2(\omega)}\right)\right], \tag{4.78}$$

then $D(p_1, p_2) \geq 0$, and $D(p_1, p_2) = 0$ if and only if $p_2(\omega) = p_1(\omega) \forall \omega$ such that $p_1(\omega) > 0$. Now, defined the two distributions p_1 and p_2 as:

$$\begin{aligned} p_1(\boldsymbol{c}) &:= \frac{p(\boldsymbol{s}, \boldsymbol{o}, \boldsymbol{\lambda})}{p(\boldsymbol{o}, \boldsymbol{\lambda})} \\ p_2(\boldsymbol{c}) &:= \frac{p(\boldsymbol{s}, \boldsymbol{o}, \boldsymbol{\lambda}')}{p(\boldsymbol{o}, \boldsymbol{\lambda}')}, \end{aligned} \tag{4.79}$$

it follows that

$$\begin{aligned} 0 \leq D(\boldsymbol{\lambda}, \boldsymbol{\lambda}') &= \sum_{\boldsymbol{c}} p_1(\boldsymbol{c}) \log\left(\frac{p_1(\boldsymbol{c})}{p_2(\boldsymbol{c})}\right) = \sum_{\boldsymbol{c}} \frac{p(\boldsymbol{s}, \boldsymbol{o}, \boldsymbol{\lambda})}{p(\boldsymbol{o}, \boldsymbol{\lambda})} \log\left(\frac{\frac{p(\boldsymbol{s}, \boldsymbol{o}, \boldsymbol{\lambda})}{p(\boldsymbol{o}, \boldsymbol{\lambda})}}{\frac{p(\boldsymbol{s}, \boldsymbol{o}, \boldsymbol{\lambda}')}{p(\boldsymbol{o}, \boldsymbol{\lambda}')}}\right) \\ &= \sum_{\boldsymbol{c}} \frac{p(\boldsymbol{s}, \boldsymbol{o}, \boldsymbol{\lambda})}{p(\boldsymbol{o}, \boldsymbol{\lambda})} \log\left(\frac{p(\boldsymbol{s}, \boldsymbol{o}, \boldsymbol{\lambda}) p(\boldsymbol{o}, \boldsymbol{\lambda}')}{p(\boldsymbol{s}, \boldsymbol{o}, \boldsymbol{\lambda}') p(\boldsymbol{o}, \boldsymbol{\lambda})}\right) \\ &= \log\left(\frac{p(\boldsymbol{o}, \boldsymbol{\lambda}')}{p(\boldsymbol{o}, \boldsymbol{\lambda})}\right) + \sum_{\boldsymbol{c}} \frac{p(\boldsymbol{s}, \boldsymbol{o}, \boldsymbol{\lambda})}{p(\boldsymbol{o}, \boldsymbol{\lambda})} \log\left(\frac{p(\boldsymbol{s}, \boldsymbol{o}, \boldsymbol{\lambda})}{p(\boldsymbol{s}, \boldsymbol{o}, \boldsymbol{\lambda}')}\right) \\ &= \log\left(\frac{p(\boldsymbol{o}, \boldsymbol{\lambda}')}{p(\boldsymbol{o}, \boldsymbol{\lambda})}\right) + \sum_{\boldsymbol{c}} \frac{\left[p(\boldsymbol{s}, \boldsymbol{o}, \boldsymbol{\lambda}) \log p(\boldsymbol{s}, \boldsymbol{o}, \boldsymbol{\lambda}) - p(\boldsymbol{s}, \boldsymbol{o}, \boldsymbol{\lambda}) \log p(\boldsymbol{s}, \boldsymbol{o}, \boldsymbol{\lambda}')\right]}{p(\boldsymbol{o}, \boldsymbol{\lambda})} \end{aligned} \tag{4.80}$$

If one defines

$$Q(\boldsymbol{\lambda}, \boldsymbol{\lambda}') := \sum_{\boldsymbol{c}} p(\boldsymbol{s}, \boldsymbol{o}, \boldsymbol{\lambda}) \log\Big(p(\boldsymbol{s}, \boldsymbol{o}, \boldsymbol{\lambda}')\Big), \tag{4.81}$$

then the previous expression can be rewritten as

$$= \log\left(\frac{p(\boldsymbol{o}, \boldsymbol{\lambda}')}{p(\boldsymbol{o}, \boldsymbol{\lambda})}\right) + \frac{Q(\boldsymbol{\lambda}, \boldsymbol{\lambda}) - Q(\boldsymbol{\lambda}, \boldsymbol{\lambda}')}{p(\boldsymbol{o}, \boldsymbol{\lambda})} \geq 0. \tag{4.82}$$

Then,

$$\log\left(\frac{p(\boldsymbol{o}, \boldsymbol{\lambda}')}{p(\boldsymbol{o}, \boldsymbol{\lambda})}\right) \geq \frac{Q(\boldsymbol{\lambda}, \boldsymbol{\lambda}') - Q(\boldsymbol{\lambda}, \boldsymbol{\lambda})}{p(\boldsymbol{o}, \boldsymbol{\lambda})}, \tag{4.83}$$

which implies that if $Q(\boldsymbol{\lambda}, \boldsymbol{\lambda}') - Q(\boldsymbol{\lambda}, \boldsymbol{\lambda}) \geq 0$ (or, equivalently, if $Q(\boldsymbol{\lambda}, \boldsymbol{\lambda}') \geq Q(\boldsymbol{\lambda}, \boldsymbol{\lambda})$), then $p(\boldsymbol{o}, \boldsymbol{\lambda}') \geq p(\boldsymbol{o}, \boldsymbol{\lambda})$. In order to maximize $Q(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$, it suffices to find models such that $\frac{\partial Q(\boldsymbol{\lambda}, \boldsymbol{\lambda}')}{\partial \boldsymbol{\lambda}'} = 0$ and whose parameters satisfy the aforementioned stochastic constraints.

It is thus sufficient to iteratively use parameters of $\overline{\boldsymbol{\lambda}}$ instead of the ones of the model $\boldsymbol{\lambda}$ in the above formulas and repeat the re-estimation calculation, in order to let $\Pr(\boldsymbol{O}|\overline{\boldsymbol{\lambda}})$ increase until an upper bound is reached. The estimated parameters of the "last" model $\overline{\boldsymbol{\lambda}}$ are called (local) *maximum likelihood estimates of the HMM* (the Baum-Welch procedure leads only to local maxima [69]). The stochastic constraints of the parameters of the model are automatically satisfied at each iteration:

$$\begin{aligned} &\sum_{i=1}^{m} \overline{\pi_i} = 1 \\ &\sum_{j=1}^{m} \overline{\gamma_{ij}} = 1 \qquad \forall i = 1, 2, \ldots, m \\ &\sum_{k=1}^{N} \overline{b_j(k)} = 1 \qquad \forall j = 1, 2, \ldots, m. \end{aligned} \tag{4.84}$$

Re-estimation formulas provide parameters that correspond only to a local maximum of the likelihood function [69]. Initial estimates of the parameters can be properly chosen as follows so that the global maximum of the likelihood function is found:

- $\boldsymbol{\Gamma}$ and $\boldsymbol{\pi}$ can be randomly or uniformly chosen;
- $\boldsymbol{B}$ needs to be chosen carefully, especially in the continuous case.

The re-estimation formulas (4.77) require a number of operations to be computed that becomes unfeasible when T grows. More precisely, it is the case for the variable $\xi_t(i,j)$ $\forall i,j = 1,2,\dots,m$ and $t \leq T$ defined in (4.73), on which variables 4.74 and re-estimation formulas 4.77 are based. To compute $\xi_t(i,j)$ in a feasible time, one can exploit the *Forward - Backward algorithm*. More in detail, $\xi_t(i,j) := \Pr(c_t = S_i, c_{t+1} = S_j|\boldsymbol{O},\boldsymbol{\lambda})$ can be rewritten in terms of the forward $\alpha_t(i)$ and backward $\beta_t(i)$ variables as

$$\xi_t(i,j) = \frac{\alpha_t(i)\gamma_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\Pr(\boldsymbol{O}|\boldsymbol{\lambda})}, \tag{4.85}$$

since:

- $\alpha_t(i)$ accounts for the first $t+1$ observations $(O_0, O_1, \dots, O_t)$ and to be in state $C_t = S_i$ at time t;
- $\gamma_{ij}b_j(O_{t+1})$ accounts for the transition from the state S_i, reached at time t, to state S_j, reached at time $t+1$, and to observe O_{t+1};
- $\beta_{t+1}(j)$ accounts for the remainder of the observed sequence of outputs;
- $[\Pr(\boldsymbol{O}|\boldsymbol{\lambda})]^{-1}$ is the normalization factor.

4.5 Summary and Conclusions

The chapter here presented provided a comprehensive guide of all the statistical tools, algorithms and models that will be used in Chapters 5 and 6, to model Network Scanners. ore precisely, they will be modeled from the point of view of the attacked network/system, for reasons that are detailed in the next pages. Network Scanners will be modeled by exploiting informations contained in logs of received packets: timestamp, targeted port, destination IP address and source IP address. This information is leveraged to obtain the data that are used for our work. Chapter 5 considers, as data, counters of probes received by the network within intervals of 10 minutes. This, because it aims at modeling intensities of Network Scanners. Chapter 6 instead exploits differences of consecutively probed IP addresses and of timestamps of consecutive probes, respectively. Finite Mixture Models (FMMs) are leveraged to group the considered data into clusters, and Hidden Markov Models (HMMs) compute then the probabilities to move from a cluster to another. To model intensities of Network Scanners in Chapter 5, the considered datasets contained data collected over long time windows. This allowed us to investigate macroscopic behaviors of Network Scanners. Chapter 6 deals with their "microscopic" behaviors, that is, their single executions. The goal is twofold. Firstly, we aimed at modeling single executions of the two considered Network Scanners. Secondly, we used the obtained models to recognize Network Scanners that originate freshly collected test samples.

The obtained models and their analyses are detailed in the following two chapters.

5

Long Term Analyses of ZMap and Shodan

Contents

5.1 Introduction

Chapter 4 provided the required statistical modeling tools, models and algorithms needed to build models of logs of packets originated by a Network Scanner (NS) and received by a network. Network Scanners have been described in Chapter 2, whereas Chapter 3 reviews their existing modeling techniques. This chapter aims to build statistical models of Network Scanners (NSs) from the point of view of the targeted network. This choice is motivated by various facts. First of all, to the best of our knowledge, the actual State of the Art is poor of models built with this perspective, which is motivated by the fact that scanned networks do not know a priori the Network Scanner deployed to perform the Network Scanning Activity (NSA), that is targeting the system. Recall that a *Network Scanning Activity* is defined to be a “scanning campaign”, which could apply multiple executions of one or multiple Network Scanners (NSs), or also a single execution of a single Network Scanner (cite chapter after writing). A *Network Scanner* (NS) is instead a tool developed and applied to scan the IPv4 space, all or part of it. Our work focused on building and validate stochastic models of intensities of Network Scanners, that could be leveraged by security experts and researchers to detect of Network Scanning Activities.

There exist various types of Network Scanners. Some of them, like ZMap [20], are open source. This means that their algorithm, codes, etc., are publicly available. One could object that a model for open source scanners is useless, since their operation is to everyone's knowledge. Despite it is true, its features (speed, intensity, i.e., number of probes per minute, . . .) can be adapted by the performer(s) of the scanner, and can depend by their parameters that need to be chosen and fixed before executing the Network Scanner itself. Furthermore, ZMap being open source, its code, performances and behavior can be changed drastically by the performer of the scanner. Other Network Scanners, like Shodan [25], have instead a black-box approach, meaning that their mode of operation is not publicly available.

For these reasons, we chose to build stochastic models of two large scale Network Scanners: one open source, ZMap [20], which is a fast open source horizontal Network Scanner capable to scan the whole IPv4 space over one single service in less that 45 minutes [20], and one with a black-box approach, Shodan, which is a block scanner. Let us note that the methodology we proposed and detailed in Chapter 4 and will apply in Chapters 5 and 6 does not exploit "outer knowledge" of ZMap: for both scanners, the only data we leveraged are the information collected by the hosted network. More precisely, their traces have been collected from a /20 darknet [77] hosted in the High Security Lab (LHS) at Inria Nancy - Grand Est[5].

The goal of this chapter is to present Finite Poisson Mixture Models and the corresponding Hidden Markov Models to model intensities of the two considered Network Scanners, i.e., the counts[6] of IP addresses targeted in a given and fixed interval of time. We also investigated if intensities vary with respect to the probed service, in order to assess if they are a macroscopic feature of the considered Network Scanners that suffices to recognize it from logs of packets received by the considered port.

The remainder of this chapter is organized as follows. Section 5.2 describes a darknet and motivates the choice to leverage it to collect logs of the two considered Network Scanners. Section 5.3 details the reasons that guided us to choose ZMap and Shodan as the two scanners to model. Section 5.4 details the collected datasets used to infer the models. Sections 5.5 and 5.6, after providing a more detailed definition of "intensity" of Network Scanners, explain details of the obtained Finite Mixture Models and Hidden Markov Models. Section 5.8 concludes the chapter.

5.2 Darknet: definitions and usages

The term "darknet" can refer to various definitions [77]:

- any communication system that steals information and data and conceals the identity of its user;
- servers and programs that are used to illegally distribute copy-righted material;
- servers configured to collect suspicious data, running in passive mode without any interaction with attackers.

Throughout our work, we refer to the last definition of the term. A darknet is a network telescope to observe activities over the IPv4 space and cyber attacks using passive monitoring [77]. In practice, it is a collection of IP addresses that are routable and allocated, but unused [8]. Since the IP addresses of a darknet are unused, they are new hosts that never communicated with other devices. It follows that any observed traffic received by these IP addresses is suspicious and deserves future investigation [77]. A darknet is also known with various other terms

[5] `http://www.lhs.loria.fr`

[6] The reason why the term "count" is chosen, is that here the number we have is the number of probes received by the darknet within 10 minutes. In other words, it is the final count of probes received

[77, 8]: Internet sink, network telescope, unused IP addresses, darkspace, blackhole monitors, suspicious traffic, Internet background radiation, non-responsive traffic, non productive traffic. A darknet consists of network sensors, implemented and dispersed on strategic points throughout the Internet, hosted by Internet Service Providers (ISPs), research institutes, universities, backbone networks, etc. A darknet thus represents a partial view of the entire IPv4 space, which cannot be distinguished from the rest of it. It can be (and it has been) used:

- to gather information and extract insights on [77]:
    - scanning (or probing) activities;
    - misconfiguration;
    - political events;
- have insights into malicious activities that target the IPv4 space, or part of it [78];
- identify potential attackers, who sent packets to the darknet [78].

Research fields that actually exploit darknets are deployment, *traffic analysis* and visualization. The initial part of our work lies in the second category. More in detail, we collected data from a /20 darknet (i.e., 4096 IP addresses) hosted in the High Security Lab at Inria Nancy - Grand Est, which has been active since November 2014. At that time, it was collecting about 4 millions packets each day, whereas nowadays this number has almost tripled (it is reached by up to 11 millions packets per day). The choice to use this traffic was motivated by the fact that all the traffic received by a darknet is, as said before, unsolicited .

Once the origin of the data is set, we needed to choose the Network Scanners to study. The choice fell into ZMap and Shodan, for the reason explained in the next section. After it, Section 5.4 provides a detailed description of the used datasets, and shows how they have been extracted from the amount of packets received by our darknet.

5.3 ZMap and Shodan

As stated above, this chapter aims at modeling intensities of Internet-wide Network Scanners. We chose to study Shodan and ZMap, respectively, guided by their main and substantial differences. As mentioned before, Shodan is a block scanner with a black-box approach [25]. Much more information is available about ZMap [20], an horizontal open source Network Scanner developed by the University of Michigan that continuously launches it. It is able to scan the entire IPv4 space in less than 45 minutes with a single machine at a maximum speed of gigabit Ethernet.

ZMap proceeds as follows. First of all, it creates an order of the IP addresses to target through a random permutation generated by a cyclic multiplicative group. After that, probes are sent to the targets. Note that a fixed number of probes per target are sent (by default, one [20]). It then accepts response packets with the correct state field for the duration of the scan.

ZMap consists of a *scanner core*, a *probe modules* and *output handlers*:

- *Scanner code*: it organizes and manages
    - parsing of configuration files and command lines;
    - generation and exclusion of IP addresses;
    - monitoring of progress and performance;
    - reading and writing network packets.

- *Probe modules*: they are extensible, and can be customized. They:
  - generate probe packets;
  - interpret whether incoming packets are valid responses or not.
- *Output handlers*: they allow the results of the scan procedure to be:
  - piped to other processes;
  - added to a database;
  - transmitted to user code for further actions.

The sending and reception of packets act independently and continuously through the scan.

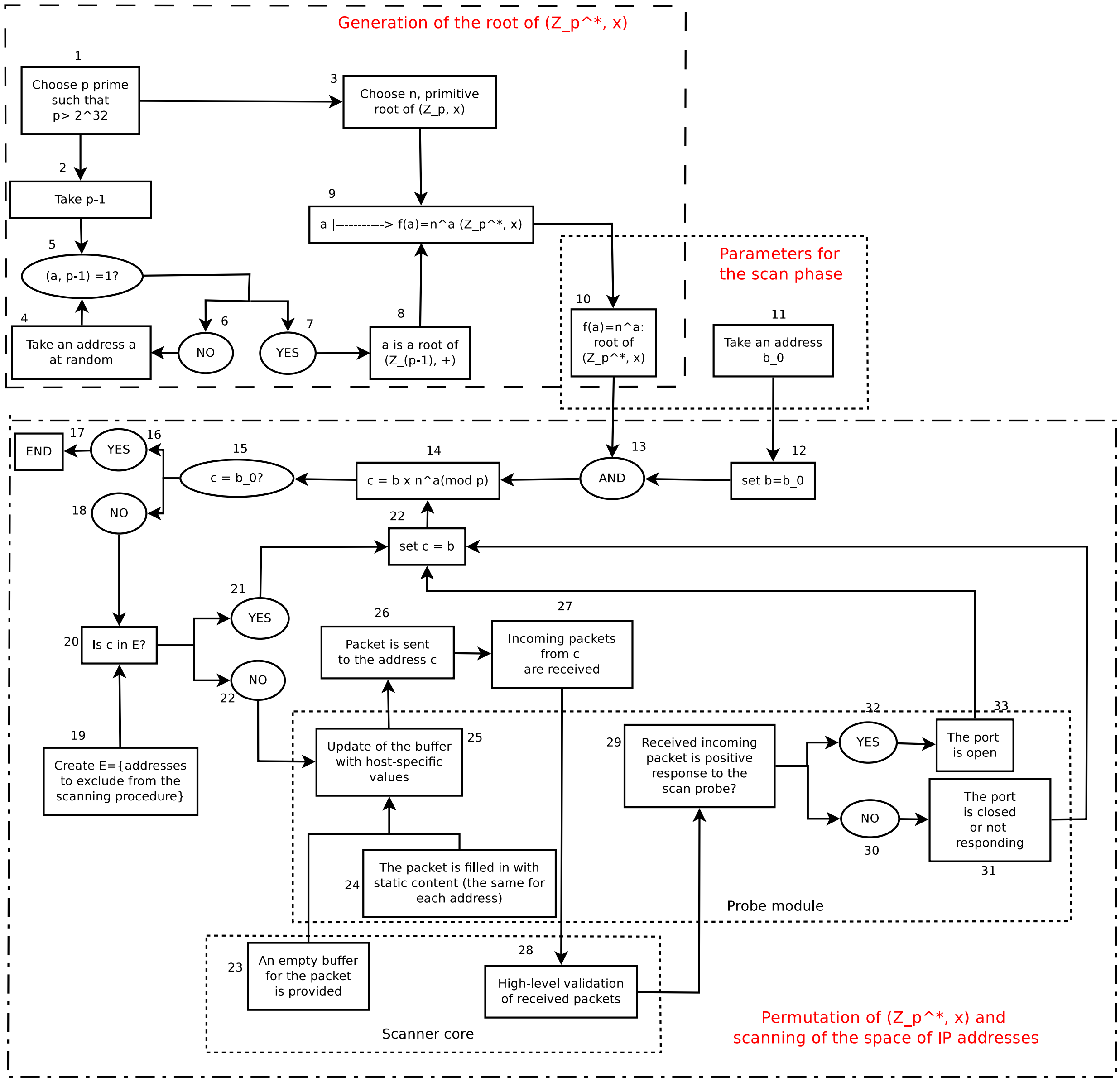

Generation of the root of (Z_p^*, x)
1
Choose p prime such that p> 2^32
3
Choose n, primitive root of (Z_p, x)
2
Take p-1
9
a |----------> f(a)=n^a (Z_p^*, x)
5
(a, p-1) =1?
6
7
4
Take an address a at random
NO
YES
8
a is a root of (Z_(p-1), +)
Parameters for the scan phase
10
f(a)=n^a: root of (Z_p^*, x)
11
Take an address b_0
17
16
END
YES
15
c = b_0?
14
c = b x n^a(mod p)
13
AND
12
set b=b_0
18
NO
22
set c = b
21
YES
20
Is c in E?
NO
22
26
Packet is sent to the address c
27
Incoming packets from c are received
19
Create E={addresses to exclude from the scanning procedure}
25
Update of the buffer with host-specific values
29
Received incoming packet is positive response to the scan probe?
32
YES
33
The port is open
NO
30
The port is closed or not responding
31
24
The packet is filled in with static content (the same for each address)
Probe module
23
An empty buffer for the packet is provided
28
High-level validation of received packets
Scanner core
Permutation of (Z_p^*, x) and scanning of the space of IP addresses

Figure 5.1: Detailed Steps of the Algorithm of ZMap

As previously mentioned, ZMap scans IP addresses according to their random permutation, generated by a cyclic multiplicative group. Thus, the whole procedure, shown in Figure 5.1, can be divided in three phases:

1. finding the root of the multiplicative group;
2. permutation of the space of IP addresses;
3. scanning the IPv4 space;

which are detailed in the following sub-sections. More precisely, Sub-section 5.3.1 shows how ZMap properly selects a cyclic multiplicative group and chooses its root, whereas Sub-section 5.3.2 details how it permutes and scans the IPv4 space. In other words, each of them deals with one of the "boxes" in Figure 5.1.

5.3.1 How to choose a proper multiplicative group and how to select its root

In order to select a random permutation for each scanning of the IPv4 space, a *new primitive root* of a cyclic multiplicative group and a *random starting address* are required. They are two values (parameters) that are chosen (or fixed) by the launcher of ZMap, and their values, once fixed, could interfere with its intensity. Let us see here, firstly, how ZMap properly selects a cyclic multiplicative group, and then how it chooses its root. Sub-section 5.3.2 will then deal with detailing how it permutes and scans the Internet.

To find a proper multiplicative group, ZMap simply needs its first parameter, a *prime number* p *greater than* 2^{32}, which is the total number of IP addresses contained in the IPv4 space. The reason why a prime number is chosen lies in the fact that it guarantees the cyclic nature of the corresponding multiplicative group, and that all the addresses in the IPv4 space will be reached once per cycle (except 0.0.0.0, which is an IANA reserved address). Authors of [20] iterated over $(\mathbb{Z}_{4294967311}, \times)$ which is the multiplicative group modulo p for the smallest prime larger than 2^{32}: $2^{32} + 15$. The multiplicative group $(\mathbb{Z}_p, \times)$ is then built and *one of its primitive roots* n is chosen (3), which is the second parameter required by ZMap. Authors of [20] set $n = 3$.

Let us now keep looking at the Figure 5.1, and consider $p-1$ (2). It is known that generators of the additive group $(\mathbb{Z}_{p-1}, +)$ are numbers s prime with $p-1$, i.e. elements of the set $\{s|(s, p-1) = 1\}$: it is thus possible to find generators of the additive group $(\mathbb{Z}_{p-1}, +)$ by:

- calculating and storing the factorization of $p - 1$;
- checking if randomly selected addresses a (4) are prime with $p - 1$, i.e. if $(a, p - 1) = 1$ (5). If the selected address a is *not* prime with $p - 1$, (6), then another address a is chosen (4). One remains in the loop until the randomly chosen address a is prime with $p - 1$ (7).

In this way, when the loop terminates, the root a of the group $(\mathbb{Z}_{p-1}, +)$ is found (8). It is then mapped to n^a (9), contained in the multiplicative group $(\mathbb{Z}_p^*, \times)$, which is in turn a root of $(\mathbb{Z}_p^*, \times)$ (10) (whereas $\mathbb{Z}_p^* = \mathbb{Z}_p - \{0\}$).

