POLITECNICO DI TORINO

Dipartimento di Elettronica e delle Telecomunicazioni
Corso di Laurea Magistrale in ”ICT for Smart Societies”

Tesi di Laurea Magistrale

HoneyPort - a scalable meta-honeypot system for security applications

Supervisors:
Prof. Marco Mellia
Dr. Idilio Drago

Candidate:
Eros Filippi

July 2019

Contents

List of Figures

Abstract

The internet has become almost essential in everyday life over the past fifteen years. Some researches state that there will be around 30 billion active networked devices around the world by 2020. The main drawback of this trend is that the continuous sharing of private information among these networked devices could lead to critical privacy and security issues that can not be neglected.
Moreover, a lot of cyber-attacks have exploited vulnerabilities in tens of thousands of IoT (Internet of Things) devices in several ways. Among them, there are the DDOS attacks that consist of sending a crippling amount of service requests to unaware websites. One of the most effective methods to gather information regarding hacking attempts is the implementation of Honeypots within local area networks.
Honeypots are information technology systems which entice attackers by providing low vulnerabilities hosts, used as a decoy, in order to record their malicious activities. In most of the literature on honeypots, the goal is to simulate a vertical system that exposes a service protocol over a communication endpoint called port. One example is the open-source project Cowrie, maintained by Michel Oosterhof, that mimics the SSH protocol service listening on port 22.
Since nowadays the typical behavior of a hacker is to attack the networked devices on maintenance backdoors, it has become necessary to develop an honeypot system capable of handling horizontal requests on all possible TCP/UDP ports.
Two fundamental shortcomings affect these systems:

- *Visibility problem*: It is difficult to communicate with the attacker horizontally on all TCP / UDP ports (port coverage).
- *Flexibility problem*: The system needs to instantiate the fitting Honeypot when the attacker requests a service.

To solve these problems some research groups have employed the strategy of placing different honeypots on a single machine. These honeypots can only listen to either predetermined or random ports.
This solution is a good starting point, but it does not ensure complete coverage on all

possible endpoints. Not to mention, they present a lack of adaptability in different environments.
My research contribution was to improve the honeypot system by making it more flexible and scalable while keeping the ability to simulate a specific service.
I developed a meta-honeypot aggregator called HoneyPort which is able to classify the protocol requests from all the 65535 TCP/UDP ports (Fig.1). Through proxy functions, it redirects the traffic to the most suitable honeypot.
The requests are firstly aggregated in a single endpoint and analyzed through a clas-