This part of the algorithm of ZMap is in the upper box in Figure 5.1.

5.3.2 Permutation and Scanning of the IPv4 Space

This phase takes, as parameters, the root n^a of the multiplicative group $(\mathbb{Z}_p^*, \times)$ outputted by the previous part of the algorithm and detailed in Sub-section 5.3.1, and an *initial random address*

b_0 (see point (11) in Figure 1), which is the third and last parameter required by ZMap and that needs to be fixed by the executer.

b is initialized as $b = b_0$ (12), and $c = b \times n^a (mod\ p)$ is calculated (14). A check if $c = b_0$ (15) is done: if $c = b_0$ (16), the algorithm is finished, which means that the space has been fully permuted and scanned (17). If instead $c \neq b_0$ (18), then one checks if the considered address c is in the set E (20) of elements of $(\mathbb{Z}_p^*, \times)$ which shouldn't be scanned (19). Note that p is greater than 2^{32}: this means that $(\mathbb{Z}_p^*, \times)$ contains more elements than the whole IPv4 address space (in [20], these exceeding elements are 15, because p has been chosen to be $p = 2^{32} + 15$), which need to be excluded from the scanning operation and thus are elements of E. It also contains, for example:

- IANA reserved allocations;
- addresses whose owners asked their removal from the scanning procedure [20].

If $c \in E$ (21), then it is not scanned, is set equal to b (22) and $c = b \times n^a (mod\, p)$ (14) is computed again. If instead $c \notin E$ (22), the following scenario discloses.

The scanner core provides an empty buffer for each packet (23) to send as probe, and the probe module fills it in with static content (24). Note that these two steps are done only once, since their result is the "basis" of the probes which are sent to the addresses to scan. This "basis" is then updated by the *probe module* with host-specific values (25), and the packet is then sent to the address c (26). The default IP-id value of packets is 54321. Clearly, it can be used *only* to detect packets generated by ZMap executed by the University of Michigan, since other users can change it as they want. If any, incoming packets from c are then received (27) and the scanner code validates them (28). The probe module then checks if the received packet is a positive response to the scan probe (29): if not (30), the port is closed or not responding (31). If yes (32), the port is open (33). In both cases, the algorithm continues with the generation of the next address of the permutation and its scanning. It means that $c = b$ (22) is set again, and $c = b \times n^a (mod\ p)$ (14) is computed.

The algorithm is shown below (same problems as above).

5.4 Datasets

Throughout Section 5.2 we explained and motivated the choice to use darknet traffic to build models of Network Scanners from the point of view of the targeted system. Section 5.1 motivated instead the choice of the two selected Network Scanners, Shodan and ZMap, whose features have been clarified in Section 5.3. The question that arise now is "how to distinguish scanning traffic, and more precisely traffic originated by Shodan and ZMap respectively, from all the rest of it?". This issue is faced and solved by knowing the sets of IP addresses that perform each considered scanner. So, we filtered packets coming from IP addresses reserved to execute Shodan, which are publicly declared, and ZMap, which is continuously executed by the University of Michigan that reserves part of its network for this task. For each probe, its time-stamp, source IP address, destination IP address and targeted port are available. Examples of logs of packets received by the hosted darknet, generated by ZMap and targeting port 23 are shown in Table 5.1, where the destination IP addresses have been anonymized to protect the secrecy of the darknet.

The goal of the work presented in this chapter is to build statistical models for the *intensity* of ZMap and Shodan. The intensity of a Network Scanner is defined as the "speed" with which their packets are received by the hosted darknet. Furthermore, let us recall that Shodan is a block scanner, whereas ZMap is horizontal. It is thus worth to wonder or investigate if the intensity

time-stamp	source IP address	destination IP address	targeted port
1452047364.234832000	141.212.122.123	192.168.1.2	23
1452047364.234876000	141.212.122.122	192.168.1.2	23
1452047423.628877000	141.212.122.114	192.168.2.1	23

Table 5.1: Few logs of probes originated by ZMap and targeting port 23

of the latter varies with respect to the targeted port, and/or if for some services it is similar to the one of ZMap regardless the targeted port. We also investigated differences and similarities of intensities of ZMap and Shodan respectively. To accomplish these goals, five datasets for ZMap and one for Shodan, summarized in Table 5.2, have been used [79]. Each of them contains logs of packets. Each log consists of the source IP address (which is the Scanning Host that sent the packet), the destination IP address, the targeted port and its time stamp. The duration of the dataset is computed by subtracting its last and the first time stamp. More in detail, the dataset labeled *Shodan* contains all the received scanning traffic generated by the Shodan scanner. It was not split into specific sets, each for a given port, because, after manual investigation, Shodan confirmed to be a large mix between horizontal and vertical scanning, which is in accordance with its features detailed in Section 5.3 and in [25]. Regarding ZMap, which is instead an horizontal scanner (detailed in Section 5.3), we chose to have a "general" dataset containing all the logs of probes originated from ZMap, regardless of the targeted service (*ZMap-All*), but also sets with logs of probes targeting a given particular port. The latter have been filtered simply according to the port that is targeted by the scanner. It is chosen to model probes received on ports 22 (SSH), 23 (Telnet), 80 (HTTP) and 443 (HTTPS), and the corresponding sets are labeled with *ZMap-22*, *ZMap-23*, *ZMap-80* and *ZMap-443*, respectively. These services have been chosen because HTTP and HTTPS are actually the most used protocols to browser the internet, the latter being used for encrypted authentications. Telnet has been chosen because it is a protocol that does not provide encryption but, despite this, it is still used nowadays to access old legacy equipments that do not support more modern protocols. It is thus likely to be probed to exploit vulnerabilities. SSH is its improvement, since it enhances Telnet with encryption. Its choice is thus a logical consequence of the choice of Telnet.

Dataset	ZMap-22	ZMap-23	ZMap-80	ZMap-443	ZMap-All	Shodan
# **Sources**	178	80	246	222	253	13
# **Destinations**	4096	4096	4096	4096	4096	4096
# **Ports**	1	1	1	1	28	244
Duration (days)	532.00	119.16	541.73	216.80	533.69	12
# **Packets**	487056	147340	1078053	1536667	7992496	708160

Table 5.2: Details of used datasets

Our work aimed at modeling intensities of the probes, received by the hosted darknet, of the two considered Network Scanners. In other words, the goal is to model counts of probes received in fixed intervals of time, which is chosen to be of 10 minutes. We chose this time value because ZMap is able to scan the whole IPv4 space in about 45 minutes under some conditions, whereas Shodan requires more time. It means that, even within its shortest duration, we could have up to 5 intervals of control on which to check the probing intensity of the Network Scanner. This allows us to investigate intensities of the two considered Network Scanners from a macroscopic point of view, since the datasets we used cover long intervals of time, and at the same time to have multiple measurements within the same execution.

Once the datasets are built, tools detailed in Chapter 4 are exploited in Sections 5.5 and 5.6 to build the models of intensities of the considered Network Scanners. Before detailing the obtained results, let us firstly recall that the probability distribution commonly used for this task is the Poisson one [13, 80], since the considered variable is discrete and is a counter of times an event appears. Results are presented in Sections 5.5 and 5.6, which focus on ZMap targeting one port and ZMap, over multiple ports and Shodan, respectively.

5.5 Intensity of ZMap targeting a single service

This section presents the Finite Mixture Models and the Hidden Markov Models obtained for the datasets that contain logs of probes sent by Zmap and targeting one single service, and detailed in Section 5.4: *ZMap-22*, *ZMap-23*, *ZMap-80* and *ZMap-443*.

ZMap-22 Figure 5.2 shows the density of counters of probes received over port 22 by the darknet hosted in the High Security Lab and originated by ZMap. More in detail, on the x axis one finds the number of probes received within 10 minutes, whereas on the y axis the density of the obtained values. Dotted lines represent single Poisson distributions, whereas the continuous red line represents the mixture of distributions. Figure 5.2 thus

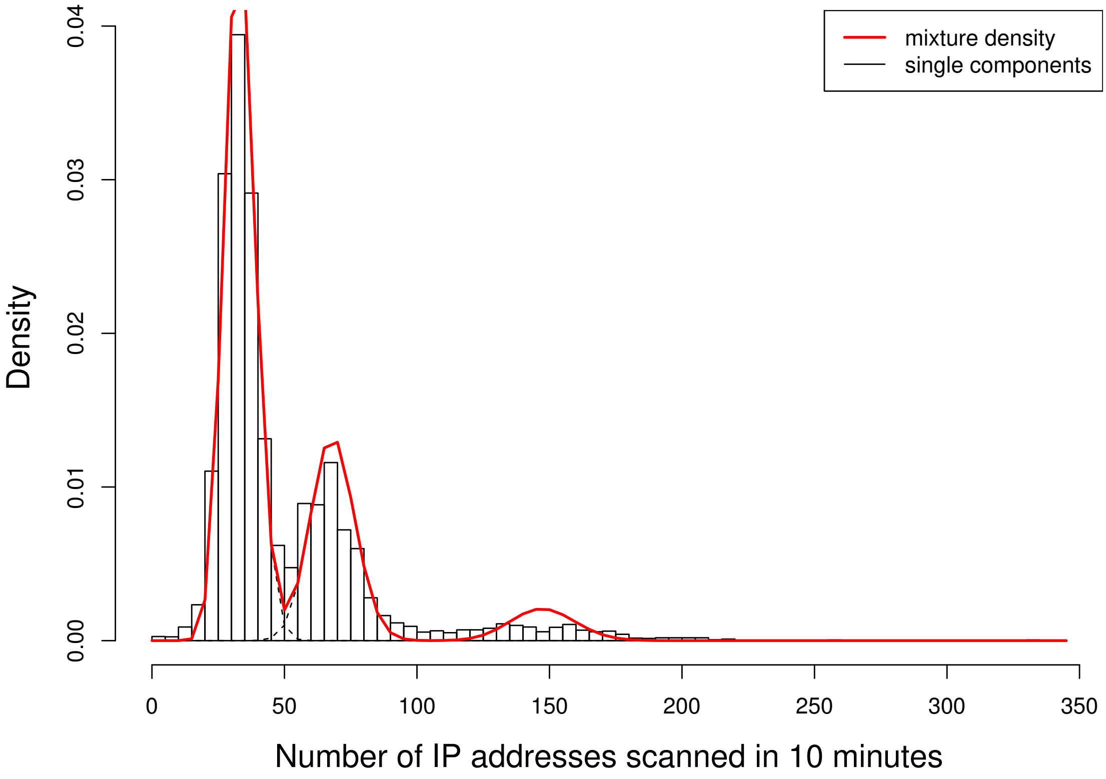
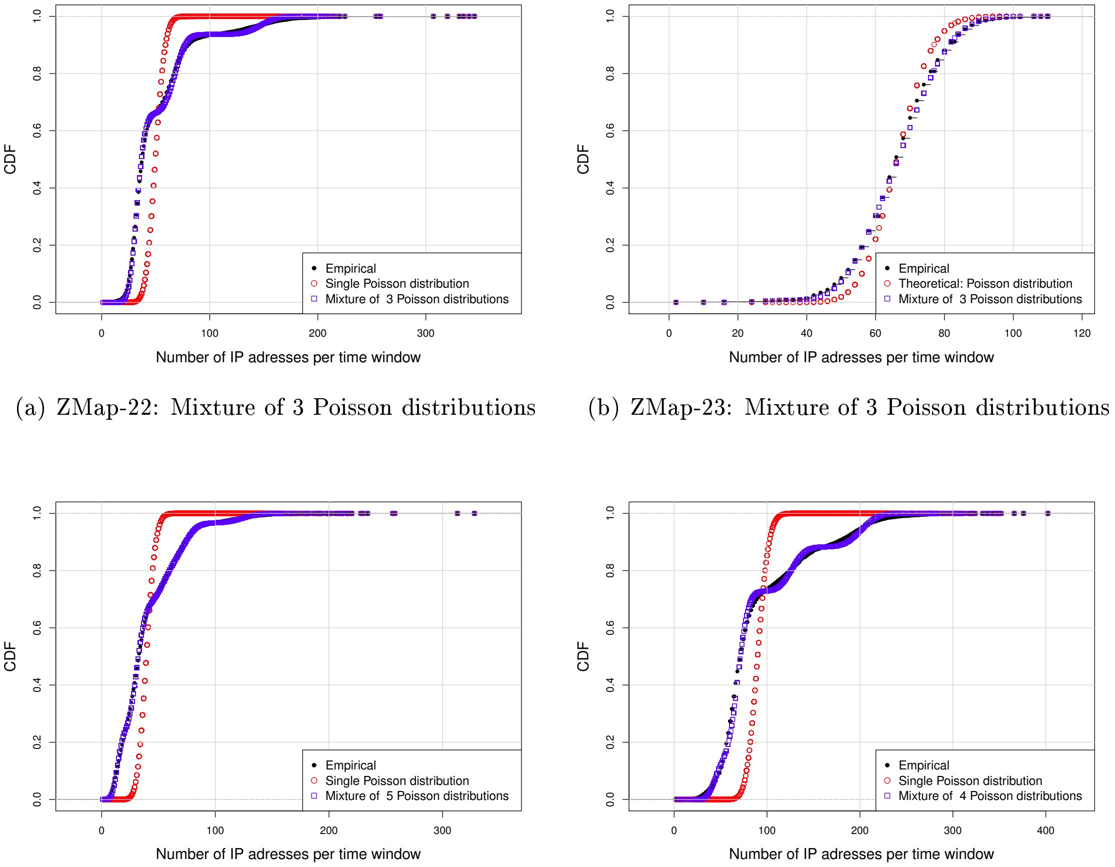


Figure 5.2: Fitting counters of IP addresses scanned within 10 minutes by ZMap on port 22 with Poisson Distributions and FMM



(a) ZMap-22: Mixture of 3 Poisson distributions

(b) ZMap-23: Mixture of 3 Poisson distributions

(c) ZMap-80: Mixture of 5 Poisson distributions

(d) ZMap-443: Mixture of 4 Poisson distributions

Figure 5.3: Cumulative Distribution Functions (CDFs) for *ZMap-22*, *ZMap-23*, *ZMap-80*, *ZMap-443*.

shows graphically that a single Poisson distribution does not suffice to well model the set of observations. It is also confirmed by looking at Figure 5.3(a), that shows the Cumulative Distribution Function (CDF) of the dataset *ZMap-22*. More precisely, black dots represent the CDF of the data, red dots the CDF of a single Poisson distribution and blue dots the CDF of the finally chosen mixture of distributions. A Finite Mixture Model with 3 Poisson components, with vectors $\boldsymbol{\lambda}$ of parameters of each component, and $\boldsymbol{\delta}$ of mixing probabilities, given by:

- $\boldsymbol{\lambda} = (32.24, 68.48, 150.60)$;
- $\boldsymbol{\delta} = (0.661, 0.277, 0.062)$;

respectively, gives good fitting, as shown in Figure 5.3(a), blue dots. The obtained FMM has been compared with the one with 4 Poisson components, and some comparisons follow. The first two parameters of the two models, as their probabilities, are really close, but the third parameter in the FMM with 3 components seems to be split into two parameters in the FMM with 4 Poisson components, whose probabilities sum to the probability of the third cluster in the FMM with 3 components. The presence of one more parameter furthermore provides only marginal improvements, as it is possible to see in Figure 5.4 that represents BIC values for each considered Finite Mixture Model and from the CDF of the FMM with 4 Poisson components, which si barely different from the one in Figure 5.3(a), for the FMM with 3 Poisson components. More in detail, in Figure 5.4 the considered FMMs built on the dataset *ZMap-22* are represented by blue squared dots, and each dataset is represented by a different color. All these considerations make thus no reasonable to

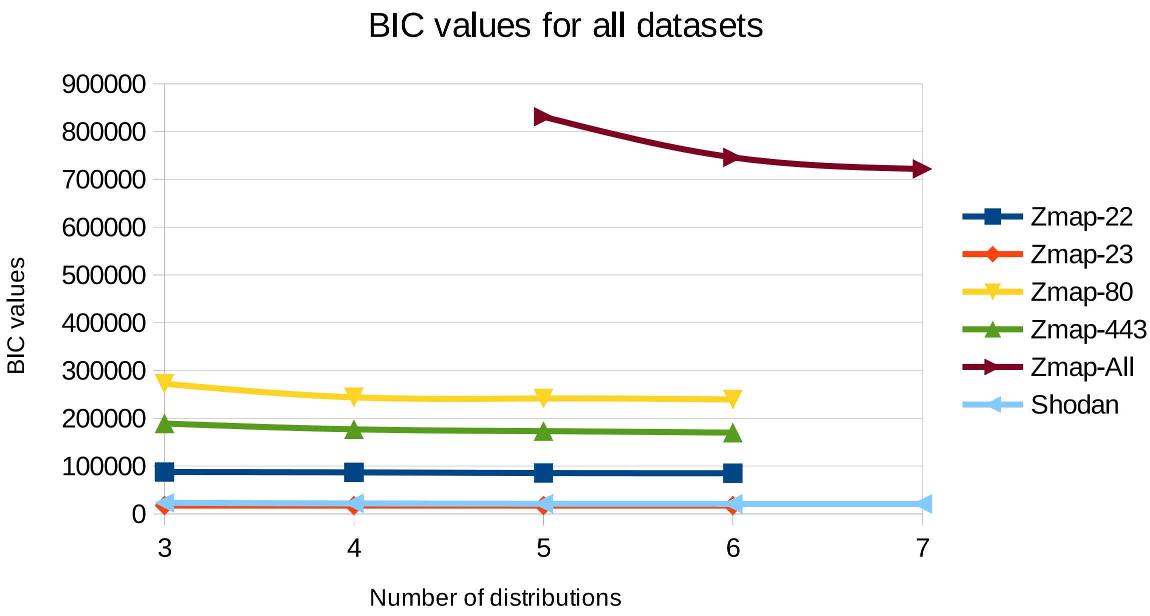


Figure 5.4: BIC values of the mixture of Poisson distribution models.

augment the minimum number of components of a FMM needed to well fit the set *ZMap-22*. Augmenting the number of distribution of the FMM implies an increasing number of parameters. In this case of Finite Poisson Mixture Models, which is the case that needs less parameters, adding one distribution implies the need of two other parameters: one for the added Poisson distribution itself, λ, which is the average number of packets received in the considered interval and which is in the cluster individuated by this “added” distribution, and one element of the vector $\boldsymbol{\delta}$ of the mixing probabilities, which is the probability of the “new” cluster. The adding of the latter implies also that the constraints over the elements of $\boldsymbol{\delta}$ need to be adapted. Therefore, an increasing of the number of distributions in the mixture implies an increasing of the number of parameters, which increases the time needed for computations. It is thus generally preferred to reduce the number of needed parameters as much as possible. In this case, 3 is the *minimum* number of components of a FMM required to well fit the set of observations. In case of particular needs, it is possible to obtain good fitting FMMs with $n > 3$ components, with only marginal improvements. The Hidden Markov Model build on the dataset *ZMap-22* has thus 3 states, each linked to a component of the Finite Mixture Model obtained above. The initial state of the HMM is S_2: the vector of probabilities for the initial state of the HMM is given by $\boldsymbol{v} = (0, 1, 0)$, and probabilities to move from a state to another (i.e., the matrix $\boldsymbol{\Gamma}$) are given in Table 5.3. It is interesting to note that the probabilities to “escape” from a state, once it is entered, are

	to State 1	to State 2	to State 3
from State 1	0.967	0.000	0.033
from State 2	0.000	0.994	0.006
from State 3	0.007	0.013	0.980

Table 5.3: Transition matrix of the HMM of ZMap-22

really low, and never higher than 3.3%.

ZMap-23 After having built the Finite Mixture Model and the Hidden Markov Model for the dataset *ZMap-22*, let us move to consider the dataset *ZMap-23*, which contains logs of packets received by the darknet, originated by ZMap and targeting port 23. Figure 5.5 represents the density of counters of packets received by port 23 in fixed time intervals of 10 minutes. It shows that a single Poisson distribution is not sufficient to fit the data, since none of the three black dotted lines provide a good fitting. This is due to the over-dispersion of the observations, which have a mean of 66.85 and a variance equal to 147.06. Recall that over-dispersion is the presence of greater variability (i.e., statistical dispersion) in a dataset that would be expected, based on a statistical model. The latter is chosen to have a theoretical population mean approximately equal to the sample mean. When,

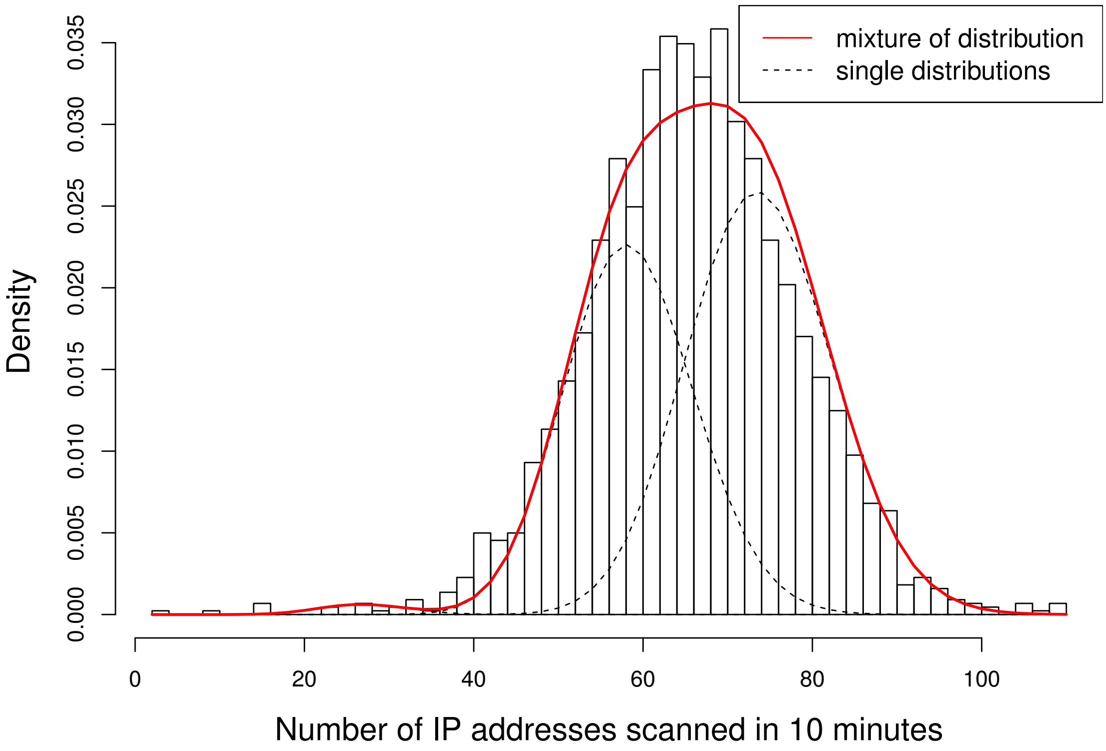


Figure 5.5: Fitting counters of IP addresses scanned within 10 minutes by ZMap on port 23 with Poisson Distributions and FMM

as in this case, the observed variance is higher than the variance of the theoretical model, then *pver-dispersion*) has occurred. Keep in mind that, in a Poisson model, the variance is equal to to the mean, λ. Since the observed variance, 147.06, is greater than the theoretical one, 66.85, over-dispersion is present in the dataset. Figure 5.5 also shows that the number of scanned IP addresses in 10 minutes could be well fitted by a Finite Mixture of three Poisson distributions Model, represented by the red line.

Appropriate FMMs for the considered dataset could have 3 and 6 Poisson distributions, respectively. The final choice was to fit the dataset *ZMap-23* with a mixture of 3 Poisson distributions, with parameters and probabilities $\boldsymbol{\lambda} = (27.16, 58.55, 73.92)$ and $\boldsymbol{\delta} = (0.008, 0.434, 0.558)$ respectively, for various reasons. First of all, the plot of the Cumulative Distribution Function (CDF), shown in Figure 5.3(b), blue dots, for the FMM with 3 components well fits the observations, whose CDF is represented by black dots. The latter is also quite entirely overlapped by the plot of the CDF of the FMM with 6 Poisson components, meaning that both FMMs are suited to model the set of observations[7]. Furthermore, the difference between the obtained BIC values, as shown in Figure 5.4, red squares, is really low, and three of the components in the FMM with the mixture of 6 Poisson distributions are really rare to be selected[8]. 3 is thus the minimum number of components required by a FMM to well fit the dataset *ZMap-23.* As for the FMM fitting the dataset *ZMap-22*, it is always possible to fit *ZMap-23* with FMMs with more components, but the obtained improvements would be marginal.

The question to answer, now, is the following. Since both datasets *ZMap-2s* and *ZMap-23* have been well modeled with FMMs with 3 Poisson distributions each, and ports 22 and 23 are both reserved for remote connections, is it reasonable to consider only one model for both datasets? The answer is negative, because ZMap behaves differently when probing port 22 and 23, as it is possible to see in Figures 5.3(b) and 5.3(a). This is also confirmed by the different vectors of parameters $\boldsymbol{\lambda}$ and of mixing probabilities $\boldsymbol{\delta}$, which are not comparable (see Table 5.4): despite the parameters of the first distribution of the FMMs for *ZMap-22* and *ZMap-23*, respectively, may be considered relatively close (27.16 and 32.24 respectively), the mixing probabilities of the corresponding clusters are highly different (0.008 against 0.661).

Dataset	$\boldsymbol{\lambda}$	$\boldsymbol{\delta}$
Zmap-22	(32.24, 68.48, 150.60)	(0.661, 0.277, 0.062)
ZMap-23	(27.16, 58.55, 73.92)	(0.008, 0.434, 0.558)

Table 5.4: $\boldsymbol{\delta}$s and $\boldsymbol{\lambda}$s of FMMs for *ZMap-22* and *ZMap-23*

The Hidden Markov Model for the dataset *ZMap-23* has thus 3 states, each corresponding to one of the three componenents of the FMM, and its initial one is State S_1, as shown by the vector containing the initial state probabilities, $\boldsymbol{v} = (1, 0, 0)$. Transition probabilities between states (or, in other terms, the matrix $\boldsymbol{\Gamma}$) are given in Table 5.5[9].

Differently from the Hidden Markov Models learned on *ZMap-22*, the probabilities to escape from a given state are considerable: with the exception of state S_3, probabilities to leave a state are higher than the ones to remain in it.

[7] it is not present in Figures because they show only chosen models. It was checked during the work

[8] Response to the note of Abdelkader. Yes, we had this info. I had checked it when we wrote the paper. I do not have anymore the values, since the case with 6 components was discarded

[9] State 1 = component 1? I think so, but how to check it?

	to State 1	**to State 2**	**to State 3**
from State 1	0.253	0.421	0.326
from State 2	0.004	0.394	0.602
from State 3	0.003	0.417	0.579

Table 5.5: Transition matrix $\mathbf{\Gamma}$ of the HMM of ZMap-23

ZMap-80 Figure 5.6 represents densities of counts of packets received by the darknet, originated by ZMap and targeting port 80. It shows that, again, a single Poisson distribution does not fit the data (see black dotted lines in Figure 5.6), but a Finite Mixture Model with 5 Poisson components is needed, represented by the red continuous line. Its vectors of

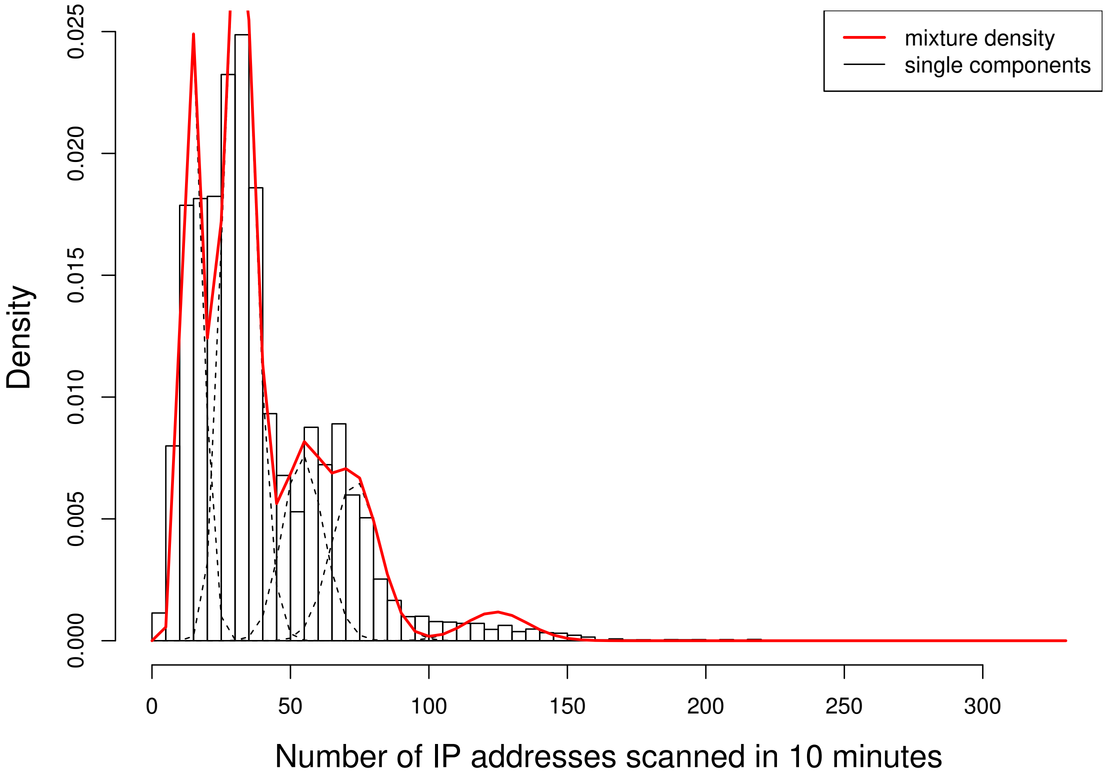


Figure 5.6: Fitting counters of IP addresses scanned within 10 minutes by ZMap on port 80 with Poisson Distributions and FMM

parameters $\boldsymbol{\lambda}$ and of mixing probabilities $\boldsymbol{\delta}$, respectively, are given by

- $\boldsymbol{\lambda} = (14.73, 31.75, 54.75, 73.79, 124.76)$;
- $\boldsymbol{\delta} = (0.242, 0.442, 0.1415, 0.1415, 0.033)$.

We wondered again, as done for the Finite Mixture Models built on previously considered datasets, if the FMM that fits the set of logs *ZMap-80* could be used to well model the datasets *ZMap-22* and *ZMap-23*. The answer is negative, for various reasons. If one looks naively at the obtained Finite Mixture Models, one may say that the main reason lies in the fact that the FMMs built to fit *ZMap-22* and *ZMap-23*, respectively, have 3 components, whereas the FMM that fits *ZMap-80* has 5 of them. In fact, this is not the reason of this non-feasibility, since, as explained above, the FMMs we present here are the ones with the minimum number of components required to well fit the data. In other words, it is possible to fit the sets *ZMap-22* and *ZMap-23* with FMMs with 5 components. But, also if one does it, the differences between parameters of each of their components and between the mixing probabilities, respectively, would be too large to consider feasible to use a single FMM to well fit more than one of the considered datasets.

The Hidden Markov Model for the dataset *ZMap-80* and built on the obtained FMM has, thus, 5 states. The third state, S_3, is the initial one, since the vector of probabilities of initial states is given by $\boldsymbol{v} = (0, 0, 1, 0, 0)$. Its transition probabilities are shown in Table 5.6: also in this case, as for the HMM built on *ZMap-22*, it is unlikely to exit a state, once it is entered. More precisely, the probability to escape from a state is never higher than 4%.

	to S.1	**to S.2**	**to S.3**	**to S.4**	**to S.5**
from S.1	0.962	0.000	0.002	0.031	0.005
from S.2	0.000	0.988	0.004	0.008	0.000
from S.3	0.003	0.003	0.982	0.012	0.000
from S.4	0.020	0.003	0.006	0.970	0.000
from S.5	0.030	0.000	0.000	0.000	0.970

Table 5.6: Transition matrix of the HMM of ZMap-80

ZMap-443 Figure 5.7, as for the previous three datasets, shows the density of counts of logs of packets received within 10 minutes by the darknet and originated by ZMap targeting port 443. Again, black dotted lines represent different Poisson distributions: none of them well fits the data. A Finite Mixture Model is thus needed, and it splits the dataset *ZMap-443* into 4 clusters. The vectors of parameters of the Poisson distributions, $\boldsymbol{\lambda}$, and of the mixing probabilities $\boldsymbol{\delta}$ of the obtained FMM are:

- $\boldsymbol{\lambda} = (44.78, 69.69, 127.37, 201.83)$;
- $\boldsymbol{\delta} = (0.145, 0.583, 0.155, 0.117)$.

We investigated if the addition of one component to the FMM provides improvements in the fitting. Despite a FMM with 5 components would slightly better fit the dataset, improvements would only be really marginal, Figure 5.4 shows that the difference of the corresponding BIC values are minimal, see green triangles. Furthermore, the Cumulative Distribution Function (CDF) of the FMM with 4 components already provides a good overlap to the CDF of the observations, as shown in Figure 5.3(d). For these reasons, the selected FMM to model the intensity of received probes generated by ZMap and targeting port 443 consists of 4 components, with parameters given above.

Again, let us wonder if one of the FMMs built for *ZMap-22*, *ZMap-23* or *ZMap-80*, respectively, could well fit the dataset *ZMap-443* or, viceversa, if the FMM built for the dataset *ZMap-443* can well fit one, or more, of the datasets *ZMap-22*, *ZMap-23* and *ZMap-80.* Once more, the answer is negative. A naive observer could again say that the main reason of the negative answer is that the FMM for *ZMap-443* has 4 components, whereas the

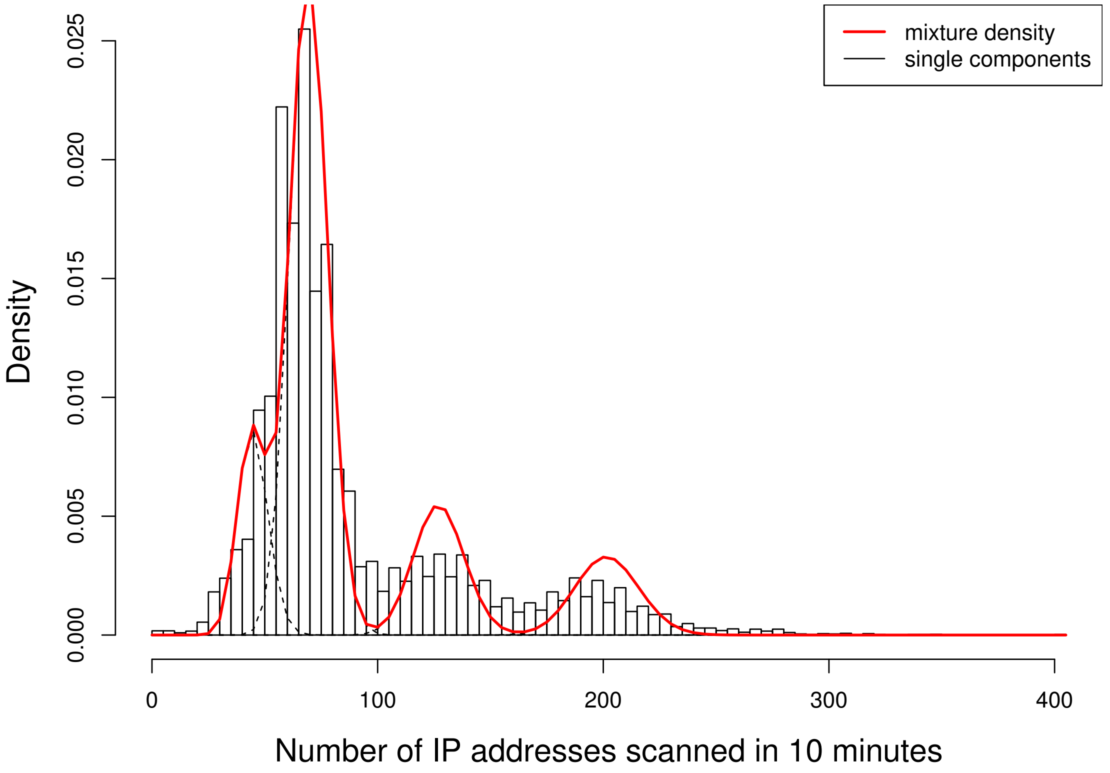


Figure 5.7: Fitting counters of IP addresses scanned within 10 minutes by ZMap on port 443 with Poisson Distributions and FMM

FMMs for *ZMap-22* and *ZMap-23* have 3, each, and the FMM for *ZMap-80* has 5. Despite it is true, let us remember once more that so far we gave FMMs with the *minimum* number of components required to well fit the considered datasets. This means that it is always possible to have FMMs with the same number of componenets, for all the considered datasets. But, even if it is thus possible to model both *ZMap-22* and *ZMap-23* with a FMM having 4 Poisson distributions, as seen before, or all the datasets considered so far with FMMs with 5 components each, their parameters would be too different from the ones of the FMM for the set *ZMap-443*. The consequence of this is that it is not advisable to use only a FMM to fit some of or all the datasets, since the fitting would not be good for all of them.

Let us move the analysis to the Hidden Markov Model, which has 4 states. Its initial one is the fourth, S_4, since its vector v of the probabilities of initial states is $\boldsymbol{v} = (0, 0, 0, 1)$. Transition probabilities between states are given in Table 5.7. Also in this case, probabilities to escape from a state are relatively low: usually less than 12%, with the exception of the probability to escape from S_2, which is equal to 32%[10].

[10] The hypothesis/interpretation of Abdelkader needs further investigation, because theoretically it is wrong: the $\boldsymbol{\delta}$ contains initial probabilities of States of the HMM, which means probabilities at time $t = 0$. Transition probabilities hold also for successive times, so the hypothesis that they depend so strongly on the initial probabilities

	to State 1	**to State 2**	**to State 3**	**State 4**
from State 1	0.885	0.010	0.028	0.078
from State 2	0.020	0.680	0.013	0.287
from State 3	0.066	0.000	0.925	0.010
from State 4	0.015	0.045	0.005	0.935

Table 5.7: Transition probabilities matrix $\mathbf{\Gamma}$ of the HMM of ZMap-443

This section focused on the learning and comparisons of Finite Mixture Models and Hidden Markov Models for datasets of logs of probes originated by ZMap, and targeting one single service. The following section presents instead results for ZMap targeting multiple ports, and Shodan.

5.6 Intensity of ZMap targeting various services and Shodan

This section shows the results we obtained for the two datasets of logs *ZMap-All* and *Shodan*, that contain counts of probes originated by ZMap and Shodan, respectively, and received by the hosted darknet in intervals of 10 minutes. The choice to consider a dataset of logs of probes originated by ZMap, regardless the targeted service, is motivated by the fact that Shodan is a block scanner. In order to find out possible similarities between intensities of the two scanners, we decided to have also a dataset in which ZMap behaves "like" Shodan. Results are shown in this section.

ZMap-All The set *ZMap-All* contains logs of probes originated by ZMap and received by our darknet. Figure 5.8 represents densities of counters of probes received within intervals of 10 minutes by our darknet. It highlights that the dataset, like all the others analyzed in Section 5.5, cannot be modeled with a single Poisson distribution, but a Finite Mixture Model with at least 6 components is needed.

Figure 5.9(a) instead displays the Cumulative Distribution Function (CDF) of the observations in the dataset, represented by black dots, of one Poisson distribution (red dots) and of the FMM with 6 components (blue dots), respectively, and confirms the necessity of a Finite Mixture Model to well fit the data. The vector containing the parameters of the 6 components is $\boldsymbol{\lambda} = (32.00, 72.83, 122.19, 183.49, 276.42, 502.56)$, whereas the vector of the mixing probabilities is given by $\boldsymbol{\delta} = (0.121, 0.216, 0.345, 0.220, 0.083, 0.015)$. Recall that 6 is the minimum number of components required by a FMM to well fit the data. If, as done for other datasets and detailed in Section 5.5, we wonder if it is possible to use a FMM to well fit two or more datasets, we come across a negative answer, for reasons that are extremely similar to the ones previously enunciated. Even if we build FMMs with 6 components for each of the previously considered (*ZMap-22*, *ZMap-23*, *ZMap-80* and *ZMap-443*), their corresponding parameters (i.e., their mixing distributions and the parameters of their components) would be deeply different. It follows that a unique model cannot provide good fitting for multiple sets.

Let us now move forward, and compute the Hidden Markov Model based on the Finite Mixture Model just learned. The HMM has thus 6 states and the initial one is S_1: the vector of the probabilities of the initial state of the model being $\boldsymbol{v} = (1, 0, 0, 0, 0, 0)$. Its transition matrix $\mathbf{\Gamma}$ is shown in Table 5.8.

needs to be investigated. $\boldsymbol{\delta}$ does not give the "overall" importance of States.

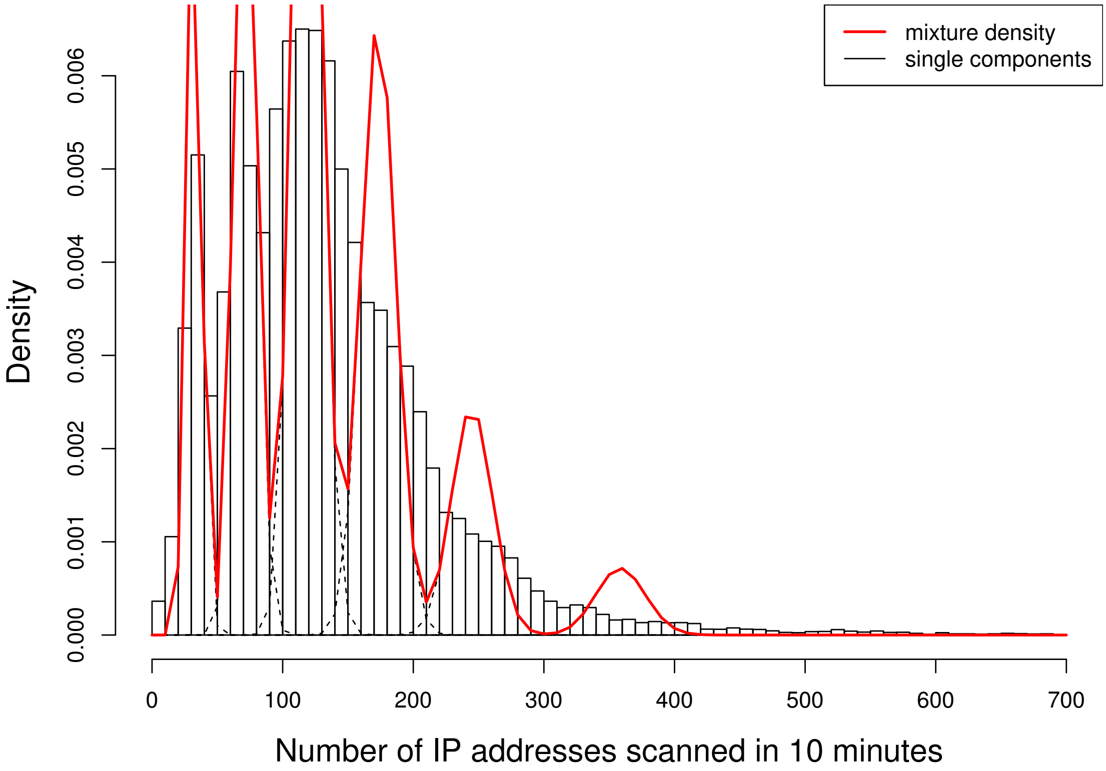
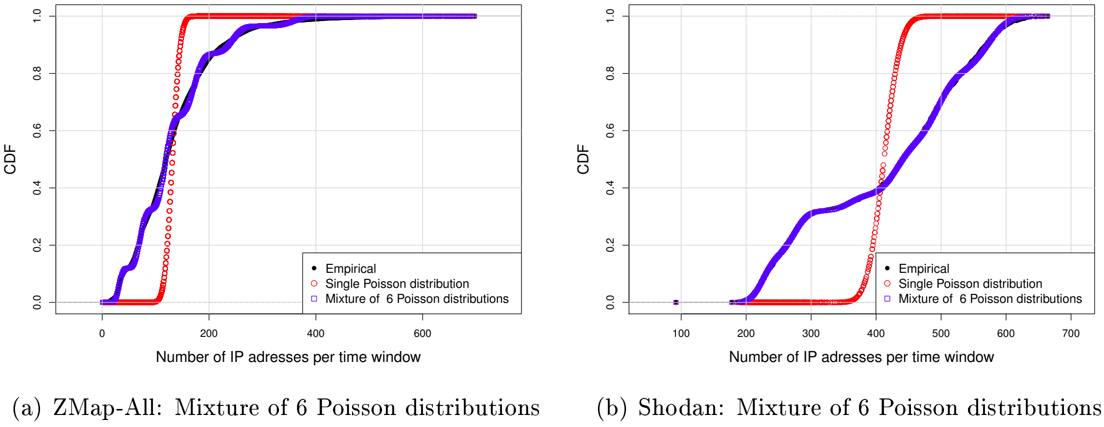


Figure 5.8: Fitting counters of IP addresses scanned within 10 minutes by ZMap with Poisson Distributions and FMM



(a) ZMap-All: Mixture of 6 Poisson distributions

(b) Shodan: Mixture of 6 Poisson distributions

Figure 5.9: Cumulative Distribution Functions for *ZMap-All* and *Shodan*

Note that the HMM for ZMap targeting multiple services shares a feature with all the ones previously built on datasets generated by ZMap, which is that, it is rare to move from one

	to S.1	to S.2	to S.3	to S.4	to S.5	to S.6
from S.1	0.905	0.000	0.002	0.008	0.015	0.070
from S.2	0.001	0.877	0.112	0.009	0.001	0.000
from S.3	0.001	0.019	0.868	0.104	0.007	0.001
from S.4	0.001	0.001	0.037	0.864	0.093	0.004
from S.5	0.003	0.000	0.002	0.052	0.886	0.057
from S.6	0.048	0.001	0.002	0.009	0.083	0.857

Table 5.8: Transition matrix of the HMM of ZMap-All

state to another: once a state is entered, it is unlikely to change. The probability to change state is never higher than 14.3%.

Shodan Figure 5.10 shows that the set containing logs of probes generated by Shodan cannot be well fitted by a unique Poisson distribution. Despite it could be fairly well fitted with a FMM with 4 components, the finally chosen FMM consists of 6 components, which is the one represented by the red continuous line in Figure 5.10, and whose Cumulative Distribution Function (CDF) is represented in Figure 5.9(b) by blue dots. This choice is motivated by the improvements that it provides in terms of BIC values, shown in Figure

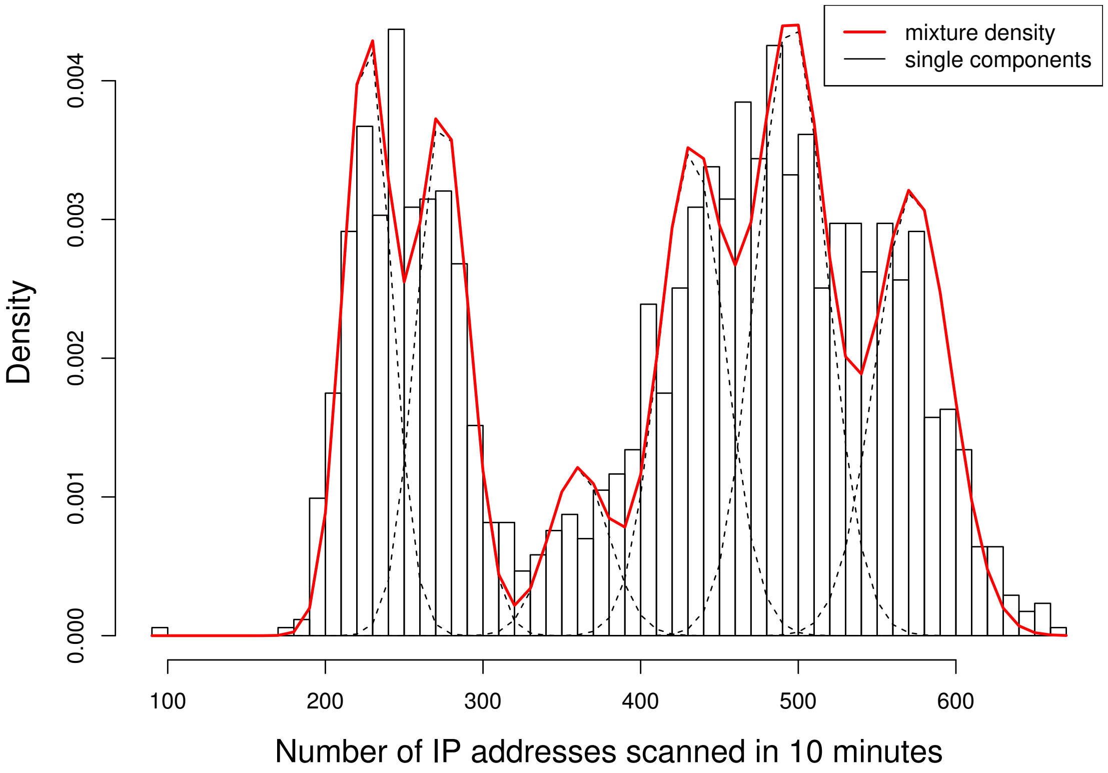


Figure 5.10: Fitting counters of IP addresses scanned within 10 minutes by Shodan with Poisson Distributions and FMM

5.4 (see light blue triangles), and of CDFs. The parameters of the 6 components and the mixing probabilities of the chosen FMM are:

- $\boldsymbol{\lambda} = (226.82, 274.87, 360.97, 432.85, 496.08, 573.15)$;
- $\boldsymbol{\delta} = (0.163, 0.157, 0.057, 0.182, 0.248, 0.193)$.