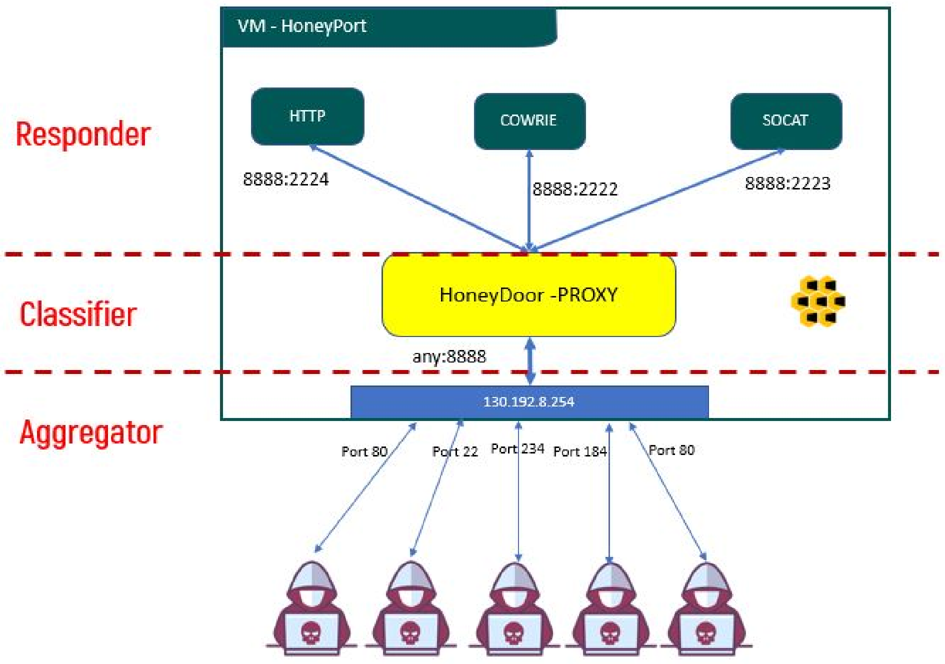


Figure 1: HoneyPort

sifier to identify the service demanded by the attacker. Then a connection is settled toward the most suitable honeypot, optimizing the number of occupied sockets.
The proxy classifier inside the HoneyPort manages to reach great flexibility because it adds a level of isolation between honeypot and endpoint, solving in this way the visibility problem.
The microservices approach used during the development of HoneyPort allows the administrator to instantiate or deactivate honeypots based on his needs.

The system was validated through the comparison of one month of data packets collected after the deployment of three different kinds of environments. The three environments were:

- Darknet: A single IP address that is advertised but does not host any client or server.
- Honeypot: Implementation of the Cowrie honeypot together with a Web server on a host connected through the internet.
- HoneyPort: Implementation of the HoneyPort system on a host connected through the internet.

The validation was performed by assessing the following two questions:

- What is the most contacted system?
- Could the variety of requested services be affected by the HoneyPort?

The Honeypot registered an increase in the number of service requests by 97% compared to the Darknet environment, while HoneyPort was able to reach an astonishing 157% .
So, the HoneyPort system has the potential of gathering a more conspicuous amount of data packets and information about the cyber-attacks. It is also capable of dealing with a greater variety of service requests with respect to the honeypot system.
Adding proxy functionalities to a honeypot represents a significant step forward in the field of security research because it makes the system flexible to a great variety of service attacks. Also the port it doesn't need to be pre-configured during installation .
As highlighted from the results, HoneyPort is a promising project that, through further improvement, could represent a new frontier in honeypots development.
The HoneyPort system is now available for the computer science community as an open-source project on *Github*.

Chapter 1

Introduction and problem description

1.1 Overview

Over the past fifteen years, the internet has become almost essential for everyday life. It allows us to work, organize, contact people through emails, socialize with applications such as Instagram and Facebook and many other things.
Based on CISCO's global forecast highlights for 2020 [1], the average speed of the internet will reach 50 Mbps, which will create a 3-fold increase in the traffic compared to 2015.
The amount of gigabyte generated every two minutes by 26.3 billion networked devices will be equivalent to the memory space of all the movies ever made until now. The 71% of the devices will be part of IoT (Internet of Things).
There are still no common agreements on the meaning of IoT, so many organizations and researchers tried to clarify this definition.
Some proposed definitions are:
"*A world where physical objects are seamlessly integrated into the information network, and where the physical objects can become active participants in business process*"[2] and from Cisco "*IoT is simply the point in time when more "things or objects" were connected to the Internet than people.*"[3]
For our purpose, by IoT devices we mean physical or virtual object or virtual objects that can communicate through the network layer to human users or other objects so that the IoT applications are endless.
Almost all IoT devices must collect user information for analysis or data forecasting.

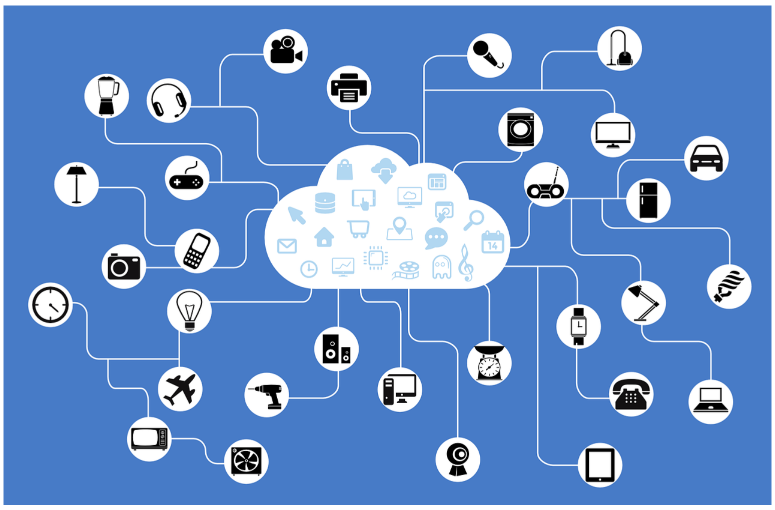


Figure 1.1: Internet of Things

The majority of these devices are wearables that collect data of various kinds: from simple heartbeat, temperature, humidity in the air up to personal information such as GPS position, determine daily routines, etc.
Other IoT sensible applications are in the construction industry, such as monitoring system for the health status of buildings or building actuators as light bulbs, thermostats, appliances.

1.2 Motivation

These ”smart things” could be affected by vulnerabilities produced by careless program design which, if hacked, may raise a serious problem to people.

Case study [4] shows that many smart devices sold to large retailers have many security holes that create opportunities for malwares or backdoors installation. A backdoor can then spread the infection to other smart appliances within the local area network to steal personal information or even physically harming the users. Botnets have become one of the major attacks on the internet today not only for IoT devices. A botnet is a network of devices that are compromised and controlled by an attacker. Each compromised computer is installed with a malicious program called a bot. The infected devices actively communicate with other bots in the botnet or with several bot controllers to receive commands from the botnet owner (called botmaster). Botmasters maintain complete control of their botnets and can conduct Distributed Denial-of-Service (DDoS) attack, e-mail spamming, keylogging, abusing online advertisements, spreading new malware, etc. [5].

So, there is an important Security Risk for what concerns IoT and this problem will only grow in the next future.

1.2.1 Monitoring system

The security policy used to protect a network requires to be regularly updated in order to be safe against new attack patterns and malwares.
More Knowledge on different emerging hardware leads to the possibility to identify threats more easily. It is of crucial importance to collect accurate, concise, and high-quality information about malicious activities[6].
There are many methodologies to extract information in the cybersecurity fields.
One of the effective ways to watch Internet activity is to employ passive monitoring using sensors or traps such as darknet [7].
Another solution widely used for this purpose is the deployment of active or interactive monitoring based on honeypots.

1.2.2 Darknet and common attacks

Darknet data is defined as traffic targeting advertised, but unused, IP addresses. Since these network addresses are unused, they represent new hosts that have never been communicating with other devices consequently any observed traffic destined to these non-interactive hosts raises suspicion and hence necessitates investigation.
These darknet-based monitoring systems are designed to attract or trap attackers for intelligence gathering[7].
The related research work did by the Politecnico di Torino was base on data collected during 3 weeks from different Darknets implemented in Brazil, Netherland and Italy.[8]
The majority of the traffic was represented by TCP, in particular, Scan traffic generated by a small fraction of the senders' IP address. this means that they are likely non-

occasional SYN scan.
There was a bit of Backscatter traffic that is a side effect of spoofed DoS/DDoS attacks(Fig. 1.2).
In this kind of DoS attack, the attacker spoofs the source address of the network pack-