Despite the minimum number of components of a FMM to well fit the datasets *ZMap-All* and *Shodan*, respectively, is 6, their parameters are highly different. A similar phenomenon was shown for the FMMs which fit *ZMap-22* and *ZMap-23*, respectively: the minimum number of components is the same, but their parameters are really different. The Network Scanners proceed thus very differently, as also highlighted by their CDFs in Figure 5.9.

The HMM describing *Shodan* has, as the one built on the dataset *ZMap-All*, 6 states, but it starts in state S_5, as shown by the vector $\boldsymbol{v} = (0, 0, 0, 0, 1, 0)$ of probabilities of initial states. Its transition matrix $\boldsymbol{\Gamma}$ is instead represented in Table 5.9. As for for *ZMap-All*, the HMM for *Shodan* has low transition probabilities between different states, with a probability to escape a state lower than 25%. It is interesting to note that this feature is common to all modeled datasets, with the exception of *ZMap-23*, whose different behavior is also clear from its CDF (Figure 5.3(b)).

	to S.1	**to S.2**	**to S.3**	**to S.4**	**to S.5**	**to S.6**
from S.1	0.909	0.000	0.000	0.000	0.000	0.091
from S.2	0.000	0.758	0.000	0.068	0.162	0.012
from S.3	0.000	0.000	0.905	0.000	0.095	0.000
from S.4	0.000	0.140	0.000	0.820	0.040	0.000
from S.5	0.000	0.157	0.089	0.000	0.753	0.001
from S.6	0.066	0.007	0.000	0.000	0.005	0.922

Table 5.9: Transition matrix of the HMM of Shodan.

5.7 Discussions

Sections 5.5 and 5.6 presented all the models we obtained for all the considered datasets. More precisely, Section 5.5 provided the Hidden Markov Models for each of the considered datasets of logs of packets originated by ZMap. More in detail, ports 22, 23, 80 and 443 have been selected. Section 5.6 instead provided Hidden Markov Models for the datasets *ZMap-All* and *Shodan*, containing logs of probes originated by each of the two scanners, respectively, and collected by our darknet. Parameters and initial states of each obtained Hidden Markov Models are computed.

As briefly outputted in Section 5.5, the Hidden Markov Models built on the datasets *ZMap-22* and *ZMap-23* require the same number of minimum states, even if the models themselves cannot be compared because of the difference of their parameters. This means that the intensities of each of the two datasets are clustered into three groups, and that the distributions and initial probabilities of the clusters of each datasets are different. One motivation of this shared feature can be found in the service associated to the two considered ports, which are related since SSH (port 22) is the enhancement of Telnet (port 23). It is thus reasonable to expect that their behaviors share some peculiarities, that here we proved to be the number of clusters of intensities, although two different models continue to be needed.

Let us now move on the two Hidden Markov Models detailed in Section 5.6, built on the datasets *ZMap-All* and *Shodan*, respectively. Both of them have 6 states, which it is the only thing they have in common. Counts of logs in each of the two datasets are thus grouped into 6 clusters. Unlike the previous case, the two datasets contain logs of probes originated by each Network Scanner probing multiple ports. Due to the black-box approach of Shodan, we must assume that the two considered Network Scanners behave in manners which are different one from the other. This means that the only thing that the two datasets *ZMap-All* and *Shodan* have in common in that the probes, whose logs are stored in them, target multiple ports. The only inference that it is thus possible to deduce is that 6 clusters generally suffice to group counters of probes originated by a Network Scanner and received by our network or, in other words, that Network Scanners are executed with an intensity that assumes a limited number of values, and comes from one of the six clusters.

All the HMMs detailed in Sections 5.5 and 5.6, even if they have different number of states, from a minimum of 3 and a maximum of 6, share a common feature. More precisely, their Transition Probabilities Matrices (TPMs) $\mathbf{\Gamma}$ do. Their diagonals contain the greatest elements of each row. Said in other words, the element γ_{ii} of $\mathbf{\Gamma}$, $\forall \quad i$, are such that $\gamma_{ii} >> \gamma ij \quad \forall j \neq i$. As explained in both sections, this means that, once entered in one state, the probabilities to escape from it are low. But, what does it mean to "migrate" to another state? And to remain in the same state? It is reasonable to think, that intensities of Network Scanners remain constant throughout one execution. In other words, once they are launched, their intensity remains constant, unless unexpected events. Recall that we have multiple counts of probes from the same executions of the Network Scanners, as detailed in Section 5.4. Under the light of this scenario, migrating from one state to another means that:

- an execution of the considered Network Scanner ended and another began;
- the latter execution has an intensity that is different from the one of the former.

Remaining in the same state, on the other hand, has one of the two following meanings:

- two consecutive counters have similar values because they are taken within the same execution of the considered Network Scanner. Recall that it is reasonable to assume that the intensity of an execution of the Network Scanner does not change drastically before its end;
- the two successive executions of the considered Network Scanners have the same intensity. This could be motivated, for example, by te fact that users of the Network Scanners do not change its parameters before launching it again. This holds especially for ZMap, since it is impossible to know how Shodan works.

The only Hidden Markov Model that slightly varies from this pattern is the one built on the dataset *ZMap-23*, which is the smallest one originated by ZMap, covering only 119 days and containing about 147,000 logs. It is also the only one that has two component that "overlap", as shown in Figure 5.5, merging into one peak. It could be too small to originate a "definitive" model that shares the same pattern of the others.

Under the above and latter interpretation, then, we infer that ZMap, when targeting ports 22 and 23, has generically an intensity that is picked up from a set of 3. This means that, even if intensities of ZMap targeting ports 22 are different from the ones of ZMap targeting port 23, the considered Network Scanner probes the two ports with a limited number of intensities, 3, despite the intensities themselves are different. The triplet of parameters of ZMap chosen by the user are then generally selected from the same and fixed set of triplets, a subset of all eligible triplets of parameters for ZMap. A similar deduction can be done for ZMap targeting multiple ports:

the triplets of parameters chosen by the user of the Network Scanners are generally selected from a “fixed” set of eligible triplets. If it was not the case, then intensities of executions could vary much more then what obtained throughout our work.

5.8 Summary

This chapter highlighted and detailed, in Sections 5.5 and 5.6, Finite Mixture Models and Hidden Markov Models learned on each of the datasets detailed in Section 5.4 and collected by a darknet, see Section 5.2. Section 5.7 dealt with a discussion about the obtained models and results. Summarizing, the minimum number of components of the learned Finite Mixture Models (i.e., the minimum number of states of the corresponding Hidden Markov Model) varies between 3 and 6. It is possible to have FMMs (i.e., HMMs) with the same number of components (i.e., of states) for each dataset, but none of them could be used to well fit more than one dataset. More precisely, it will well fit only the dataset used to learn it. This, because their parameters would be still too different to be compared. Since this would not decrease the number of needed models, but at the contrary it would only increase the total number of parameters to handle, we chose to keep the number of components (i.e., of states) of the FMMs (i.e., of the HMMs) as low as possible. The fact that the obtained Hidden Markov Models cannot be compared with the others, or that equivalently cannot well fit other datasets, means that each dataset has its own intensities. More precisely, the Network Scanner that originated the considered datasets has its own intensities, that are adapted to the targeted service(s) and are not comparable with the ones of other Network Scanners.

All the obtained models are *discriminative*, since they suffice to recognize what Network Scanner probed the hosted network and port. Intensities of Network Scanners is thus a useful property for long term analyses over long periods. But, on the other hand, it is less practical if one aims at identifying a Network Scanner over short periods (not longer than one single execution), with only the availability of fewer packets.

The work presented through this chapter aimed at investigating a long term feature of ZMap and Shodan, which is their intensities, over long periods of time, from several days to several months. More precisely, we questioned if intensity is a feature, or a characteristic, that could be used to recognize Network Scanners, without knowing anything but logs of their probes. We showed that the same Network Scanner, ZMap, deploys intensities that vary with respect to the probed service, and that are not comparable with the ones of the same Network Scanner targeting other ports. The obtained models are thus *discriminative* in the sense that they allow us to recognize what Network Scanner probed the hosted network and the considered service, by only exploiting intensities of received probes. But, results suggested us to investigate more in depth, if the goal is to recognize Network Scanners using short-term features. In other words, since we showed that the intensity of the Network Scanner is related to the probed port but also to its executions, our work moved to model single executions of the considered Network Scanners, and to check if it is possible to use them to recognize Network Scanners from collected logs of probes. The following Chapter 6 will thus apply the same statistical models and tools detailed in Chapter 4 and leveraged in Chapter 5, to infer HMMs of single executions of the two considered scanners, ZMap and Shodan (see Section 5.3), and to recognize them from collected logs.

6

Short Term Analyses of ZMap and Shodan

Contents

6.1 Introduction

Chapter 5 focused on learning and comparing statistical models that could be exploited to infer characteristics and intrinsic properties of intensities of the two scanners we considered, ZMap [20] and Shodan [25]. More precisely, we focused separately on ZMap targeting single services, and on ZMap and Shodan targeting various ports. In the first case, intensities of ZMap vary with respect to the targeted service. A significance of it could be that there exist some services that are more "valuable" to be exploited than others, in the sense that there are more attackers that leverage Network Scanners to probe their corresponding ports. Due to this, the corresponding ports may be scanned with higher intensities than others, and/or the clusters of their probing intensities are more numerous than others. Furthermore, interests on services could vary with time, according to the discovery of new vulnerabilities for example. The work presented in the previous chapter showed that intensities of ZMap and Shodan are their inner features, and intensities of ZMap vary also with respect to the probed service. The models we obtain thus suffice to recognize what Network Scanner originated probes collected over the hosted darknet and over long intervals of time.

This chapter aims instead at providing short term analyses of ZMap and Shodan. In other words, the methodology here proposed leverages short term features of the considered Network Scanners to build models that are successfully applied for their recognition from freshly collected logs. Here, we answer the following research questions: "Is it possible to have a set of models that can be used to recognize what Network Scanner originated the collected probes and that are not strictly dependent or related to the network that received the probes?", "Do these models vary with respect to the service probed by the scanner?". To answer these questions, two types of observations have been selected to be considered as features of the Network Scanners for the proposed methodology:

- *spatial movements*, which we defined to be *differences of IP addresses consecutively targeted by the scanner*;
- temporal movements, defined as *differences of time stamps of two consecutive probes.*

We exploited them separately to build, for both of them, their corresponding Finite Mixture Models and Hidden Markov Models. The latter will then be applied to recognize the considered Network Scanners. We build models for single executions of each of the considered Network Scanner, and use them to recognize what of them originated freshly collected sets of logs of probes.

To accomplish the goal logs collected by a darknet have been used [81] and split into datasets, according to the Network Scanner which originated the probes. They cover long intervals of time, ranging from several days to several months, containing thus logs of multiple executions of the same Network Scanner (recall that ZMap may need only 45 minutes to probe the whole IPv4 space over a single port, using a single machine [20] (see Section 5.3). In order to model the behavior of one single execution of the considered Network Scanner with respect to time and space, the two initial datasets, one for each Network Scanner, need to be split into sets containing logs of a single execution. Each dataset is thus split into samples (each containing logs of a single execution) by considering time lapses between two consecutive probes received by the darknet: when it is an outlier, an execution of the scanner ends and the following begins. Once all these samples are available, they have been used to learn Finite Gaussian Mixture Models, and their respective Hidden Markov Models. The firsts create clusters of logs and assign to each of them a Gaußian distribution. HMMs, which provide transition probabilities between states, are then built and applied to some test samples, in order to assess what Network Scanner originated the logs in the considered sample. The accuracy of our approach is higher than 95%.

The remainder of the chapter is organized as follows. Section 6.2 details the methodology to obtained samples of logs of single executions of Network Scanners (see Sub-Section 6.2.1), to learn Finite Gaussian Mixture Models and Hidden Markov Models from samples and to choose a list of "good models" for each scanner (see Sub-Section 6.2.2), and to recognize what Network Scanner originated freshly collected logs (see Sub-Section 6.2.3). Section 6.3 shows and analyzes results obtained with respect to spatial movements of Network Scanners, whereas Section 6.4 focuses on results obtained when analyzing temporal movements. Section 6.5 concludes the chapter with a summary of the presented results.

6.2 Classification Methodology

This Section details the approach we developed and applied to assess, with high accuracy, what scanner originates collected logs. The approach we propose and detail here stores logs of probes generated by a single execution of ZMap and Shodan, respectively, into learning samples. Each of them contains logs of only a single execution of the considered Network Scanner. Then, the obtained *learning samples* are used to learn models, Finite Gaußian Mixture Models firstly and Hidden Markov Models, secondly. The first ones are necessary as basis for the latter, that are then applied to identify ZMap and Shodan, respectively, in freshly collected real traces of their new single executions, stored in the so-called *test samples*. Figure 6.1 gives the processing steps of the proposed approach, which is detailed in the remainder of this Section. The first issue to face is, after all the collected logs originated by ZMap and Shodan, respectively, are stored into the corresponding datasets, *how to identify the beginning and the end of a scanning execution of the considered Network Scanner*. This problem is targeted and solved in Section 6.2.1, which provides at its end the so-called *samples*, that contain logs of one single execution of one scanner. Once samples are available, Sub-Section 6.2.2 details how stochastic models are built and learned from them. It also answers the question if all the obtained models (one for each sample) are necessary, or if a selection of them suffices. In the second case, the selection of models would be provided. The last step of the approach we propose is to recognize the scanner that produced freshly collected logs. This issue is the focus of Sub-Section 6.2.3, which details the adopted procedure to accomplish this last task.

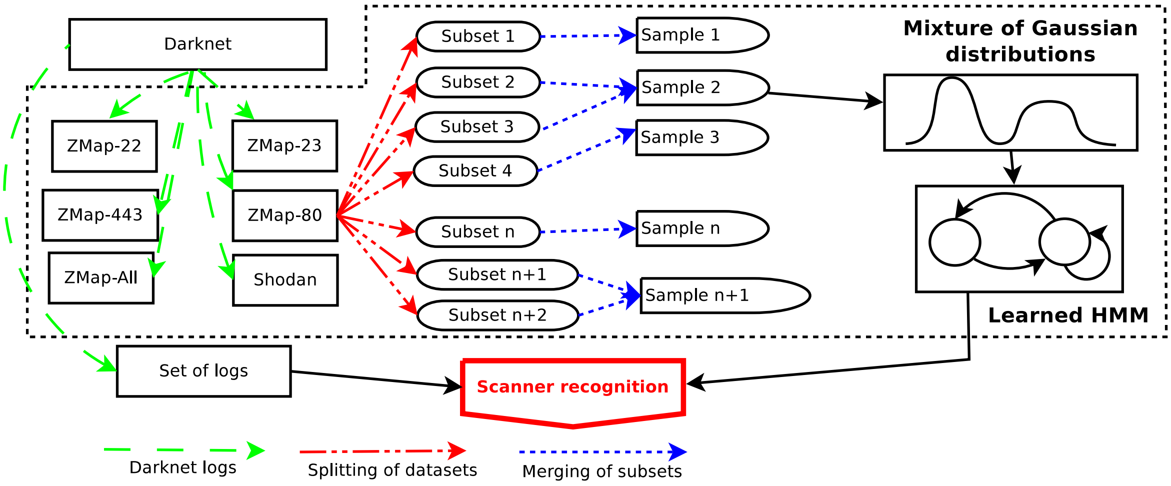


Figure 6.1: Processing steps of the proposed methodology

6.2.1 From Datasets into Samples

After the approach adopted here has been detailed, this Sub-Section shows how *samples* have been obtained and extrapolated from the two datasets containing logs of probes originated by ZMap and Shodan, respectively. Again, we leveraged real network logs generated by ZMap and Shodan, respectively, and collected over the /20 darknet hosted at the High Security Lab at Inria Nancy - Grand Est[11]. We leveraged darknet traffic for the same reasons detailed in Section 5.2. Table 5.2 summarizes the two datasets *ZMap* and *Shodan*. It also shows inner characteristics of the two Network Scanners: ZMap being horizontal and fast when scanning one single port [20], and Shodan being more massive and targeting various ports [25].

Once they are gathered and stored, manual investigation revealed that logs were not stored and downloaded in a chronological order, which we provided. Then, differences of two IP addresses consecutively probed and of two successive timestamps (the latter differences are, later on, referred as “time gaps”) are computed and stored: the firsts will be used to model spatial behaviors, the latter to model temporal behaviors of the scanners. Successively, we identified common ports below 1024 common in the two datasets: in other words, we identified those services that have been targeted by both ZMap and Shodan. This analysis was led by the seek of

[11] `http://www.lhs.loria.fr/wp/`

Dataset	*ZMap*	*Shodan*
Number of Sources	253	13
Number of Destinations	4096	4096
Number of Ports	28	244
Duration (days)	533.69	12
Number of Packets	7992496	708160

Table 6.1: Details of scanning datasets for ZMap and Shodan

any answer to the question “do executions of Network Scanners behave differently, with respect to their temporal or spatial movements, when probing different services?”. Each of the two datasets *ZMap* and *Shodan* has been split into (sub-)datasets, one for each of the 17 common ports lower than 1024. They are displayed in Table 6.2, together with their associated service. So, in total, 34 (sub-)datasets have been analyzed, each with a label “Scanner-port”, containing the name of the Network Scanner that originated the probes and the targeted port number. The procedure to split the datasets according to ports commonly probed is presented in the Algorithm 6.1.

Port	**Service**
21	File Transfer Protocol (FTP) control (command)
22	Secure Shell (SSH), secure logins, file transfers (spc, sftp), port forwarding
23	Telnet protocol - unencrypted communications
25	Simple Mail Transfer Protocol (SMTP), for email routing between servers
53	Domain Name System (DNS)
80	Hypertext Transfer Protocol (HTTP) QUIC (from Chronium) for HTTP
102	ISO Transport Service Access Point (TSAP) class 0 protocol
110	Post Office protocol, version 3 (POP3)
123	Network Time Protocol (NTP) used for time synchronization
143	Internet Message Access Protocol (IMAP), management of e-mail messages on a server
443	Hypertext Transfer Protocol over TLS/SSL (HTTPS) QUIC (from Chronium) for HTTPS
465	Authenticated SMTP over TLS/SSL (SMTPS)
500	Internet Security Association and Key Management Protocol (ISAKMP) Internet Key Exchange (IKE)
502	Modbus Protocol
587	e-mail message submission (SMTP)
993	Internet Message Access Protocol over TLS/SSL (IMAPS)
995	Post Office Protocol 3 over TLS/SSL (POP3S)

Table 6.2: Details of common ports in the datasets *ZMap* and *Shodan*

Recall that we are looking for datasets that contain logs of *one* single execution of the considered scanner. With this goal in mind, for each (sub-)dataset we filtered disjoint *learning* and *test datasets*. Each of them has an initial log that is selected randomly. The others are chronologically ordered logs that follow the first just chosen. The length of *learning datasets* is fixed to be equal to the 10% of the considered sub-dataset, whereas the length of *test datasets* is the 5% of the length of the sub-dataset. From each of the sub-dataset “Network Scanner-port”, we filtered thus various *learning* and *test datasets*, each with a randomly selected first log and

Algorithm 6.1 Split *ZMap* and *Shodan* into datasets according to ports

```
Input ZMap; Shodan
Output ZMap-port-p; Shodan-port-p
for dataset ∈ {ZMap; Shodan} do
    for $i = 1$ : length(dataset) do
        diff_IP[i] ← dst_IP[i] - dst_IP[i - 1]
        timegap[i] ← timestamp[i] - timestamp[i - 1]
        dataset[i] ← [dataset[i], diff_IP[i], timegap[i] ]
    end forcompute Unique_ports_dataset
end for
compute Common_ports
for dataset in {ZMap; Shodan} do
    for $j = 1$ : length(Common_ports) do
        p ← Common_ports[i]
        compute dataset_port_p ← dataset[port = p]
    end for
end for
```

a length of 10% and 5%, respectively, of the considered sub-dataset. The procedure to obtain *learning* and *test datasets* is formalized in Algorithm 6.2

Algorithm 6.2 Building *learning* and *test datasets*

```
Input ZMap-port-p; Shodan-port-p
Output learning-Zmap-port-p; test-Zmap-port-p; learning-Shodan-port-p; test-Shodan-port-p
for $i = 1$ : length(Common_ports) do
    for Sub_dataset ∈ {Zmap-port-p; Shodan-port-p} do
        $l$ ← length(Sub-dataset)
        randomly select $k \leq l - 0.1 * l$
        learning-Sub_dataset[1 : $0.1 * l$] ← Sub_dataset[$k : k + 0.1 * l$]
        test-Sub_dataset[1 : $0.1 * l$] ← Sub_dataset[$k : k + 0.1 * l$]
    end for
end for
```

In each of them, time gaps are analyzed. The presence of an outlier, which is a value greater or smaller than 3 times the standard deviation of the time gaps of the *learning/test sub-sets*, is an indicator that an execution of the considered scanner could be terminated, and a new one could start. This leads to split the *learning* and *test datasets* into *learning* and *test sub-sets.* The procedure to obtain *learning* and *test sub-sets* is formalized in Algorithm 6.3.

Theoretically, a time outlier could be present also in case of malfunctions, or delay, or disruptions of the scanner. In order to be sure that the *learning* and *test sub-sets* freshly filtered have been correctly split in accordance with complete executions of the scanners and not because of disruptions, the mutual Jaccard index of two chronologically consecutive *learning* and *test sub-sets*, respectively, is computed. More precisely, we stored into sets the IP addresses that have been probed in each *learning* and *test sub-sets*, and computed the mutual Jaccard index of these sets that correspond to two chronologically consecutive *learning* and *test sub-sets*, respectively. The latter are then merged into *learning* and *test samples* if and only if the mutual Jaccard

Algorithm 6.3 Building *learning* and *test sub-sets*

Input *learning-Zmap-port-p; test-Zmap-port-p; learning-Shodan-port-p; test-Shodan-port-p*
Output *learning-Subset-Zmap-port-p; test-Subset-Zmap-port-p; learning-Subset-Shodan-port-p; test-Subset-Shodan-port-p*

```
for $i = 1$ : length(Common_ports) do
    $p$ = Common_ports[i]
    for Sub_dataset $\in$ {Zmap-port-p; Shodan-port-p} do
        for $j \in$ {learning; test} do
            compute $sd \leftarrow$ stdev($j$-Sub_dataset)
            for $k = 1$ : length($j$-Sub_dataset) do
                if $|j$-Sub_dataset[$k$, timegap]$| > 3 * sd$ then
                    m = [m ; $k$]
                    len_m = length(m)
                    $j$-Subset-Sub_dataset[1 : k - m[len_m - 1] - 1] $\leftarrow$
                    $j$-Sub_dataset$\Big[$m[len_m - 1] + 1 : $k$ - 1$\Big]$
                end if
            end for
        end for
    end for
end for
```

index of the corresponding sets that contain the IP addresses probed within them is less or equal to 0.1. This means that the overlapping between the two sets of probed IP addresses is small, and it is reasonable to assume that the *learning* and *test sub-sets* that originated them can be considered as parts of the same single execution of the analyzed Network Scanner, and thus can be merged together. This merging makes us sure that each obtained *learning* and *test sample* S_i contains logs of a single execution of the considered Network Scanner. Data corresponding to time difference outliers in each *sample* are removed. Each *learning* and *test sample* is then denoted by a label of the type "scanner-port", providing thus information regarding the scanner that originated logs and the targeted service. The result is thus a set of labeled *learning* and *test samples*: scanner-port-LS_i (i.e., zmap-22-LS_i) and scanner-port-TS_i (i.e., zmap-22-LS_i), respectively.