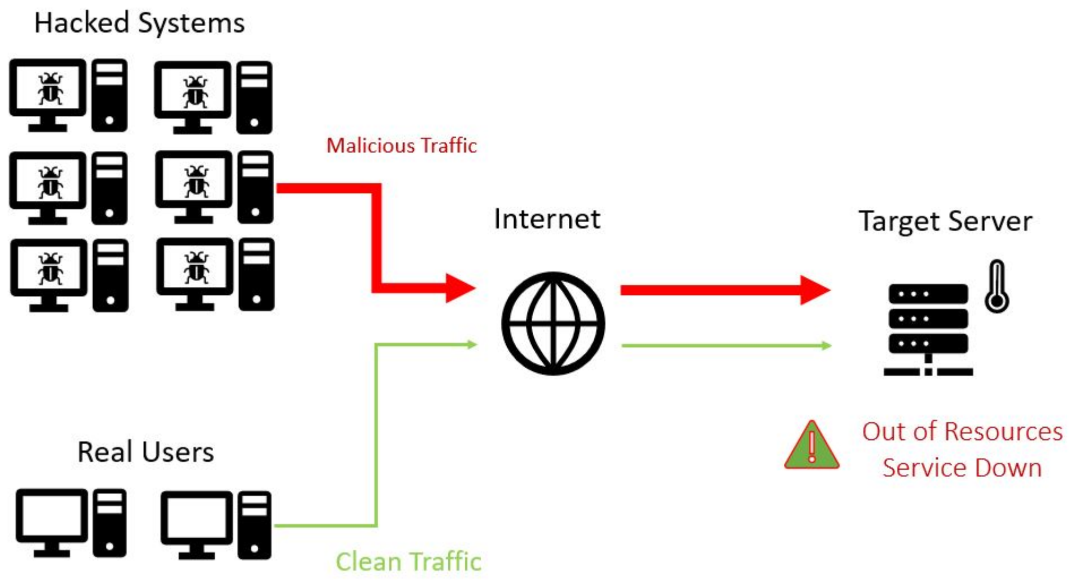


Figure 1.2: DDoS attack

ets sent to the victim. In general, the victim responds to these spoofed packets as if they were normal because the victim machine cannot distinguish between the spoofed packets and legitimate packets [9].
The traffic generated by these responses is very challenging to analyze in a Darknet environment. More than the 94% sources are common to all three darknets with a prevalence from Russia America and Bulgaria, Great Britain, USA, Ukraine, and China.
The most popular scanned ports are associated with services known to be targets of attacks, e.g., telnet (port 23), ssh (port 22) and some other linked to attacks targeting IoT devices.
The results of the papers provided further evidence that cyber attacks are still very present and constantly evolving. For this reason, further analyses are needed to implement the correct defense policies.

1.3 Goal of the thesis

This thesis aimed to create a flexible system for monitoring IoT computer attacks, able to incorporate and extend the functionality of the latest Honeypot systems.
We developed an active trap-based monitoring system in the darknet implemented at the Politecnico di Torino to make the monitored IP space more attractive for hackers.
Our system must be able to interact with different service requests on a high range of TCP/IP ports in two ways:

- Horizontally: Solve the problem of the low visibility of the Honeypots project. This means that the system has to listen to each TCP/UDP ports and redirect the traffic to the most suitable honeypot.
- Vertically: Our system collects several Honeypots and redirects all the requests to the most suitable trap based on the protocol instead of the demanded server ports. In order to make the system more versatile and efficient, low-interaction and medium-integration honeypots are deployed in a virtual environment.

Additionally, it has to guarantee a high level of security for the other hosts in the network and good privacy policies. It has to be reliable and portable to different server machines with the possibility of expansion in term of honeypots and protocol classification.

1.4 Organization of the thesis

Here we provide a Short Summary of the Thesis organization.

1. *Chapter 1* offers an introduction to the problem that we are posing and our goal: develop a scalable meta-honeypot system that can redirect service requests to the most suitable honeypots.
   It provides a general idea about previous research done on such topics and the solution we are providing.
2. *Chapter 2* illustrates the main features of the honeypot projects in the literature and explains the functioning of the most known and complete ones.
   One of the goals of the chapter is also to provide some basic knowledge of the tools adopted in the thesis work.

3. *Chapter 3* presents the architecture and the system developed to solve the problem. It narrows down in some key points of the algorithm implemented to the HoneyPort.

4. *Chapter 4* reports the various Results and insights got by running the meta-honeypot system. It offers a comparison of the data communication recorded between Darknet, Honeypot and HoneyPort environments.

5. *Chapter 5* points out the most important conclusions and offers suggestions for future work.