Just to provide an example and to check the validity/goodness of the procedure detailed above to build *learning* and *test samples*, let us look at Figure 6.2. It shows the number of packets (black lines) and the number of unique IP addresses (red lines) that have been probed and are contained in each obtained *learning sample* generated by ZMap scanning port 80. Small size *learning* and *test samples* have been discarded from further consideration by considering only the ones whose length is greater than the average length of the obtained *learning* (*test*, respectively) *samples*. Despite the merging, many samples contain a number of logs of probes that is equal to the number of probed IP addresses. This means that all the IP addresses within the *sample* are probed exactly once. If the number of logs is higher than the number of probed IP addresses, then some of the latter have been probed at least twice. The facts that this scenario happens for few samples and that, in them, the number of logs is only slightly higher than the number of probed IP addresses, are indicators that the merging of *sub-sets* was needed and correctly performed.

Once we have on our hands *learning samples*, the following Sub-Section 6.2.2 will present how

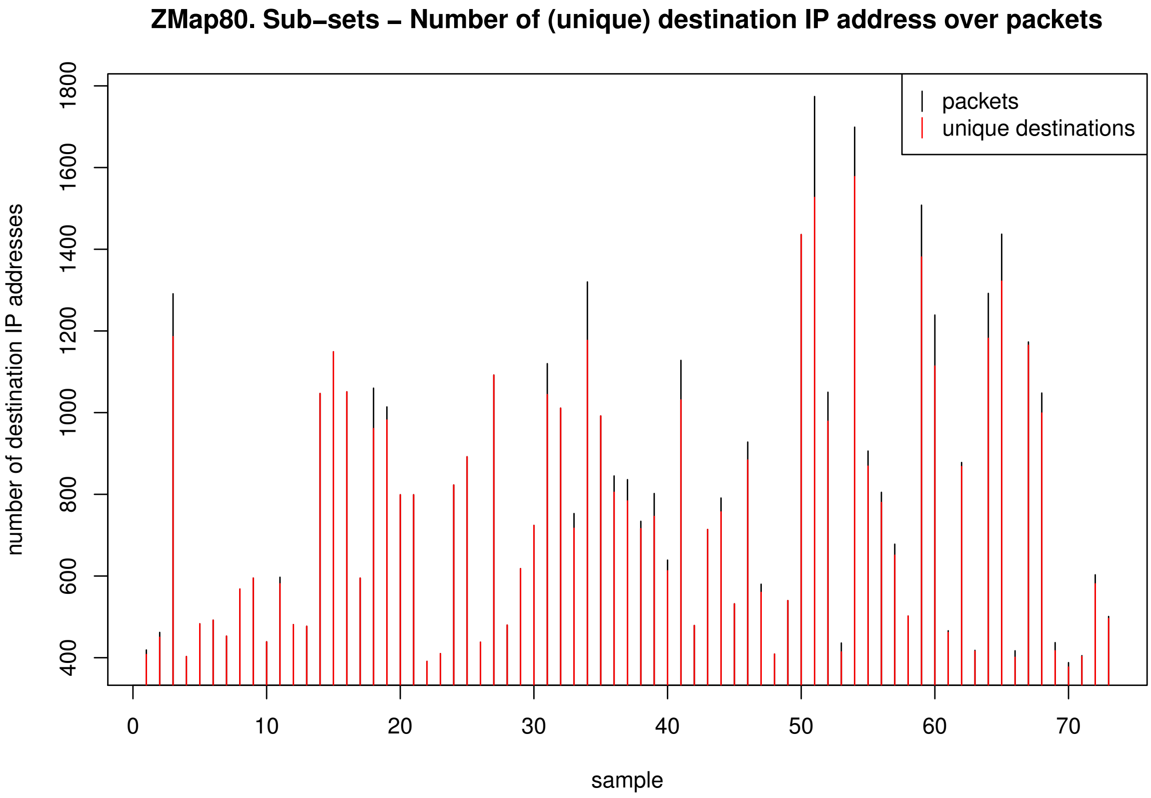


Figure 6.2: Number of unique IP addresses over number of packets of samples of ZMap executions targeting port 80

to use them to learn Hidden Markov Models of the two considered Network Scanners. Sub-Section 6.2.3 will show instead how *test samples* have been leveraged.

6.2.2 How to learn Models?

Previous Sub-Section 6.2.1 detailed the procedure we followed to obtain *learning* and *test* samples, respectively. The first ones will be here used to learn Finite Gaußian Mixture Models and Hidden Markov Models, whereas the latter in Sub-Section 6.2.3 to validate them. The choice of Gaußian distributions is made for both observations, because time gaps are continuous, thus well modeled by this type of distribution, and because discrete distributions based on large samples, as differences of consecutively targeted IP addresses in our case, can be easily and well approximated by the Gaußian distribution, thanks to the Central Limit Theorem [82].

The reason why Finite Mixture Models are needed is graphically represented in Figure 6.3, which shows densities of values of differences between two IP addresses consecutively probed by ZMap targeting port 80. It shows that one single Gaußian distribution (two of them are represented by continuous black lines in Figure 6.3) is not sufficient to model logs contained in the *learning* and *test samples*. A Finite Gaußian Mixture Model is thus required, which clusters logs and provides a probability to each of them. The mixture of the two Gaußian distributions is represented by the red continuous line in Figure 6.3.

For each *learning* and *test sample*, we thus obtain a clustering of observations, each one with its own Gaußian distribution, and the probability of each cluster. Transition probabilities between clusters are provided by the corresponding Hidden Markov Model, whose states are linked with the m components of the Finite Gaußian Mixture Model just obtained.

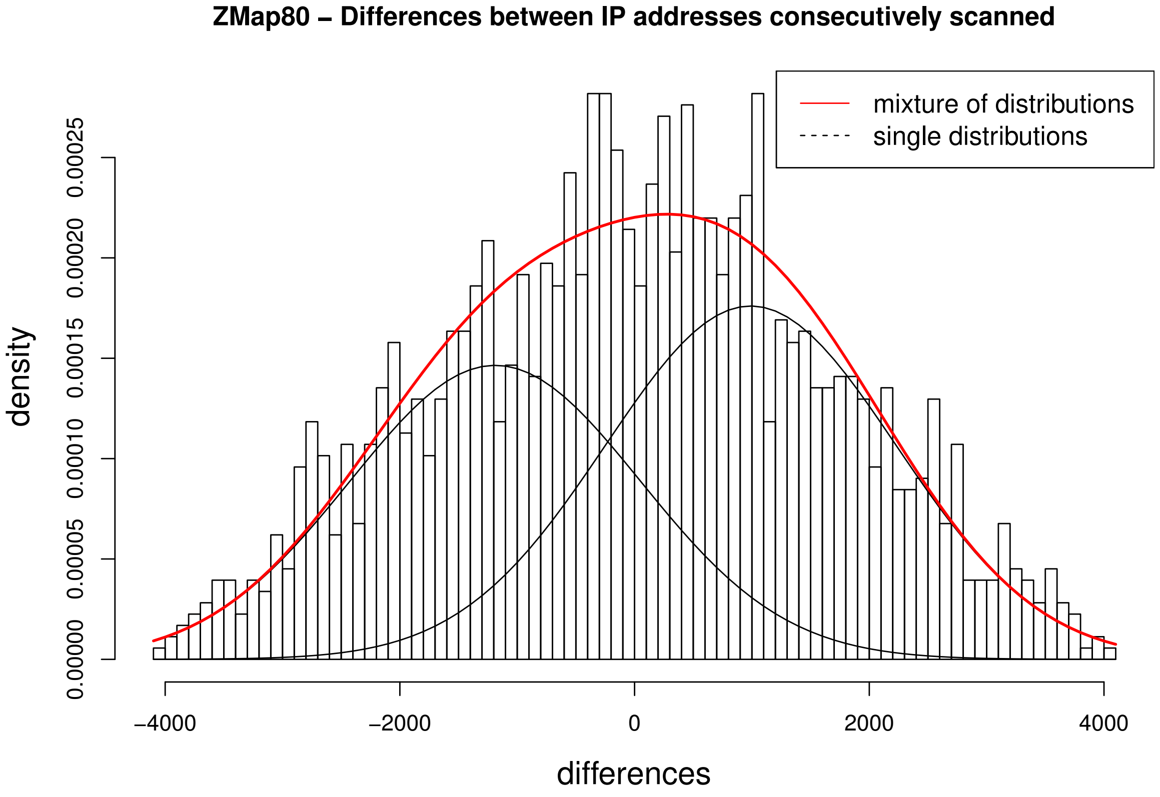


Figure 6.3: Mixture of 2 Gaußian distribution of differences of IP addresses for a ZMap execution targeting the port 80

Up to now, we have obtained a Hidden Markov Model for each *learning sample*. We would like to reduce their number. The questions that we want to answer now are: “Are there models, among the obtained ones, that are somehow similar?”, “Are there some models that are ‘better’ than others?”, and “Do we need all the obtained models, or is it possible to have a reduced set of *candidate* ones?”. In other words, we wonder if there exist some models that can be used to well fit multiple *learning samples*. To answer all these questions, each *learning sample* LS_i has been fitted to the HMM built on the learning sample LS_j, HMM_j, and its log-likelihood computed. Figure 6.4 represents the normalized matrix of all the obtained log-likelihoods of 73 samples of ZMap probing port 80. More in detail, each row is linked to a learning sample LS_i, whereas columns are reserved for the models HMM_j. A normalized log-likelihood value close to 1 is represented, in Figure 6.4, by a color close to blue, and states that the considered model well fits the considered *learning sample*. All the obtained HMMs are then grouped using *DBSCAN* [83] which, requiring only two parameters, ϵ and the minimum number of models in each cluster, gathers pairs of models if both the following conditions are satisfied:

- their distance is less or equal to the chosen ϵ;
- one of them is surrounded by a sufficient number of other points.

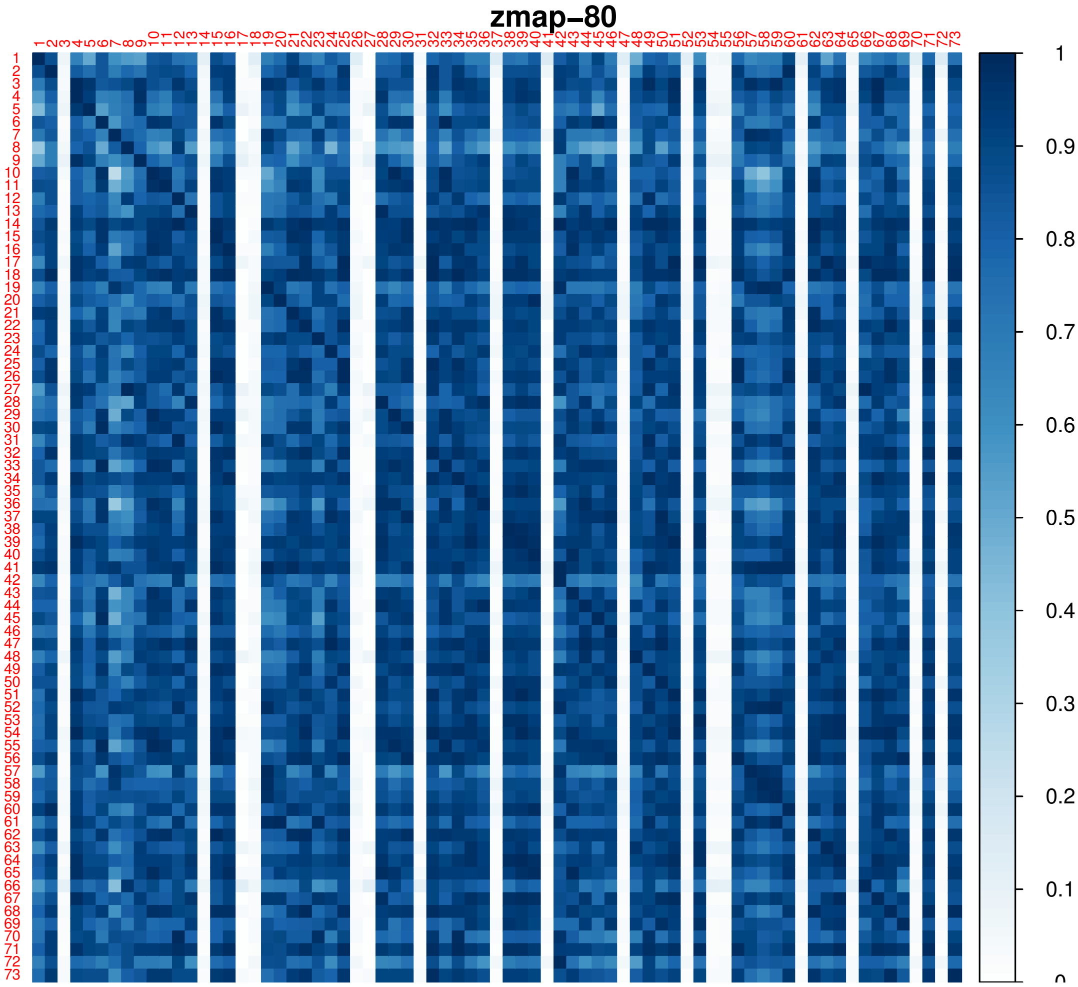


Figure 6.4: Normalized log-likelihood matrix of learned HMMs models and 73 samples for a ZMap scanning activity over port 80

All models within a single group can be considered as equivalent to describe the *learning samples* that originated them, but only one is required: for this reason, only one model is selected from each group, and is the *candidate model* of the group of HMMs. The final result is thus a list of *candidate models* M_i for each combination "Network Scanner-port", that are thus labeled as "Network Scanner-port-M_i (i.e., zmap-23-M_i). The procedure to cluster Hidden Markov Models and obtain the collection of candidate models is presented in Algorithm 6.4.

This Sub-Section furnished a list of *candidate models*, each provided with a label of the kind "scanner-port-M_i". What is still needed is to see if they are able to recognize Network Scanners from sets of logs. To accomplish this task, *test samples* generated in Sub-Section 6.2.1 are used, and the procedure to follow has been presented in Sub-Section 6.2.3.

6.2.3 Recognition of Network Scanners

The previous Sub-Section 6.2.2 provided a set of *candidate modes*, that will be here leveraged to fingerprint Network Scanners from the *test samples* collected and generated in Sub-Section 6.2.1, and labeled like "scanner-port-TS_i". It follows obviously that all the *candidate models* are

Algorithm 6.4 Split *ZMap* and *Shodan* into datasets according to ports

```
    Input $LS_i$
    Output Clusters of models
$n \leftarrow$ number of $LS_i$
for $i = 1 : n$ do
    for $j = 1 : n$ do
        fit $LS_i$ with $HMM_j$
        llik$[i, j]$ $\leftarrow$loglik$(HMM_j(LS_i))$
    end for
end for
nllik = normalize(llik)
DBSCAN$(nllik, \epsilon$, min-number-elements) $\rightarrow$ clusters of models
```

here validated on *test samples*. For each of them, the Network Scanner that originated them and the targeted port are known. The initial part of its label, “scanner-port”, is, from here on, referred as the *true label* of the considered test sample. The reason of this definition lies in the fact that, later, we will define the so-called *predicted labels*. Each *test sample* is now fitted to each *candidate model*, and the log-likelihood computed. All the values of the log-likelihoods obtained for each *test sample* are compared, and the model that provided the highest one is the one that best describes the given *test sample*. Recalling that each Hidden Markov Model is labeled as “scanner-port-M_j”, the first part of its label, “scanner-port” is stored as the *predicted label* of the considered *test sample*. The last step of the proposed approach is to investigate if it is able to identify the Network Scanner that originated the *test sample*, and eventually the targeted port. More simply, we checked if the *true label* and the *predicted label* of each *test sample* are equal. Accuracy of the classification is then computed by counting how many times *predicted labels* are equal to the *true* ones, and dividing the obtained value for the number of all samples.

$$\text{accuracy} = \frac{|\text{predicted label} = \text{true label}|}{|\text{test samples}|} \tag{6.1}$$

The approach we applied to model and fingerprint the considered scanner has been here detailed. Section 6.3 introduces results we obtained when applying the proposed approach to spatial movements of Network Scanners. Section 6.4 instead presents results obtained when the approach we proposed has been applied to their temporal movements.

6.3 Differences of consecutively scanned IP addresses

The approach applied here has been developed and detailed in Section 6.2. Here, we present results that it produces when applied to spatial movements of Network Scanners. In other words, this section shows how spatial movements of Network Scanners, i.e., differences of two IP addresses consecutively probed, can be modeled from the point of view of the targeted network, and how the obtained models can be used to recognize future Network Scanners from freshly collected logs.

All the candidate HMMs obtained with the procedure detailed in Section 6.2 have a number of states between 1 and 3, of which 91,3% have 2 states, as shown in Figure 6.5. Logs of *learning samples* have been thus clustered into 2 groups by the underlying FMMs, which correspond mainly to backward or forward spatial movements over destination IP addresses, respectively.

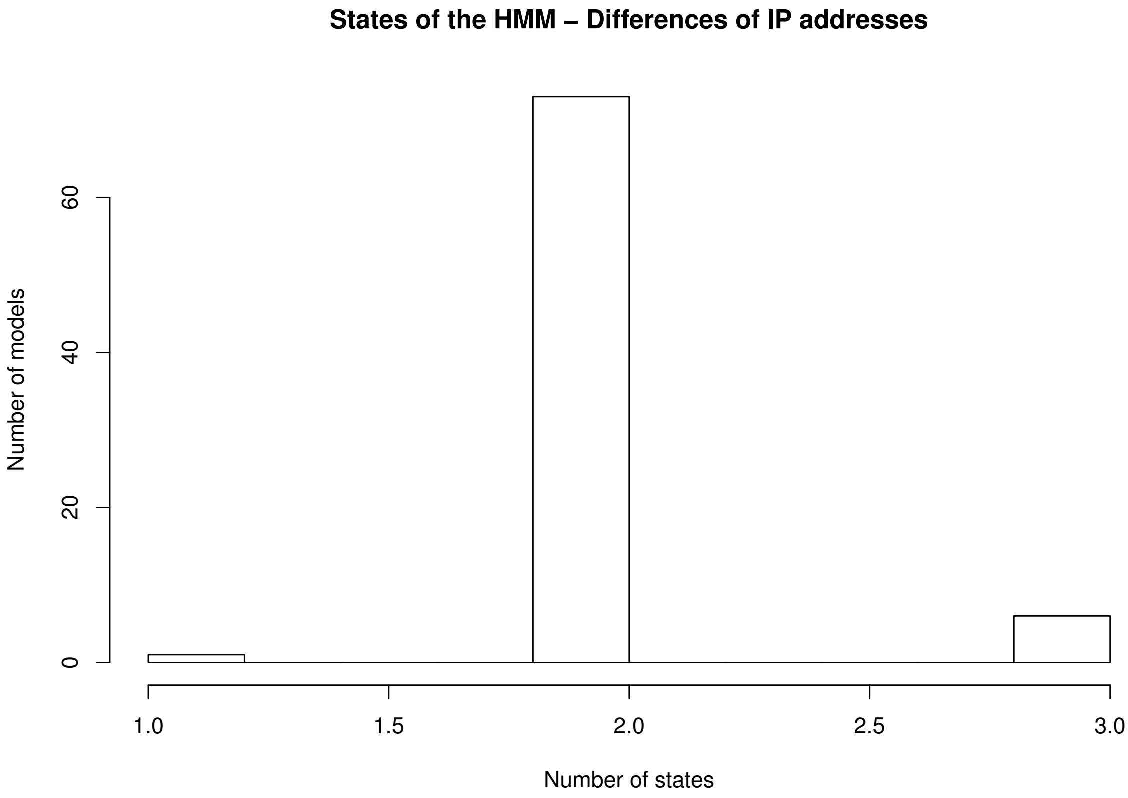
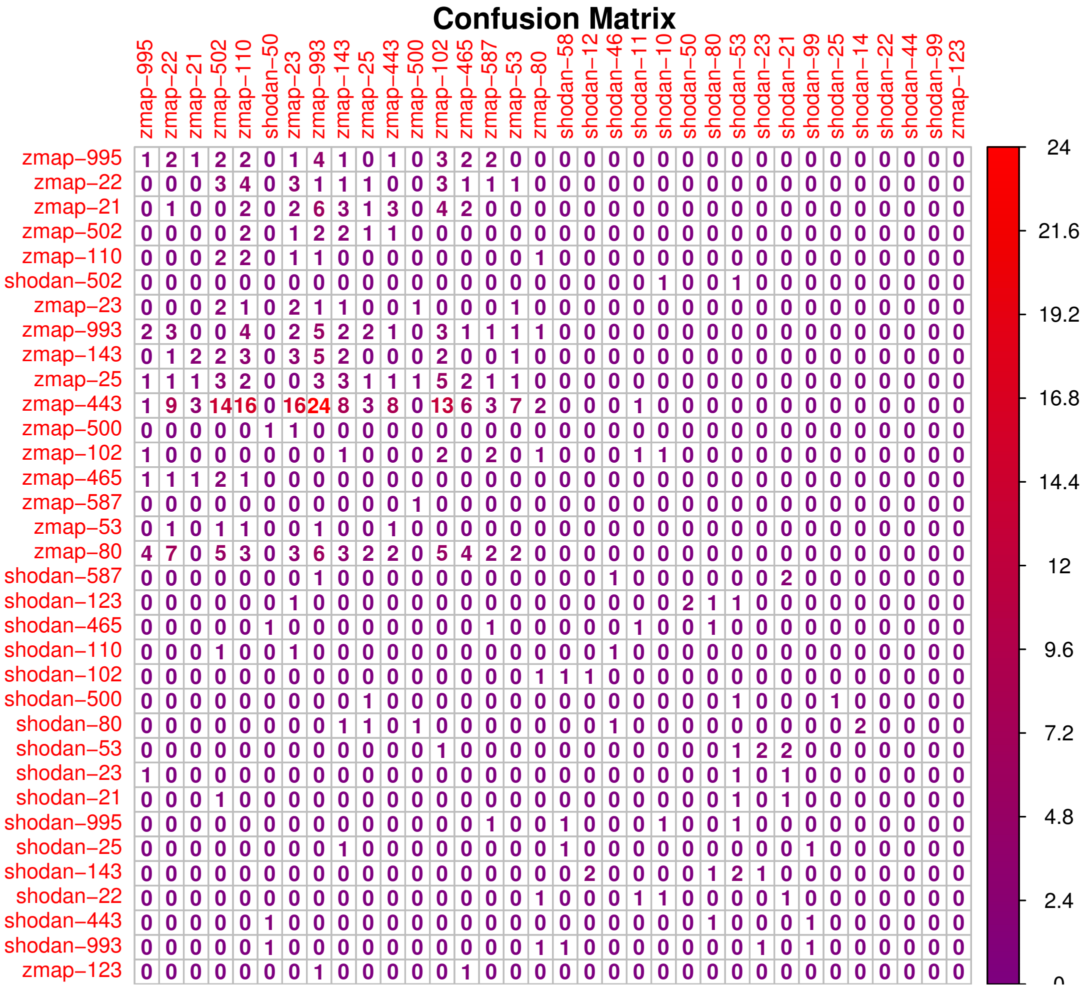


Figure 6.5: Numbers of states of candidate HMMs for differences of IP addresses

All the *candidate models* have been then fitted to all the *test samples*, as described in Sub-Section 6.2.3 and the log-likelihoods stored and compared. The label "scanner-port" of *candidate models* that outputted the highest log-likelihood is stored as the *predicted label* of the given *test sample*.

To compute the accuracy of the proposed approach, we counted how many times the *true label* of the *test sample* is equal to its *predicted label*. Confusion matrices are shown in Figures 6.6 and 6.7, which represent how many test samples have been correctly and wrongly labeled when fingerprinting Network Scanners. In both figures, rows are reserved for *true labels* of *test samples*, and columns for their *predicted labels*. Elements of each row, then, count how many times a *test sample*, whose *true label* is the one of the chosen row, has been labeled with each *predicted label*, one for each column. For example, let us look at Figure 6.6 which contains the confusion matrix of the classification of samples using differences of probed IP addresses. More precisely, let us look at the row labeled "zmap-443": 1 test sample generated by ZMap scanning port 443 has been labeled with the predicted label "zmap-995", 9 with label "zmap-22", 3 with "zmap-21", and so on. Only 8 have been correctly labeled with the predicted label "zmap-443". Row "zmap-443" of Figure 6.6 also shows that the number of test samples generated by ZMap targeting port 443 are more than the others, since the sum of elements in the row reserved to the true label "zmap-443" is higher than other sums of elements of other rows.