Chapter 2

State of the art

2.1 Honeypot state of the art

Definition: " A honeypot consists in an environment where vulnerabilities have been deliberately introduced in order to observe attacks and intrusions"[10].
So, it consists of a machine that appears to be part of the network but which is isolated and protected, and which seems to contain information or functionalities that can be attractive for hackers (Fig. 2.1).
One of the most important information that the honeypots can obtain is the behavior of the hacker and the samples downloaded by the botnets to infect other devices. These data are of crucial importance in the fields of 0-day exploit that refers to a newly discovered software vulnerabilities in which an official patch or update to fix the issue has not been released.
Moreover, it is difficult to exhaustively examine the logs for anomalous behaviors because most of the production machines have a large amount of network traffic, consisting of both unwanted connections.
Since honeypots have no production value, any interaction within the machine is always suspicious or no legitimate so that the false positive is very rare respect to other security solutions.
This allows the system administrator to target and analyze system without having to determine which traffic is suspicious and which is legitimate, analyzing logs from production systems[11].
The main disadvantages of honeypots are :

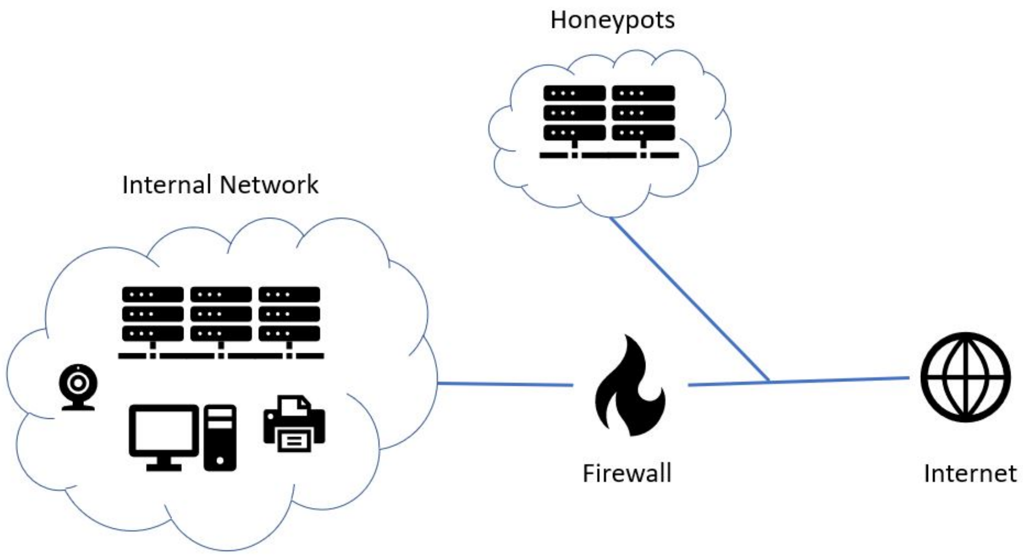


Figure 2.1: Honeypots network infostructure

- They could be fingerprinted form expert hacker.
- Install honeypots in a real network could be dangerous for the other devices because it may be used to reach other systems in the same network and infect them.
- The hacking activity must be destinated to interact directly on the IP address and port where the honeypot running.

There are three categories of honeypot based on the degree of exactness of the system resources information provided to the attackers.

2.1.1 Low-interact Honeypot

Low-interact honeypot provides limited emulated services with the only intent of detecting the source of the unauthorized activity. Since they are simply the fingerprints of OS functions, it is almost impossible to compromise it. They are more used in active client honeypot (example Honeyware [12]) to identify and determine some web services security holes.

2.1.2 Medium-interact Honeypot

Medium-interaction honeypot provides much more interaction to the adversaries than the low-interaction one does.
The network protocol is implemented and the simulation algorithm is based on emulating logical application responses where a pre-set of fake replies are employed according to the requests of the attacker.
For instance, a medium-interaction system could partially implement the HTTP or SSH protocol to emulate a well-known vendor's implementation.

2.1.3 High-interact Honeypot

High-interaction honeypot is the far most complex solution because it allows the attacker to interact with a real operating system and applications. The implementation typically involves the deployment of real operating systems on physical or virtual machines.
This solution can gather much more information on the behavior of the attackers.
It is very hard to distinguish the honeypot from the other systems but, on the contrary, this solution can expose the entire machine to many security risks.
Experienced hackers can easily install rootkits to detect and disable monitoring mechanisms. In this case, the control of the machine would be totally lost and the system could be used for other hostile activities.
The maintenance is very complex because the honeypot requires to be cleaned and reinstalled at each infection. Moreover, isolate the honeypot host from the local area network can be considered good practice. A common way to provide remote shell access on an operating system is using the Secure Shell (SSH) network protocol.
The Secure Shell (SSH) Protocol is a protocol for secure remote login and other secure network services over an insecure network.
The User Authentication Protocol authenticates the client to the server in several ways:

2.1.4 Cowrie

- Automatically generated public-private key pair is the most secure method but, it needs a prior communication of host keys or certification.
- The Username-Password Authentication. The protocol provides the option that the server name-host key association is not checked when connecting to the host for the first time so, this method could be affected by man-in-the-middle attacks or brute force attack through dictionaries.

[13] Commercial IoT devices often have an SSH service running by default with a weak password in order to give backdoor access to the consumers during the setting phase.
In a dictionary attack, each password in the dictionary is hashed and subsequently checked against the hash in question. If the hashes match, then the password has been found. Otherwise, it will continue to check and hash each password in the list until it has reached the end[14].

Cowrie is a medium interaction honeypot designed to emulate the entire shell interaction performed by the attacker over SSH and Telnet protocol. It provides a fake file system, simulating a Debian Linux server with the ability to add and remove files.
The honeypot allows the attacking entity (hacker or botnet) to attempt a login to the system, believing it is entering into a legitimate SSH session with the real server. Upon successful guessing of the password, the attacker is then moved into a fake system in which the attacker can navigate around, interact with and use some basic tools (such as cat, wget, etc.)

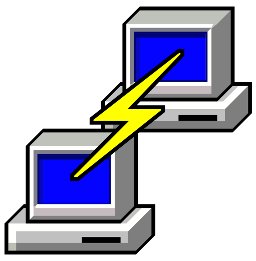


Figure 2.2: SSH

In this fake system, all interactions are monitored and recorded with session logs stored in a UML compatible format or JSON file providing also the timing of the prompt.
Moreover, the system also allows the use of wget and other commands commonly used to fetch or download files and manipulate it on the infected host. It also saves the downloaded samples obtained by wget for later investigation.
Cowrie is an open source project on GitHub written in python, continuously updated

with new features. It is well known to the community but, this involves the fact that it can be easily fingerprinted by expert hackers.
Another problem of Cowrie regards its"low visibility", because the service is listening on specific TCP ports. Therefore SSH requests to different ports are not taken into account.

The fundamental shortcomings of the Honeypot systems that we are trying to recover :

- *Visibility problem*: The ability to communicate with the attacker horizontally on all TCP / UDP ports.
- *Flexibility problem*: The ability to instantiate in real-time the required Honeypot system when the attacker requests a service.

2.1.5 Useful Tools

The following subsection will present some tools employed during the thesis work to test the system and data analysis.

Nmap

The primary phases to hack an information system include reconnaissance and scanning (also called network enumeration) where the Hacker tries to gathers as much information as possible about the target. In this steps, the attacker could determine the accessible hosts on the network, which operating system was installed on those machines (OS fingerprinting), the type of services that are exposed on some ports (Port scanning), and many others information just by analyzing the response to some ICMP or TCP ping.
Nmap is afree and open-source network that grew into the world's most popular network security scanner[15] that can perform a scan on all the TCP and UDP ports on every possible IP address in the network range.

HDFS

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware.
It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant.
"HDFS is highly fault-tolerant and provides high throughput access to application data. HDFS is tuned to support large files. It should provide high aggregate data bandwidth and scale to hundreds of nodes in a single cluster."[16]

Apache Spark

Apache Spark is a general-purpose cluster computing engine which is very fast and reliable. This system provides Application programming interfaces in various programming languages such as Java, Python, Scala.