Let us look at Figure 6.6, and more in detail at its diagonal. If the proposed approach would be able to fingerprint the Network Scanner and the targeted port, these elements would be much higher than other elements of rows. This, because the counters of how many times each *true label*



	zmap–995	zmap–22	zmap–21	zmap–502	zmap–110	shodan–50	zmap–23	zmap–993	zmap–143	zmap–25	zmap–443	zmap–500	zmap–102	zmap–465	zmap–587	zmap–53	zmap–80	shodan–58	shodan–12	shodan–46	shodan–11	shodan–10	shodan–50	shodan–80	shodan–53	shodan–23	shodan–21	shodan–99	shodan–25	shodan–14	shodan–22	shodan–44	shodan–99	zmap–123
zmap–995	1	2	1	2	2	0	1	4	1	0	1	0	3	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap–22	0	0	0	3	4	0	3	1	1	1	0	0	3	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap–21	0	1	0	0	2	0	2	6	3	1	3	0	4	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap–502	0	0	0	0	2	0	1	2	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap–110	0	0	0	2	2	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
shodan–502	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0
zmap–23	0	0	0	2	1	0	2	1	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap–993	2	3	0	0	4	0	2	5	2	2	1	0	3	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap–143	0	1	2	2	3	0	3	5	2	0	0	0	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap–25	1	1	1	3	2	0	0	3	3	1	1	1	5	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap–443	1	9	3	14	16	0	16	24	8	3	8	0	13	6	3	7	2	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap–500	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap–102	1	0	0	0	0	0	0	0	1	0	0	0	2	0	2	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
zmap–465	1	1	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap–587	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap–53	0	1	0	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap–80	4	7	0	5	3	0	3	6	3	2	2	0	5	4	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
shodan–587	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	2	0	0	0	0	0	0	0
shodan–123	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	1	0	0	0	0	0	0	0	0	0
shodan–465	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
shodan–110	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
shodan–102	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
shodan–500	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
shodan–80	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	2	0	0	0	0
shodan–53	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	2	2	0	0	0	0	0	0	0
shodan–23	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0
shodan–21	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0
shodan–995	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0
shodan–25	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
shodan–143	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	1	2	1	0	0	0	0	0	0	0	0
shodan–22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0
shodan–443	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0
shodan–993	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0
zmap–123	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6.6: Confusion matrix of the classification of samples using differences of IP addresses per scanner-port

	zmap	**shodan**
zmap	368	4
shodan	17	50

Figure 6.7: Confusion matrix of Zmap and Shodan classification using differences of IP addresses per scanner

"scanner-port" is equal to the corresponding *predicted* one would be high. Note that it is true that we know the targeted service from logs. But, we inferred the Hidden Markov Models without leveraging this information, and we wanted to investigate if Network Scanners adapt their spatial behavior with respect to the targeted service. Figure 6.6 shows that the methodology detailed in this chapter is not able to accurately classify the *test samples* according to the couple (scanned port, scanner), i.e. the Network Scanner does not adapt its behavior with respect to the targeted service. If we look at Figure 6.7, instead the two elements of its diagonal are (much) higher than the other elements of its rows. This means that the methodology we deployed and detailed in Section 6.2 is able to detect what Network Scanner originated probes whose logs are collected in *test samples*.

Using the confusion matrix, the accuracy is computed by summing elements of the diagonals and dividing the obtained sum by the number of all samples (i.e, the sum of all elements of the matrices). When assessing both the scanning tool generating logs of the sample and the targeted port, accuracy is really low (0.06). But, when focusing only on the detection of the Network Scanner that originated the probes in test samples, it reaches 0.952. More in detail, only 21 over 439 *test samples* have been wrongly classified: 17 of them have been generated by Shodan and labeled as being originated by ZMap, whereas 4 have been assigned to Shodan while being truly generated by ZMap. It is possible then, with an accuracy of 95%, to state what scanner originated logs in *test samples*, and that spatial movements of the considered scanners do not depend on the targeted port, and differ only between different scanners. Let us look once more at Figure 6.7: despite the overall accuracy is 95%, only 1% of all *test samples* truly generated by ZMap have been wrongly labeled. This percentage rises to 25.4% for Shodan. This suggests that the approach we proposed fingerprints ZMap with more accuracy than Shodan, when applied to their spatial movements.

We investigated if the approach provides better results when applied to temporal movements of the considered Network Scanners. Results are presented in Section 6.4.

6.4 Time gaps between consecutive probes

Previous Section 6.3 applied the approach we proposed in Section 6.2 to learn Hidden Markov Models of ZMap and Shodan using their spatial movements. The *candidate models* have been then fitted to *test samples*, obtained in Sub-Section 6.2.1, to fingerprint the two considered Network Scanners. Our approach provided an accuracy of 95% when applied to spatial movements of Network Scanners. More precisely, the accuracy when fingerprinting ZMap is 99%, whereas it drops to 74.6% when fingerprinting Shodan. This suggested us to investigate if another observation would provide more homogeneous results, or, in other words, better results for Shodan. For this reason, we applied again the approach detailed in Section 6.2, but to differences of consecutive time stamps of successive probes originated by ZMap and Shodan, i.e., to their temporal movements.

Differently from the previous case, detailed in Section 6.3, selected *candidate models* for differences of timestamps have between 1 and 7 states, with 92.2% of them having up to 4 states, as it is possible to see in Figure 6.8. They have been learned from the same *learning samples* obtained in Sub-Section 6.2.1), and leveraged to learn HMMs for differences of IP addresses in Section 6.3. Recall that each model is provided with a label of the kind "scanner-port-M_i", which recalls what scanner was deployed to generate the *learning sample* on which the HMM was learned, and the service it targeted. This label, as done in Section 6.3, becomes useful when computing accuracy of the approach. Each *test sample*, obtained in Sub-Section 6.2.1, has been then fitted to *all* the obtained *candidate models*, and *each* corresponding log-likelihood computed. The model that corresponds to the highest log-likelihood is then stored to be the one that best fits the considered *test sample*. Its label, more precisely its initial part, "scanner-port", is stored as the *predicted label* of the given *test sample*.

To compute the accuracy of the proposed approach when applied to temporal movements of the scanners, as done in Section 6.3 for spatial movements, we counted how many times the *predicted label* of the *test sample* is equal to its *true label*. We checked if the approach is able to detect what scanner originated logs in the *test sample*, and if it detects also what service was targeted by the Network Scanner. Recall that, despite logs contain the information regarding

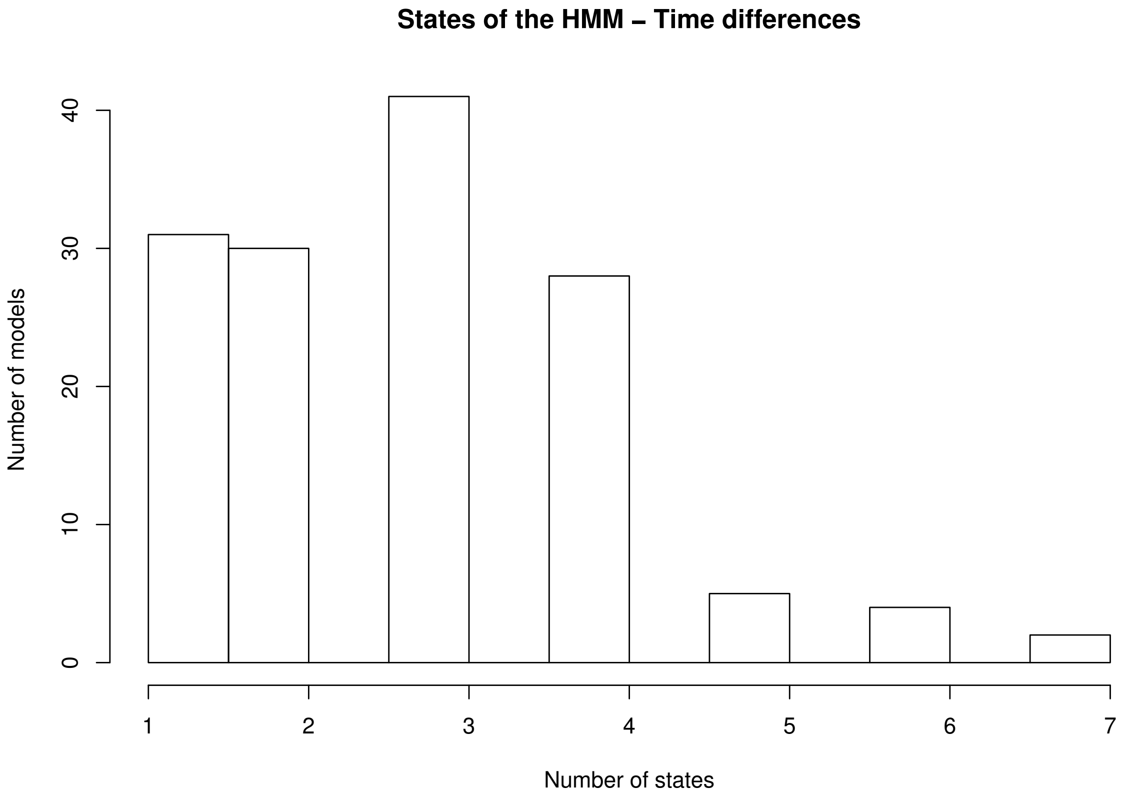
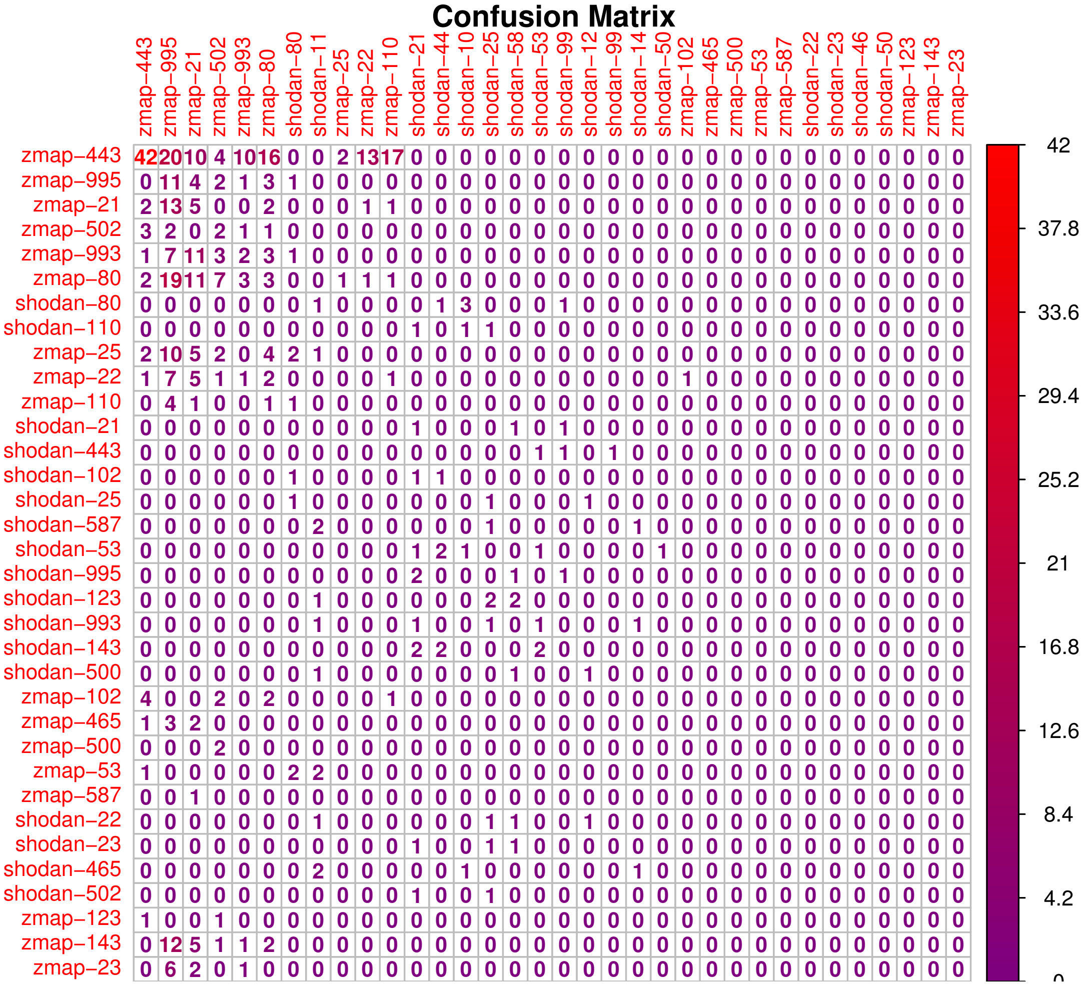


Figure 6.8: Numbers of states of candidate HMMs of time differences.

the targeted port, models have been learned only by leveraging time gaps between consecutive probes. If the approach is able, only by using this information, to detect what service was probed by the Network Scanner, it would mean that it adapts its behavior with respect to the scanned port.

Results are shown in Figure 6.9, which counts how many test samples have been correctly labeled by the approach, also with respect to the targeted port, and in Figure 6.10, which does not take into account the probed service. In the first case, the accuracy is low: only 16% of *test samples* have been correctly labeled. But, in the second case, it rises up to 98%. This brings to the conclusion that ZMap and Shodan don't change their temporal behavior when targeting a particular port, but that temporal movements are inner features of the Network Scanners, enough to fingerprint them.

Let us investigate a bit more the 98% of accuracy. Is there a difference when fingerprinting ZMap and Shodan respectively? In other words, do we have the same phenomenon detailed at the end of Section 6.3? The answer can be found by looking at Figure 6.10, which states that *test samples* originated by Shodan have been *all* correctly labeled. The accuracy with respect to Shodan is then equal to 100%. About ZMap, instead, only 10 *test samples* over 372 have been wrongly labeled. It means that the accuracy of the approach applied to fingerprint ZMap is 97.3%, high enough to state the goodness of the proposed methodology.



	zmap−443	zmap−995	zmap−21	zmap−502	zmap−993	zmap−80	shodan−80	shodan−11	zmap−25	zmap−22	zmap−110	shodan−21	shodan−44	shodan−10	shodan−25	shodan−58	shodan−53	shodan−99	shodan−12	shodan−99	shodan−14	shodan−50	zmap−102	zmap−465	zmap−500	zmap−53	zmap−587	shodan−22	shodan−23	shodan−46	shodan−50	zmap−123	zmap−143	zmap−23
zmap−443	42	20	10	4	10	16	0	0	2	13	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap−995	0	11	4	2	1	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap−21	2	13	5	0	0	2	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap−502	3	2	0	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap−993	1	7	11	3	2	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap−80	2	19	11	7	3	3	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
shodan−80	0	0	0	0	0	0	0	1	0	0	0	0	1	3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
shodan−110	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap−25	2	10	5	2	0	4	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap−22	1	7	5	1	1	2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
zmap−110	0	4	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
shodan−21	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
shodan−443	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
shodan−102	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
shodan−25	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
shodan−587	0	0	0	0	0	0	0	2	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
shodan−53	0	0	0	0	0	0	0	0	0	0	0	1	2	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
shodan−995	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
shodan−123	0	0	0	0	0	0	0	1	0	0	0	0	0	0	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
shodan−993	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
shodan−143	0	0	0	0	0	0	0	0	0	0	0	2	2	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
shodan−500	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap−102	4	0	0	2	0	2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap−465	1	3	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap−500	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap−53	1	0	0	0	0	0	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap−587	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
shodan−22	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
shodan−23	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
shodan−465	0	0	0	0	0	0	0	2	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
shodan−502	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap−123	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap−143	0	12	5	1	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
zmap−23	0	6	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6.9: Confusion matrix of the classification of samples using differences of timestamps per scanner-port

	zmap	**shodan**
zmap	362	10
shodan	0	67

Figure 6.10: Confusion matrix of ZMap and Shodan classification using differences of timestamps per scanner

6.5 Summary

This chapter described the techniques we developed and applied to model Network Scanners and and recognize them with an accuracy up to 100%. More precisely, we developed a methodology in Section 6.2 to have samples containing logs of single executions of the considered Network Scanners (see Section 6.2.1). Some of them, the so-called *learning samples*, have been leveraged to learn Hidden Markov Models, with a procedure detailed in Sub-Section 6.2.2. Results are detailed in Sections 6.3 and 6.4. *Test samples* have been instead used to recognize Network Scanners (see Sub-Sections 6.2.3), and to compute the accuracy of the approach, as seen in Sections 6.3 and 6.4. Both study cases showed that the behavior of Network Scanners is not related to the targeted service, but that their movements, both spatial and temporal, suffice to fingerprint what scanner originated logs. This could lead to the conclusion that attackers, once chosen the Network Scanner to deploy, use the same configurations when scanning various ports.

There are various scenarios and possibilities to enhance and continue on the same lines of this work. The first road to take would be to investigate why, as seen in Section 6.3, the approach proposed in Section 6.2 provides optimal results when applied to spatial movements of ZMap, but only fairly good ones (accuracy of 74.6%) when applied to spatial movements of Shodan. Is it related to an intrinsic feature of Shodan? Or does the approach need to be enhanced or adapted when applied to this case? Another road to follow would be to create Hidden Markov Models that model both movements of Network Scanners, in order to check if they provide better results and higher accuracy, especially than the one obtained in Section 6.3 for spatial movements of Shodan.

7

General Conclusions

7.1 Conclusions

This PhD addresses the problem of recognizing what Network Scanner generated the probing packets collected by the monitored network. This issue is consequent to the assessment of whether the observed network is targeted by a Network Scanning Activity. For this reason, we detailed Network Scanner Activities in Chapter 2, and provided various approaches to detect them in Chapter 3. Since the actual literature, at the best of our knowledge, is poor in providing tools or approaches to recognize what Network Scanner is deployed by performers of the detected Network Scanning Activity, we provided an architecture that fills this gap. More in detail, the methodology we developed and proposed here leverages Hidden Markov Models to model two Network Scanners, ZMap and Shodan. The obtained models have been then leveraged to recognize the Network Scanner that originated freshly collected probes.

7.2 Achievements

We provided models based on long- and short-term analyses, respectively, of the two considered Network Scanners. Long-term analyses focused on intensities of the Network Scanners, i.e. on the number of probes received by the monitored network and originated by the considered Network Scanner within intervals of time of a fixed length. The Hidden Markov Models based on intensities of Network Scanners showed that, despite it is a feature that suffices to characterize and distinguish Network Scanners, the amount of data which is needed to build the models require an amount of time to be collected that makes it difficult, for security experts, to leverage the obtained models and results in (almost) real time. For this reason, we moved our attention to short-term analyses, by modeling, with Hidden Markov Models, spatial and temporal movements of ZMap and Shodan. The obtained models have been then leveraged to recognize the Network Scanner that originated freshly collected probes, with an accuracy that reached, in some cases, the 100%.

7.3 Future Work

Because of time constraints within this PhD some research questions remain still to be faced. The first one is to investigate the reasons why Zmap varies its intensity with respect to the probed ports. A naive and quick answer to this is that some services (i.e. ports) are more interesting than others. The *real* question is: “Why are some services more interesting than others?”. As detailed in Chapter 2, interests for some services never fade off, whereas for others it flares up and then decreases. The second case usually occurs when a vulnerability of the service is found. So, the interesting work that we hope to perform in the future is to motivate the interest behind an increase of intensity of probes directed to a given port.

Another research question is to assess if Shodan is able to have a behavior similar to the one of Shodan: can it probe with higher intensities vulnerable services?

Let us not consider the Finite Mixture Models created on temporal movements, which group logs into a number of clusters between 1 to 7. Although the meaning of clusters in case of spatial movements is clear (one for forward, the other for backward movements), a characterization for the clusters provided by Finite Mixture Models is desired.

The lase desired work would be to create Hidden Markov Models that are able to describe both movements of the Network Scanners, in order to increase the accuracy of the recognizing process.

Appendices

A

Telnet and SSH

Port 23 is reserved for Telnet, which is a protocol developed in 1969 whose name stands for "*tel*etype *net*work". It is used on the Internet or local area networks to provide an interactive, bidirectional and text-oriented communication through the usage of a virtual terminal connection. More precisely, Telnet is a client-server protocol which is based on a reliable connection-oriented transport. It is thus used to establish a connection to TCP port 23, and provides access to a command-line interface of an operating system on a remote host. As mentioned few lines above, Telnet was developed in 1969, when the majority of users of computers connected to the network were members of computer departments of universities, public or private centers of research or other academic institutions. This means that there were no reasons to worry about security, which became a concern in the 1990s with the explosion of the bandwidth and, consequently, of the number of people connected to the Internet. The rise of the latter coincided also with the increasing of the number of hackers of servers of other people. For this reason, it became necessary to have encrypted alternatives, recommended by experts in computer security (like the SANS istitute) for remote logins under all circumstances. They motivate this warning with the following reasons:

- Telnet does not encrypt any data sent over the connection (passwords included). Then, it is relatively easy (and, at least, feasible) to eavesdrop on communications and use the passwords later for malicious purposes. It suffices to have access to a router, switch, hub or gateway located on the network between two hosts using Telnet, to intercept packets and obtain whatever typed, by leveraging a packet analyzer.

- The majority of implementations of Telnet have no authentication which could ensure that communication takes place between the two desired hosts and it is not intercepted in the middle by a sneaker.

- Over the years many vulnerabilities have been discovered in Telnet daemons that are commonly used.

All these motivated the quick drop of usage of the Telnet protocol, especially on the public Internet. It has been practically replaced by the Secure Shell (SSH) protocol, released in 1995. Telnet is still used nowadays, but only to access old legacy equipment that does not support more modern protocols, like many industrial and scientific devices which have only Telnet available as a communication option.

SSH enhances Telnet with a strong encryption, which aims at avoiding interception of sensitive data, and public key authentication, that ensures that the remote control is really who it claims to be. More in detail, SSH is a cryptographic network protocol able to operate network services in a secure manner over an unsecured network. An example is remote login to computer systems carried out by users. Said in other words, SSH provides a *secure channel* over an unsecured network, in a client-server architecture, by connecting an SSH client application with an SSH server. This protocol is commonly leveraged for command-line logins and remote command executions, but it is worth to note that any network service can be secured with SSH. It was designed to replace Telnet and unsecured remote shell protocols, that send info in plain text. This renders them at risk of interception and disclosure by leveraging packet analysis. The encryption provided by SSH provides confidentiality and integrity of data over an unsecured network (like the Internet) even if the American National Security Agency (NSA) can decrypt SSH, thus read contents of SSH sessions. The remote computer is authenticated by SSH through a public key cryptography. There is also the possibility to authenticate the user.

There exist several ways to use SSH.

- The most common one uses public-private key pairs, that are automatically generated, to encrypt a network connection. To log on, a password authentication comes into help.

- Another way performs the authentication by leveraging a public-private key pair that is generated manually. In this case, users and/or programs do not need passwords to log in. In this scenario, a matching pair of public and private keys can be generated by anyone. The public key is then placed on all computers that must allow access to who owns the matching private key, that is kept secret by the owner: it is never transferred through the network during authentication, even if authentication is based on it. SSH only verifies whether the public key is offered by the same person that owns the matching private key.

  The important point, in all SSH versions, is to associate public keys with identities, i.e. to verify unknown public keys, before accepting them as valid. The reason of this lies in the fact that if the public key of an attacker is accepted without validation, then an unauthorized attacker will be authorized as a valid user.

SSH servers are contacted through the standard TCP port 22.

Bibliography

[1] E. Bou-Harb, M. Debbabi, and C. Assi, "Cyber Scanning: a Comprehensive Survey," *Ieee communications surveys & tutorials*, vol. 16, no. 3, pp. 1496–1519, 2014.

[2] T. M. Chen and S. Abu-Nimeh, "Lessons from Stuxnet," *Computer*, vol. 44, no. 4, pp. 91–93, 2011.

[3] Kaspersky, "Carbanak APT. The Great Robbery," https://securelist.com/files/2015/02/Carbanak_APT February 2015, accessed: 2016-01-12.

[4] E. Raftopoulos, E. Glatz, X. Dimitropoulos, and A. Dainotti, "How Dangerous is Internet Scanning?" in *International Workshop on Traffic Monitoring and Analysis.* Springer, 2015, pp. 158–172.

[5] Z. Durumeric, M. Bailey, and J. A. Halderman, "An Internet-Wide View of Internet-Wide Scanning," in *23rd USENIX Security Symposium (USENIX Security)*, 2014, pp. 65–78.

[6] S. Panjwani, S. Tan, K. M. Jarrin, and M. Cukier, "An Experimental Evaluation to Determine if Port Scans are Precursors to an Attack," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN).* IEEE, 2005, pp. 602–611.

[7] A. K. Kaushik, E. S. Pilli, and R. Joshi, "Network Forensic System for Port Scanning Attack," in *Advance Computing Conference (IACC), 2010 IEEE 2nd International.* IEEE, 2010, pp. 310–315.

[8] E. Bou-Harb, M. Debbabi, and C. Assi, "A Time Series Approach for Inferring Orchestrated Probing Campaigns by Analyzing Darknet Traffic," in *10th International Conference on Availability, Reliability and Security (ARES).* IEEE, 2015, pp. 180–185.

[9] V. Ghiette, N. Blenn, and C. Doerr, "Remote Identification of Port Scan Toolchains," in *8th IFIP International Conference on New Technologies, Mobility and Security (NTMS).* IEEE, 2016, pp. 1–5.

[10] P. Chen, L. Desmet, and C. Huygens, "A Study on Advanced Persistent Threats," in *Communications and Multimedia Security*, ser. Lecture Notes in Computer Science, B. De Decker and A. Zúquete, Eds. Springer Berlin Heidelberg, 2014, vol. 8735, pp. 63–72. [Online]. Available: http://dx.doi.org/10.1007/978-3-662-44885-4_5

[11] S. Lagraa and J. François, "Knowledge Discovery of Port Scans from Darknet," in *Symposium on Integrated Network and Service Management, IFIP/IEEE.* IEEE, 2017, pp. 935–940.

[12] R. J. Barnett and B. Irwin, "Towards a Taxonomy of Network Scanning Techniques," in *Proceedings of the 2008 annual research conference of the South African Institute of Computer*

Scientists and Information Technologists on IT research in developing countries: riding the wave of technology. ACM, 2008, pp. 1–7.

[13] E. Bou-Harb, M. Debbabi, and C. Assi, "On Fingerprinting Probing Activities," *Computers & Security*, vol. 43, pp. 35–48, 2014.

[14] J. Mazel, R. Fontugne, and K. Fukuda, "Profiling Internet Scanners: Spatial and Temporal Structures," 2016.

[15] M. De Vivo, E. Carrasco, G. Isern, and G. O. de Vivo, "A Review of PortScanning Techniques," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 2, pp. 41–48, 1999.

[16] M. Allman, V. Paxson, and J. Terrell, "A Brief History of Scanning," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement.* ACM, 2007, pp. 77–82.

[17] D. Leonard and D. Loguinov, "Demystifying Internet-Wide Service Discovery," *IEEE/ACM Transactions on Networking*, vol. 21, no. 6, pp. 1760–1773, 2013.

[18] C. Gates, "Coordinated Scan Detection." in *NDSS*, 2009.

[19] P. K. Manna, S. Chen, and S. Ranka, "Exact Modeling of Propagation for Permutation-Scanning Worms," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE.* IEEE, 2008, pp. 1696–1704.

[20] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast Internet-Wide Scanning and its Security Applications." in *Usenix Security*, 2013.

[21] G. F. Lyon, *NMap Network Scanning: The Official NMap Project Guide to Network Discovery and Security Scanning.* Insecure, 2009.

[22] "Nmap In The Movies," https://nmap.org/movies/, accessed: 2018-09-20.

[23] "Nmap," https://nmap.org/, accessed: 2018-09-20.

[24] "Shodan," https://www.shodan.io/, accessed: 2018-09-20.

[25] R. Bodenheim, J. Butts, S. Dunlap, and B. Mullins, "Evaluation of the Ability of the Shodan Search Engine to Identify Internet-Facing Industrial Control Devices," *International Journal of Critical Infrastructure Protection*, vol. 7, no. 2, pp. 114–123, 2014.

[26] R. O'Harrow Jr., "Cyber Search Engine Shodan Exposes Industrial Control Systems to new Risks," *The Washington Post*, June 2012. [Online]. Available: https://www.washingtonpost.com/investigations/cyber-search-engine-exposes-vulnerabilities/2012/06/03/gJQAIK9KCV_story.html?noredirect=on&utm_term=.94b229756b66

[27] D. Goldman, "Shodan: The Scariest Search Engine on the Internet," *CNN Tech*, April 2013. [Online]. Available: https://money.cnn.com/2013/04/08/technology/security/shodan/

[28] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle, "GrIDS - A Graph Based Intrusion Detection System for Large Networks," in *Proceedings of the 19th national information systems security conference*, vol. 1. Baltimore, 1996, pp. 361–370.

[29] M. H. Bhuyan, D. Bhattacharyya, and J. K. Kalita, "Surveying Port Scans and their Detection Methodologies," *The Computer Journal*, vol. 54, no. 10, pp. 1565–1581, 2011.

[30] N. Kato, H. Nitou, K. Ohta, G. Mansfield, and Y. Nemoto, "A Real-Time Intrusion Detection System (IDS) for Large Scale Networks and its Evaluations," *IEICE Transactions on Ccommunications*, vol. 82, no. 11, pp. 1817–1825, 1999.

[31] P. A. Porras and A. Valdes, "Live Traffic Analysis of TCP/IP Gateways." in *NDSS*, 1998.

[32] C. Leckie and R. Kotagiri, "A Probabilistic Approach to Detecting Network Scans," in *Network Operations and Management Symposium, 2002. NOMS 2002. 2002 IEEE/IFIP.* IEEE, 2002, pp. 359–372.

[33] S. Robertson, E. V. Siegel, M. Miller, and S. J. Stolfo, "Surveillance Detection in High Bandwidth Environments," in *DARPA Information Survivability Conference and Exposition, 2003. Proceedings*, vol. 1. IEEE, 2003, pp. 130–138.

[34] L. Ertoz, E. Eilertson, A. Lazarevic, P.-N. Tan, P. Dokas, V. Kumar, and J. Srivastava, "Detection of Novel Network Attacks using Data Mining," in *Proc. of Workshop on Data Mining for Computer Security.* Citeseer, 2003.

[35] H. Kim, S. Kim, M. A. Kouritzin, and W. Sun, "Detecting Network Portscans through Anomoly Detection," in *Signal Processing, Sensor Fusion, and Target Recognition XIII*, vol. 5429. International Society for Optics and Photonics, 2004, pp. 254–264.

[36] G.-H. K. Haan, "Detection of Portscans Using IP Header Data," *Proceedings of TBRC'05*, 2005.

[37] C. Gates, J. J. McNutt, J. B. Kadane, and M. I. Kellner, "Scan Detection on very Large Networks using Logistic Regression Modeling," in *11th IEEE Symposium on Computers and Communications, ISCC.* IEEE, 2006, pp. 402–408.

[38] G. J. Simon, H. Xiong, E. Eilertson, and V. Kumar, "Scan Detection: A Data Mining Approach," in *Proceedings of the 2006 SIAM International Conference on Data Mining.* SIAM, 2006, pp. 118–129. [Online]. Available: https://doi.org/10.1137/1.9781611972764.11

[39] M. Roesch *et al.*, "Snort: Lightweight Intrusion Detection for Networks." in *Lisa*, vol. 99, no. 1, 1999, pp. 229–238.

[40] J. Udhayan, A. Krishnan, R. Anitha *et al.*, "Reconnaissance Scan Detection Heuristics to Disrupt the Pre-Attack Information Gathering," in *International Conference on Network and Service Security, N2S.* IEEE, 2009, pp. 1–5.

[41] L. T. Heberlein, G. V. Dias, K. N. Levitt, B. Mukherjee, J. Wood, and D. Wolber, "A Network Security Monitor," in *Computer Society Symposium on Research in Security and Privacy.* IEEE, 1990, pp. 296–304.

[42] V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time," *Computer Networks*, vol. 31, no. 23-24, pp. 2435–2463, 1999.

[43] M. Fullmer and S. Romig, "The OSU Flowtools Package and CISCO NetFlow Logs," in *Proceedings of the 2000 USENIX LISA Conference*, 2000.

[44] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan, "Fast Portscan Detection Using Sequential Hypothesis Testing," in *Symposium on Security and Privacy.* IEEE, 2004, pp. 211–225.

[45] S. Kong, T. He, X. Shao, C. An, and X. Li, "Scalable Double Filter Structure for Port Scan Detection," in *International Conference on Communications, ICC.*, vol. 5. IEEE, 2006, pp. 2177–2182.

[46] Y. Zhang and B. Fang, "A Novel Approach to Scan Detection on the Backbone," in *Sixth International Conference on Information Technology: New Generations, ITNG.* IEEE, 2009, pp. 16–21.

[47] A. Sridharan, T. Ye, and S. Bhattacharyya, "Connectionless Port Scan Detection on the Backbone," in *25th IEEE International Performance, Computing, and Communications Conference, IPCCC.* IEEE, 2006, pp. 10–.

[48] R. Basu, R. K. Cunningham, S. E. Webster, and P. Lippmann, "Detecting Low-Profile Probes and Novel Denial-of-Service Attacks," in *Proceedings of the 2001 IEEE Workshop on Information Assurance, US Military Academy.* Citeseer, 2001.

[49] W. W. Streilein, R. K. Cunningham, and S. E. Webster, "Improved Detection of Low-Profile Probe and Denial-of-Service Attacks," in *Proceedings of the 2001 Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection*, 2001.

[50] W. El-Hajj, F. Aloul, Z. Trabelsi, and N. Zaki, "On Detecting Port Scanning Using Fuzzy Based Intrusion Detection System," in *International Wireless Communications and Mobile Computing Conference, IWCMC.* IEEE, 2008, pp. 105–110.

[51] D.-p. Liu, M.-w. Zhang, and T. Li, "Network Traffic Analysis Using Refined Bayesian Reasoning to Detect Flooding and Port Scan Attacks," in *International Conference on Advanced Computer Theory and Engineering, ICACTE.* IEEE, 2008, pp. 1000–1004.

[52] M. Z. Shafiq, M. Farooq, and S. A. Khayam, "A Comparative Study of Fuzzy Inference Systems, Neural Networks and Adaptive Neuro Fuzzy Inference Systems for Portscan Detection," in *Workshops on Applications of Evolutionary Computation.* Springer, 2008, pp. 52–61.

[53] M. V. Mahoney and P. K. Chan, "PHAD: Ppacket Header Anomaly Detection for Identifying Hostile Network Traffic," Tech. Rep., 2001. [Online]. Available: http://hdl.handle.net/11141/94

[54] J. Kim and J.-H. Lee, "A Slow Port Scan Attack Detection Mechanism Based on Fuzzy Logic and a Stepwise Policy," 2008.

[55] G. Conti and K. Abdullah, "Passive Visual Fingerprinting of Network Attack Tools," in *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security.* ACM, 2004, pp. 45–54.

[56] K. Lakkaraju, W. Yurcik, and A. J. Lee, "NVisionIP: Netflow Visualizations of System State for Security Situational Awareness," in *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security.* ACM, 2004, pp. 65–72.

[57] K. Abdullah, C. Lee, G. Conti, and J. A. Copeland, “Visualizing Network Data for Intrusion Detection,” in *Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC.* IEEE, 2005, pp. 100–108.

[58] A. J. Lee, G. A. Koenig, X. Meng, and W. Yurcik, “Searching for Open Windows and Unlocked Doors: Port Scanning in Large-Scale Commodity Clusters,” in *Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on*, vol. 1. IEEE, 2005, pp. 146–151.

[59] J. McPherson, K.-L. Ma, P. Krystosk, T. Bartoletti, and M. Christensen, “Portvis: a Tool for Port-Based Detection of Security Events,” in *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security.* ACM, 2004, pp. 73–81.

[60] C. Muelder, K.-L. Ma, and T. Bartoletti, “Interactive Visualization for Network and Port Scan Detection,” in *International Workshop on Recent Advances in Intrusion Detection.* Springer, 2005, pp. 265–283.

[61] S. Musa and D. J. Parish, “Visualising Communication Network Security Aattacks,” in *Information Visualization, 2007. IV'07. 11th International Conference.* IEEE, 2007, pp. 726–733.

[62] Z. Jiawan, L. Liang, L. Liangfu, and Z. Ning, “A Novel Visualization Approach for Efficient Network Scans Detection,” in *Security Technology, 2008. SECTECH'08. International Conference on.* IEEE, 2008, pp. 23–26.

[63] C. Gates, “Co-ordinated Port Scans: a Model, a Detector and an Evaluation Methodology.” 2006.

[64] D. L. Whyte, *Network Scanning Detection Strategies for Enterprise Networks.* Carleton University Canada, 2008.

[65] S. Staniford, J. A. Hoagland, and J. M. McAlerney, “Practical Automated Detection of Stealthy Portscans,” *Journal of Computer Security*, vol. 10, no. 1-2, pp. 105–136, 2002.

[66] V. Yegneswaran, P. Barford, and J. Ullrich, “Internet Intrusions: Global Characteristics and Prevalence,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 31, no. 1, pp. 138–147, 2003.

[67] C. Carver, J. Hill, J. R. Surdu, and U. W. Pooch, “A Methodology for Using Intelligent Agents to Provide Automated Intrusion Response,” in *Proceedings of the IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop, West Point, NY*, 2000, pp. 110–116.

[68] W. Zhang, S. Teng, and X. Fu, “Scan attack detection based on distributed cooperative model,” in *Computer Supported Cooperative Work in Design, 2008. CSCWD 2008. 12th International Conference on.* IEEE, 2008, pp. 743–748.

[69] L. R. Rabiner, “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition,” *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[70] L. Rabiner and B. Juang, “An Introduction to Hidden Markov Models,” *Acoustic, Speech and Signal Processing (ASSP) Magazine*, vol. 3, no. 1, pp. 4–16, 1986.

[71] W. Zucchini and I. L. MacDonald, *Hidden Markov Models for Time Series: an Introduction using R*, ser. Monographs on statistics and applied probability. Boca Raton: CRC Press, 2009, a Chapman & Hall book. [Online]. Available: http://opac.inria.fr/record=b1132479

[72] M. A. T. Figueiredo and A. K. Jain, "Unsupervised Learning of Finite Mixture Models," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 24, no. 3, pp. 381–396, 2002.

[73] H. Akaike, "A New Look at the Statistical Model Identification," *IEEE transactions on automatic control*, vol. 19, no. 6, pp. 716–723, 1974.

[74] G. e. a. Schwarz, "Estimating the Dimension of a Model," *The annals of statistics*, vol. 6, no. 2, pp. 461–464, 1978.

[75] D. Ramage, "Hidden Markov Models Fundamentals," *Lecture Notes*, 2007, https://see.stanford.edu/materials/aimlcs229/cs229-hmm.pdf.

[76] L. R. Welch, "Hidden Markov Models and the Baum-Welch Algorithm," *Newsletter of the IEEE Information Theory Society*, vol. 53, no. 4, December 2003.

[77] C. Fachkha and M. Debbabi, "Darknet as a Source of Ccyber Intelligence: Survey, Taxonomy, and Characterization," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1197–1227, 2016.

[78] S.-s. Choi, J. Song, S. Kim, and S. Kim, "A Model of Analyzing Cyber Threats Trend and Tracing Potential Attackers Based on Darknet Traffic," *Security and Communication Networks*, vol. 7, no. 10, pp. 1612–1621, 2014.

[79] G. De Santis, A. Lahmadi, J. Francois, and O. Festor, "Modeling of IP Scanning Activities with Hidden Markov Models: Darknet Case Study," in *8th IFIP International Conference on New Technologies, Mobility and Security (NTMS),*. IEEE, 2016, pp. 1–5.

[80] Z. Li, A. Goyal, and Y. Chen, "Honeynet-Based Botnet Scan Traffic Analysis," in *Botnet Detection*. Springer, 2008, pp. 25–44.

[81] G. De Santis, A. Lahmadi, J. Francois, and O. Festor, "Internet-Wide Scanners Classification using Gaussian Mixture and Hidden Markov Models," in *New Technologies, Mobility and Security (NTMS), 2018 9th IFIP International Conference on*. IEEE, 2018, pp. 1–5.

[82] A. C. Berry, "The Accuracy of the Gaussian Approximation to the Sum of Independent Variates," *Transactions of the American Mathematical Society*, vol. 49, no. 1, pp. 122–136, 1941.

[83] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[84] J. Mazel, R. Fontugne, and K. Fukuda, "Identifying Coordination of Network Scans Using Probed Address Structure," in *8th International Workshop on Traffic Monitoring and Analysis, TMA 2016, Louvain la Neuve, Belgium, April 7-8, 2016.*

Résumé

Le travail accompli dans cette thèse a consisté à construire des modèles stochastiques de deux scanners de l'Internet qui sont ZMap et Shodan. Les paquets provenant de chacun des deux scanners ont été collectés par le Laboratoire de Haute Sécurité (LHS) hébergé à Inria Nancy Grand Est, et ont été utilisés pour construire par apprentissage des chaînes de Markov cachées (HMMs). La première partie du travail consistait à modéliser l'intensité des deux scanners considérés. Nous avons cherché à savoir si l'intensité de ZMap varie en fonction du service ciblé et si les intensités des deux scanners sont comparables. Les résultats ont montré que la réponse à la première question est positive (c'est-à-dire que l'intensité de ZMap varie en fonction des ports ciblés), alors que la réponse à la deuxième question est négative. En d'autres termes, nous avons obtenu un modèle pour chaque ensemble de logs. La partie suivante du travail consistait à étudier deux autres caractéristiques des mêmes scanners : leurs mouvements spatio-temporels. Nous avons créé des ensembles d'échantillons de logs avec chacune d'elle contient une seule exécution de ZMap et Shodan. Ensuite, nous avons calculé les différences d'adresses IP ciblées consécutivement par le même scanner (c.-à-d. dans chaque échantillon), et les timestamps correspondants. Les premiers ont été utilisés pour modéliser les mouvements spatiaux, tandis que les seconds pour les mouvements temporels. Une fois que les modèles de chaînes de Markov cachées sont construites, ils ont été appliqués pour identifier les scanners d'autres ensembles de logs. Dans les deux cas, nos modèles ne sont pas capables de détecter le service ciblé, mais ils détectent correctement le scanner qui génère de nouveaux logs, avec une précision de 95% en utilisant les mouvements spatiaux et de 98% pour les mouvements temporels.

Abstract

The work accomplished in this PhD consisted in building stochastic models of ZMap and Shodan, respectively, two Internet-wide scanners. More in detail, packets originated by each of the two considered scanners have been collected by the High Security Lab hosted in Inria, and have been used to learn Hidden Markov Models (HMMs). The first part of the work consisted in modeling intensity of the two considered scanners. We investigated if the intensity of ZMap varies with respect to the targeted service, and if the intensities of the two scanners are comparable. Results showed that the answer to the first question is positive (i.e., intensity of ZMap varied with respect to the targeted ports), whereas the answer to the second question is negative. In other words, we obtained a model for each set of logs.

The following part of the work consisted in investigating other two features of the same scanners: their spatial and temporal movements, respectively. More in detail, we created datasets containing logs of one single execution of ZMap and Shodan, respectively. Then, we computed differences of IP addresses consecutively targeted by the same scanner (i.e., in each sample), and of the corresponding timestamps. The former have been used to model spatial movements, whereas the latter temporal ones. Once the Hidden Markov Models are available, they have been applied to detect scanners from other sets of logs. In both cases, our models are not able to detect the targeted service, but they correctly detect the scanner that originates new logs, with an accuracy of 95% when exploiting spatial movements, and of 98% when using temporal movements.

Résumé

L'énorme croissance d'Internet expose ses utilisateurs à diverses menaces. Tout le monde et tout est connecté quotidiennement : appareils intelligents, imprimantes, chauffage domestique télécommandé, voitures, webcams, systèmes de contrôle de feux tricolores et d'appareils industriels... Ces appareils et leurs logiciels peuvent être sujets à des vulnérabilités, c'est-à-dire des faiblesses qui pourraient être exploitées par des agents malveillants pour les altérer, dégrader les services qu'ils fournissent, ou anéantir les activités dont ils sont capables. Avec 25 milliards de tentatives d'intrusion quotidiennes [1], Internet est de loin l'objet le plus attaqué aujourd'hui. Les informations sur les périphériques connectés sur les services qu'ils peuvent offrir et leur système d'exploitation (le cas échéant) sont collectées par le biais d'activités *Activités d'analyse (ou de sondage) réseau* qui permettent de découvrir en masse les hôtes vulnérables et/ou les vulnérabilités des hôtes ciblés [5]. D'une manière générale, les activités d'analyse et de vérification de réseau sont des techniques de reconnaissance utilisées pour collecter une information au sujet de l'hôte ciblé. Ces activities servent notamment à déterminer les ports de communication ouverts et les services disponibles [6, 1, 7]. Les activités d'analyse de réseau (NSA) sont l'une des techniques les plus populaires pour découvrir et cartographier les services qui sont à l'écoute sur des ports ou hôtes spécifiques. Ils permettent aux attaquants de créer une liste de faiblesses et de vulnérabilités potentielles sur les ports ouverts, menant à l'exploitation et au compromis d'un hôte distant [7]. Il s'agit de techniques de balayage hautement distribuées avec des capacités élevées de coordination et de furtivité composite. Ces caractéristiques rendent toutes les techniques de détection disponibles jusqu'en 2014 irréalisables [1]. Ces techniques sont largement utilisés par les attaquants comme première étape des tentatives d'exploitation [8], qui visent à prendre le contrôle du serveur [9], exploiter les vulnérabilités [5, 8] et recueillir des informations sur le système qui sera successivement la cible d'une cyberattaque. La nécessité de prévenir, de protéger, et d'atténuer les attaques et d'identifier leurs sources ne cesse donc de croître [1, 7].

Dans le cadre de ce doctorat, nous avons décidé d'étudier et de nous concentrer sur les activités d'analyse de réseau. Ce choix repose sur le fait qu'en 2015, environ 50% de toutes les cyberattaques ont été précédées par une forme d'activités d'analyse de réseau [8], qui sont largement et régulièrement saisies par les infrastructures de surveillance, mais peu étudiées [4]. Afin d'assurer une défense efficace, les menaces ciblées et avancées doivent être atténuées le plus tôt possible, c'est-à-dire pendant les phases d'analyse du réseau [9]. Afin de cacher leur identité, et/ou de gagner en vitesse d'analyse et en furtivité, les scanners réseau, comme Wireshark, NMap, ZMap, Masscan et Shodan sont souvent utilisés par les attaquants qui visent à effectuer une activité d'analyse réseau. C'est pour ces raisons que notre travail vise à les modéliser à partir des logs de trafic collectés par le réseau hébergé. Les identifier aide les experts en sécurité et les gestionnaires de réseau à empêcher les cyberattaques de réussir et les vulnérabilités d'être exploitées. Néanmoins, seules quelques organisations détectent les activités d'analyse de réseau et bloquent leurs sondes. Quand cela se produit, c'est souvent par hasard, puisqu'ils sont souvent découverts lors d'autres opérations de maintenance [5]. Seul un travail limité existe pour détecter les activités distribuées d'analyse de réseau, et les méthodes statistiques sont les moins exploitées [1]. Le travail de cette thèse vise à fournir une approche pour reconnaître quel scanner réseau est utilisé pour sonder le réseau hébergé. En d'autres termes, une fois qu'il est indiqué qu'une activité d'analyse de réseau a pris fin ou est en train de sonder le réseau observé, nous sommes en mesure d'identifier quel scanner de réseau a été déployé. Pour cette raison, puisque la seule information disponible est celle qui peut être recueillie à partir du réseau observé (et sondé), la première étape de notre travail a été d'apprendre les modèles de scanners de réseau du point de vue du réseau sondé. Pour cela, une seule caractéristique à la fois des scanners réseau a

été modélisée : l'intensité, les mouvements spatiaux et temporels. L'*Intensité* fait référence au nombre de sondes de balayage reçues par le réseau ciblé dans des fenêtres de temps longues. Les *mouvements spatiaux* font référence aux différences entre les adresses IP du réseau testé et celles du réseau hébergé qui sont testées l'une après l'autre. Les *mouvements temporels* font référence aux différences d'horodatage des sondes consécutives collectées par le réseau hébergé et ciblé. La littérature actuelle est pauvre en méthodes et outils permettant de reconnaître le scanner réseau utilisé pour sonder le réseau hébergé. Nous avons cherché à combler cette lacune en fournissant une technique de classification qui exploite les modèles statistiques obtenus pour reconnaître quel Network Scanner est à l'origine des sondes collectées.

La première partie de notre travail a consisté à donner un aperçu général des activités de Network Scanning après avoir fourni leur définition : Un *Activité d'analyse de réseau* est une "technique de reconnaissance capable de déterminer les ports et services ouverts" [6]. Les approches adoptées par les activités de scanning reseau (NSA) ont été classées de différentes manières, en fonction de leurs mouvements dans l'espace IPv4 [11], leur nombre de sources et de cibles, leurs capacités furtives [12], etc. Puisque la définition ci-dessus des activités d'analyse de réseau ne mentionne rien au sujet de leur exécutant ni de leurs cibles, nous avons d'abord décrit qui sont leurs exécutants et leurs cibles, et nous avons distingué les interprètes entre défensifs, lorsqu'ils sont exécutés par des agents de sécurité, et offensifs, lorsqu'ils sont exécutés par des attaquants. Les frontières d'une activité d'analyse de réseau peut être définie comme [1, 11].*horizontale*, lorsque plusieurs adresses IP sont testées sur un seul port/service ; *verticale*, lorsqu'une seule adresse IP est testée sur plusieurs ports/services. Une activité d'analyse de réseau est appelée analyse de bloc lorsque sa limite est une combinaison des deux mentionnées ci-dessus. Les activités d'analyse du réseau peuvent également être étiquetées comme actives ou passives. Les activités d'analyse active du réseau identifient les services réseau de manière proactive : elles transmettent des sondes vers les hôtes ou les périphériques et surveillent leurs réponses. Les activités de balayage passif du réseau analysent les services réseau en observant le trafic généré par et entre les serveurs et les clients, lorsqu'il passe par un point d'observation. Les activités d'analyse de réseau peuvent également être coordonnées sur un ensemble de réseaux d'analyse, lorsque diverses sources fonctionnent de manière synchronisée [15, 18] pour obtenir furtivité, puissance de feu et plus de données. Nous avons ensuite détaillé quels protocoles peuvent être utilisés par les activités actives d'analyse de réseau, et nous nous sommes penchés sur les distinctions entre le trafic normal et le trafic de sondage, la nature bénigne ou maligne d'une activité d'analyse de réseau et sa corrélation avec des attaques. Nous avons analysé les approches de détection des techniques de balayage de réseau actuellement disponibles et défini un hôte de balayage comme une machine effectuant une activité de sondage. Plus précisément, il est défini comme étant un service qui cherche des ports ouverts en envoyant des *sondes*, c'est-à-dire et en analysant leur état final [16], ou comme "une adresse IP source qui balaye au moins un nombre fixe d'adresses IP uniques du réseau hébergé sur le même port et protocole à un taux de sondage Internet minimum estimé à 10 paquets par seconde" [5]. Nous avons ensuite fourni la définition des scanners réseau. Un Scanner réseau est un outil de balayage de masse, qui envoie des paquets vers diverses destinations et analyse les réponses correspondantes, ou leur absence. Son but est d'acquérir des connaissances sur les hôtes ou réseaux et/ou ports distants [84]. Plusieurs hôtes d'analyse peuvent être déployés par un scanneur réseau. Nous avons également fourni une analyse des scanners réseau les plus utilisés.

Notre travail s'est poursuivi avec une étude sur les techniques de détection des hôtes de scan et la coordination des activités de scan du réseau. La première peut être détectée par le biais d'approches algorithmiques, basées sur le seuil, informatique douce, basée sur les règles et visuelle. L'étapes importante est de bien choisir l'approche de détection qui convient le mieux aux besoins et qui atteint la performance requise. Une question ouverte à laquelle les experts en sécurité doivent faire face, et qui dépend strictement du réseau surveillé, est de savoir où placer les modules de détection de balayage de port et leurs meilleures configurations pour une utilisation dans un environnement multi-étape. Ces modèles doivent construire des profils du comportement réseau normal, ce qui est une tâche complexe pour diverses raisons. Tout d'abord, en raison de la taille énorme des données de trafic réseau qui doivent être traitées et analysées. Ensuite, parce que les modèles de trafic sont constamment mis à jour et changeants. Enfin, en raison de la présence de bruit dans les données. Les approches d'apprentissage machine ou les méthodes de soft computing qui sont disponibles à ce jour ne sont pas encore adéquates.

Pour modéliser les scanners réseau, nous avons choisi d'utiliser les modèles de mélanges fini et les modèles de Markov cachés, qui sont des modèles statistiques, c'est-à-dire qui caractérisent les propriétés statistiques de la sortie, et certains de leurs paramètres et conditions initiales fournissent un ensemble de sorties différentes. Les hypothèses sous-jacentes sont que le résultat peut être caractérisé comme un processus aléatoire paramétrique et que les paramètres du processus stochastique peuvent être estimés d'une manière précise et bien définie [69]. Étant donné que l'activité d'analyse du réseau est inconnue du point de vue du système analysé et qu'elle n'est observée qu'au moyen de journaux, aucune propriété spécifique des observations n'est disponible. Des modèles statistiques peuvent ainsi être utilisés pour caractériser les journaux collectés et déduire les schémas comportementaux des activités d'analyse du réseau. En l'état de notre connaissance, il n'existait pas de modèle de Network Scanners développé en utilisant uniquement des logs ou des informations collectées par le réseau ou le système ciblé. Notre travail vise à combler cette lacune, en développant des modèles pour deux Network Scanners à partir des journaux de paquets reçus par le réseau hébergé et ciblé. Les modèles de Markov cachés (HMM) sont bien adaptés pour atteindre cet objectif. Ils se caractérisent par le fait que la distribution générant une observation dépend de la valeur d'un processus ou d'une chaîne de Markov sous-jacent mais non observé [71]. Chaque valeur (appelée *state*) du processus de Markov caché est fournie avec une probabilité. En d'autres termes, la chaîne de Markov non observée du modèle de Markov caché peut supposer un nombre (fini) de valeurs, appelées *states*, décrites par une distribution de probabilités initiale qui est, en fait, la *mixture distribution* d'un *Finite Mixture Model.*

Nous sommes ensuite allés de l'avant, dans le but de construire des modèles statistiques de scanners de réseau (NS) du point de vue du réseau ciblé. Ce choix a été motivé par divers observations. Tout d'abord, l'état actuel de l'art est pauvre en modèles construits dans cette perspective, ce qui est motivé par le fait que les réseaux scannés ne connaissent pas a priori le Network Scanner déployé pour réaliser le NSA, qui cible le système. La première partie de notre travail s'est concentrée sur la construction et la validation de modèles stochastiques de l'intensité des scanners de réseau, qui pourraient être utilisés par les experts en sécurité et les chercheurs pour détecter les activités de balayage de réseau. nous avons choisi de construire des modèles stochastiques de deux scanners réseau à grande échelle : un open source, ZMap [20], qui est un scanner réseau horizontal rapide open source capable de scanner l'espace IPv4 entier sur un seul service en moins de 45 minutes [20] et un avec une approche en boîte noire, Shodan qui est un scanner bloc. pour les deux scanners, les seules données dont nous disposons sont les informations recueillies par le réseau hébergé. Plus précisément, leurs traces ont été recueillies à partir d'un Darknet [77] hébergé au High Security Lab (LHS) d'Inria Nancy - Grand Est. Un darknet est un télescope réseau permettant d'observer les activités dans l'espace IPv4 et/ou IPv6 et les cyberattaques à l'aide de la surveillance passive [77]. En pratique, il s'agit d'un ensemble d'adresses IP qui sont routables et allouées, mais non utilisées [8]. Comme les adresses IP d'un darknet ne sont pas utilisées, ce sont de nouveaux hôtes qui ne communiquent jamais avec d'autres périphériques. Il s'ensuit que tout trafic observé reçu par ces adresses IP est suspect et mérite une investigation [77]. Un darknet représente donc une vue partielle de tout l'espace IPv4, qui ne peut être distinguée du reste. La question qui s'est posée était de savoir comment distinguer le trafic de balayage, et plus précisément le trafic provenant respectivement de Shodan et de ZMap, de tout le reste. Ce problème a été affronté et résolu en connaissant les ensembles d'adresses IP qui exécutent chaque scanner considéré. Ainsi, nous avons filtré les paquets provenant d'adresses IP réservées à l'exécution de Shodan, qui sont déclarées publiquement, et de ZMap, qui est continuellement exécuté par l'Université du Michigan qui réserve une partie de son réseau pour cette tâche. Pour chaque sonde, l'horodatage, l'adresse IP source, l'adresse IP destination et le port cible sont disponibles.

Nous avons utilisé les modèles de mélange de Poisson finis et les modèles de Markov cachés correspondants pour modéliser les intensités des deux scanners réseau considérés, c'est-à-dire le nombre d'adresses IP ciblées dans un intervalle de temps donné et fixe. En d'autres termes, l'intensité des scanners réseau est définie comme la "vitesse" avec laquelle leurs paquets sont reçus par le darknet hébergé. Nous avons également cherché à savoir si les intensités varient par rapport au service sondé, afin d'évaluer s'il s'agit d'une caractéristique macroscopique des scanners réseau considérés qui suffit à les reconnaître à partir des journaux de paquets reçus par le port considéré. Pour atteindre ces objectifs, cinq ensembles de données ont été utilisés pour ZMap et un pour Shodan. Plus précisément, les ensembles de données concernaient les journaux de sondes générés par Zmap ciblant respectivement les ports 22, 23, 80, 443 et les ports multiples, et par Shodan. La distribution de probabilité que nous avons utilisée est celle de Poisson [13, 80], puisque la variable considérée est discrète et est un compteur de temps où un événement apparaît. Les modèles de Markov cachés construits sur les ensembles de données *ZMap-22* et *ZMap-23* nécessitent le même nombre d'états minimum, même si les modèles eux-mêmes ne peuvent être comparés en raison de la différence de leurs paramètres. Cela signifie que les intensités de chacun des deux ensembles de données sont regroupées en trois groupes et que les distributions et les probabilités initiales des groupes de chaque ensemble de données sont différentes. Une des motivations de cette fonctionnalité partagée peut être trouvée dans le service associé aux deux ports considérés, qui sont liés depuis SSH (port 22) est l'amélioration de Telnet (port 23). Les deux modèles de Markov cachés construits sur les ensembles de données *ZMap-All* et *Shodan*, respectivement, ont 6 états, mais des paramètres différents. Cela signifie que 6 clusters suffisent généralement pour grouper les compteurs de sondes provenant respectivement de Zmap et de Shodan, et reçues par notre Darknet. En d'autres termes, les scanners réseau sont exécutés avec une intensité qui suppose un nombre limité de valeurs et provient de l'un des six clusters. Tous les modèles de Markov cachés obtenus ont leurs probabilités de transition correspondantes qui partagent une caractéristique commune : leurs diagonales contiennent les éléments les plus importants de chaque ligne. Une fois entré dans un état, les probabilités d'en sortir sont faibles. Une fois que le Network Scanner est lancé, il est raisonnable de supposer que son intensité reste constante, à moins d'événements imprévus. Migrer d'un état à un autre signifie qu'une exécution du Scanner réseau considéré s'est terminée et qu'une autre a commencé, et que cette dernière exécution a une intensité différente de celle de la première. Rester dans le même état, par contre, pourrait signifier que deux compteurs consécutifs ont des valeurs similaires parce qu'ils sont pris dans la même exécution du Scanner de réseau considéré, ou que les deux exécutions successives des Scanner de réseau considérés ont la même intensité. Cela pourrait être motivé, par exemple, par le fait que les utilisateurs du Network Scanner ne modifient pas ses paramètres avant de le lancer à nouveau. Tous les modèles obtenus sont *discriminatifs*, puisqu'ils suffisent à reconnaître quel Network Scanner a testé le réseau et le port hébergé. L'intensité des scanners réseau est donc une propriété utile pour les analyses sur de longues périodes. Mais, d'un autre côté, il est moins pratique si l'on vise à identifier un scanner réseau sur de courtes périodes (pas plus longues qu'une seule exécution), avec seulement la disponibilité de moins de paquets.

La deuxième partie de notre travail visait à répondre aux questions suivantes : Est-il possible d'avoir un ensemble de modèles qui peuvent être utilisés pour reconnaître quel Network Scanner a généré les sondes collectées et qui ne sont pas strictement dépendants ou liés au réseau qui a reçu les sondes ? et "Ces modèles varient-ils en fonction du service testé par le scanner ? Pour répondre à ces questions, nous avons examiné deux caractéristiques des Network Scanners: les *déplacements spatiaux*, que nous avons définis comme étant différences d'adresses IP ciblées consécutivement par le scanner, et les *déplacements temporels*, définis comme différences d'horodatage de deux sondes consécutives. Nous les avons exploités séparément pour construire, pour chacun d'eux, leur modèle de mélange fini et leur modèle de Markov cachés. Ce dernier sera ensuite appliqué pour reconnaître les scanners réseau considérés. Nous construisons des modèles pour des exécutions uniques de chacun des scanners réseau considérés, et nous les utilisons pour reconnaître dans les ensembles de journaux de sondes fraîchement collectés ceuz qu'ils ont générés. Pour atteindre les objectifs, les logs collectés par un darknet ont été utilisés [81] et divisés en ensembles de données, selon le Network Scanner qui a généré les sondes. Ils couvrent de longs intervalles de temps, allant de plusieurs jours à plusieurs mois, contenant donc des journaux d'exécutions multiples du même Network Scanner (rappelons que ZMap peut n'avoir besoin que de 45 minutes pour sonder tout l'espace IPv4 sur un seul port, avec une seule machine [20]. Afin de modéliser le comportement d'une seule exécution du scanner réseau considéré en termes de temps et d'espace, les deux ensembles de données initiaux, un pour chaque scanner réseau, doivent être divisés en ensembles contenant les logs d'une seule exécution. Chaque ensemble de données est ainsi divisé en échantillons (chacun contenant les journaux d'une seule exécution) en considérant les laps de temps entre deux sondes consécutives reçues par le darknet : lorsqu'il s'agit d'une valeur aberrante, une exécution du scanner se termine et la suivante commence. Une fois tous ces échantillons disponibles, ils ont été utilisés pour apprendre les modèles de mélange fini gaussiens et leurs modèles de Markov cachés respectifs. Les premiers créent des clusters de logs et assignent à chacun d'eux une distribution gaussienne. Les HMM, qui fournissent des probabilités de transition entre les états, sont ensuite construites et appliquées à certains échantillons d'essai, afin d'évaluer quel Network Scanner a généré les logs dans l'échantillon considéré. La précision de notre approche est supérieure à 95%. Le premier problème est, après que tous les journaux collectés par ZMap et Shodan, respectivement, aient été stockés dans les ensembles de données correspondants, comment identifier le début et la fin d'une exécution de balayage du scanner réseau considéré. Les différences de deux adresses IP examinées consécutivement et de deux horodatages successifs (ces dernières différences sont, par la suite, appelées " time gaps ") sont calculées et stockées : les premières seront utilisées pour modéliser les comportements spatiaux, les secondes pour modéliser les comportements temporels des scanners. Successivement, nous avons identifié les ports communs en dessous de 1024 communs dans les deux ensembles de données, c'est-à-dire les services ciblés par ZMap et Shodan. Chacun des deux ensembles de données ZMap et Shodan a été divisé en (sous-)ensembles de données, un pour chacun des 17 ports communs inférieurs à 1024. Pour chaque (sous-)ensemble de données, nous avons filtré disjointement *learning* et *test datasets*. Chacun d'entre eux a un protocole initial qui est sélectionné au hasard. Les autres sont des journaux chronologiquement ordonnés qui suivent le premier qui vient d'être choisi. La longueur des *ensembles de données d'apprentissage* est fixée à 10% du sous-ensemble de données considéré, alors que la longueur des *ensembles de données de test* est de 5% de la longueur du sous-ensemble. A partir de chacun des sous-ensembles de données " Port Scanner-réseau ", nous avons donc filtré divers *learning* et *test datasets*, chacun avec un premier journal sélectionné au hasard et une longueur de 10% et 5%, respectivement, du sous-ensemble de données considéré. Dans chacun d'eux, les écarts temporels sont analysés. La présence d'une valeur aberrante indique qu'une exécution de l'analyseur considéré pourrait être interrompue et qu'une nouvelle analyse pourrait commencer. Ceci conduit à diviser les *learning* et *ensembles de données de test* en *learning* et *sous-ensembles de test.* Afin de s'assurer que les sous-ensembles *learning* et *test sub-sets* fraîchement filtrés ont été correctement divisés conformément aux exécutions complètes des scanners et non en raison de perturbations, l'index Jaccard mutuel de deux *learning* et *test sub-sets* chronologiquement consécutifs est calculé, respectivement. Ces derniers sont alors fusionnés en *learning* et *test samples* si et seulement si l'indice Jaccard mutuel des ensembles correspondants qui contiennent les adresses IP qu'ils contiennent est inférieur ou égal à 0,1. Cela signifie que le chevauchement entre les deux ensembles d'adresses IP sondées est faible, et il est raisonnable de supposer que les sous-ensembles *learning* et *test* qui les ont créés peuvent être considérés comme faisant partie de la même exécution unique du Network Scanner analysé, et peuvent donc être fusionnés ensemble. Cette fusion nous assure que chaque *learning* et *test sample* S_i obtenu contient les logs d'une seule exécution du Scanner réseau considéré. Les échantillons d'apprentissage sont utilisés pour apprendre les modèles à mélange fini et les modèles de Markov cachés, tandis que les échantillons d'essai servent à les valider. Nous avons obtenu un modèle de Markov caché pour chaque échantillon d'apprentissage. Nous devions réduire leur nombre en nous demandant s'il existe des modèles qui peuvent être utilisés pour bien s'adapter à plusieurs *échantillons d'apprentissage.* Chaque échantillon d'apprentissage a été ajusté à tous les HMM obtenus à partir d'échantillons d'apprentissage, et sa log-vraisemblance a été calculée. Toutes les probabilités logarithmiques obtenues sont stockées dans une matrice, dans laquelle les lignes sont liées à des échantillons d'apprentissage et les lignes à des modèles. Tous les HMM obtenus sont ensuite regroupés à l'aide de DBSCAN [83] qui, ne nécessitant que deux paramètres, ϵ et le nombre minimum de modèles dans chaque cluster, regroupe des paires de modèles si leur distance mutuelle est inférieure ou égale au ϵ choisi, et si un d'eux est entouré par un nombre suffisant de points supplémentaires. Tous les modèles d'un même groupe peuvent être considérés comme équivalents pour décrire les *échantillons d'apprentissage* qui les ont générés, mais un seul est requis : un seul modèle est sélectionné dans chaque groupe, et est le *modèle candidat* du groupe des HMMs. Le résultat final est donc une liste de *modèles candidats* M_i pour chaque combinaison "Network Scanner-port". Tous les *modèles candidats* ont ensuite été validés sur *échantillons test.* Pour chacun d'entre eux, le scanner réseau qui les a créés et le port ciblé sont connus. Chaque *échantillon test*, étiqueté comme "scanner-port-TS_i", a été ajusté à chaque *modèle candidat*, et la log-vraisemblance calculée. Toutes les valeurs des log-vraisemblances obtenues pour chaque *échantillon test* sont comparées, et le modèle qui a fourni la valeur la plus élevée est celui qui décrit le mieux le *échantillon test* donné. La partie initiale de son étiquette, "scanneur-port", est, à partir de ce qui suit, appelée *étiquette vraie* de l'échantillon d'essai considéré. Chaque modèle de Markov caché est étiqueté "scanner-port-M_j" : la première partie de son étiquette, "scanner-port" est stockée en tant que *étiquette prévue* du *échantillon test* considéré. A la fin, nous avons vérifié si l'*étiquette vraie* et l'*étiquette prévue* de chaque *échantillon test* sont égaux. L'exactitude de la classification est ensuite calculée en comptant combien de fois *étiquettes prédites* sont égales à *vrai* et en divisant la valeur obtenue par le nombre de tous les échantillons. Grâce à cette procédure, nous avons obtenu les résultats suivants. En ce qui concerne les mouvements spatiaux, tous les HMM candidats obtenus selon la procédure décrite à la section 6.2 ont un nombre d'états compris entre 1 et 3, dont 91,3% ont 2 états. À l'aide de la matrice de confusion, on calcule l'exactitude en additionnant les éléments des diagonales et en divisant la somme obtenue par le nombre de tous les échantillons (c.-à-d. la somme de tous les éléments des matrices). Si l'on se concentre uniquement sur la détection du scanner réseau qui est à l'origine des sondes dans les échantillons testés, il atteint 0,952. Plus en détail, seulement 21 des 439 échantillons testés ont été mal classés : 17 d'entre eux ont été générés par Shodan et étiquetés comme provenant de ZMap, tandis que 4 ont été assignés à Shodan tout en étant réellement générés par ZMap. Il est alors possible, avec une précision de 95%, d'indiquer quel scanner a généré les logs dans les *échantillons test*, et que les mouvements spatiaux des scanners considérés ne dépendent pas du port cible, et diffèrent seulement entre différents scanners. Regardons à nouveau la Figure 6.7 : malgré une précision globale de 95%, seulement 1% de tous les *échantillons test* réellement générés par ZMap ont été mal étiquetés. Ce pourcentage s'élève à 25,4% pour Shodan. Lorsque l'on considère les mouvements temporels, au lieu de cela, les modèles sélectionnés ont entre 1 et 7 états, avec 92,2% d'entre eux ayant jusqu'à 4 états. Chaque *échantillon test* a ensuite été ajusté pour *tous* les *modèles candidats* obtenus, et *chaque* log-vraisemblance correspondante calculée. Le modèle qui correspond à la log-vraisemblance la plus élevée est alors stocké pour être celui qui correspond le mieux à l'*échantillon test* considéré. Son étiquette, plus précisément sa partie initiale, "scanner-port", est stockée sous la forme *étiquette prédite* du *échantillon de test* donné. Pour calculer la précision de l'approche proposée lorsqu'elle est appliquée aux mouvements temporels des scanners, nous avons compté combien de fois le *étiquette prévue* de l'*échantillon à tester* est égal à son *étiquette vraie*. ZMap et Shodan ne changent pas leur comportement temporel lorsqu'ils ciblent un port particulier, mais que les mouvements temporels sont des caractéristiques internes des scanners réseau, suffisantes pour les empreintes digitales. L'approche proposée a une précision égale à 98%. Plus précisément, la précision par rapport à Shodan est alors égale à 100%. A propos de ZMap, au lieu de cela, seulement 10 échantillons sur 372 ont été mal étiquetés. Cela signifie que la précision de l'approche appliquée à l'empreinte digitale ZMap est de 97,3%, ce qui est suffisamment élevé pour affirmer la qualité de la méthodologie proposée.

Certaines questions de recherche restent à résoudre. La première consiste à rechercher les raisons pour lesquelles Zmap varie en intensité par rapport aux ports sondés. Une réponse naïve et rapide à cette question est que certains services (c'est-à-dire les ports) sont plus intéressants que d'autres. La question de emph real est: "Pourquoi certains services sont-ils plus intéressants que d'autres?". Comme détaillé au chapitre ref chap: NSA, les intérêts de certains services ne disparaissent jamais, alors que pour d'autres, ils s'intègrent et diminuent. Le second cas survient généralement lorsqu'une vulnérabilité du service est trouvée. Ainsi, le travail intéressant que nous espérons accomplir à l'avenir est de motiver l'intérêt suscité par une augmentation de l'intensité des sondes dirigées vers un port donné.

Une autre question de recherche consiste à évaluer si Shodan peut avoir un comportement similaire à celui de Shodan: peut-il sonder avec des services vulnérables d'intensité plus élevée?

Ne considérons pas les modèles de mélange finis créés sur les mouvements temporels, groupe qui se connecte en un nombre de groupes compris entre 1 et 7. Bien que la signification des groupes en cas de mouvements spatiaux soit claire (l'un pour les mouvements en avant, l'autre pour les mouvements en arrière), une caractérisation des clusters fournie par Finite Mixture Models est souhaitée.

Le travail souhaité serait de créer des modèles de Markov cachés capables de décrire les deux mouvements des scanners de réseau, afin d'accroître la précision du processus de reconnaissance.