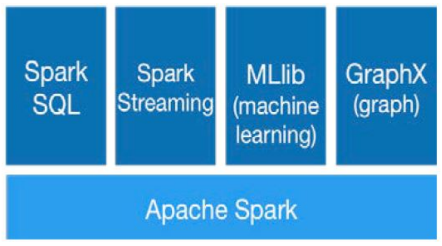


Figure 2.3: Spark Modules - from Big Data Analysis:Apache Spark Perspective (ISSN 0975-4172)

Spark is a cluster computing system from Apache. It supports in-memory computing, that enables it to query data much faster compared to disk-based engines such as Hadoop or any traditional data processing engine.
This system also provides a large number of impressive high-level tools (Fig. 2.3) such as machine learning tool M Lib, structured data processing, Spark SQL, graph processing took Graph X, stream processing engine called Spark Streaming, and Shark for fast interactive question device[17]. One of its major pros is the fact that it is possible to parallelize the instances of a program in several batches spread throughout the cluster.
If latency is the crucial point of the application or the machine has very limited storage memory available, other frameworks, like MapReduce form Google, may be more efficient than Spark.

nDpi

nDPI is an open-source library for protocol classification using both packet header and payload.
In the early days of the Internet, network traffic protocols were identified by a protocol and port. HTTP requests used TCP port 80 or SSH daemon listened to the TCP port 22.
Over the last years, this well-known protocol/port association expired due to the advent of port-independent, peer-to-peer, and encrypted protocols. Therefore, the analysis of a packets data payload becomes more and more challenging.
Deep Traffic Inspection (DPI) libraries, like nDpi, replace the first generation of port-based tools using more sophisticate classification algorithm.
nDPI usually detects protocol information by a traffic dissector written in C and the performances in terms of protocol recognition accuracy are similar to some commercial and open-source toolkits [18].
The nDpi output is detailed as follows:

- *Source_ip* : Internet Protocol source address of the attack.
- *Source_port*: Source transport port number.
- *Dest_ip*: Internet Protocol address of the Host server.
- *Dest_port*: Host server transport port number.
- *l4_proto*: Transmission Control Protocol type of connection.
- *detect_proto* : application-layer detected protocols.
- *packets_bytes*: the amount of data shared during the communication flow.

Listening figure 2.1 shows an example of nDpi output.

```
[source_ip, source_port, dest_ip, dest_port, l4_proto, detect_proto,
    packets bytes]
#examples
#[u'', u'\t1\tTCP 130.192.8.254:47972 <-> 91.189.88.161:80 [proto:
   7.169/HTTP.UbuntuONE][cat: Cloud/13][215 pkts/17065 bytes <-> 485
    pkts/3149435 bytes][Host: archive.ubuntu.com]'
```

```
#[u'\t426\tTCP 45.227.254.22:51831 -> 130.192.8.251:19938 [proto:
   0/Unknown][1 pkts/60 bytes -> 0 pkts/0 bytes]', u'\t427\tTCP
   45.227.254.22:51831 -> 130.192.8.251:21042 [proto: 0/Unknown][1
   pkts/60 bytes -> 0 pkts/0 bytes]'
#["u'', u'\t1\tTCP 130.192.8.254:47972 <-> 91.189.88.161:80 ", '
   proto: 7.169/HTTP.UbuntuONE]', 'cat: Cloud/13]'
```

Listing 2.1: nDpi Output

2.1.6 Related works

The following section will provide an explanation about some of the honeypot-related projects that simulate listening services on multiple ports.

Dionaea

The Dionaea low interaction honeypot is the successor of Nepenthes introduced in 2009.[6] It is designed to emulate vulnerabilities that are exploited by botnets during an attack and to detect and capture the related worms.
Dionaea is based upon a flexible modular architecture which can be easily extended by adding new vulnerabilities.
The core of the honeypot handles the network interface and coordinates the actions of the other modules. The actual work is performed by different modules, one for each vulnerability you required to simulate.

It can reproduce a full Windows remote procedure call (RPC) stack used by various Windows network services. Additionally, the system can simulate other protocols, such as MongoDB, MSSQL, MySQL, SIP, SMB, FTP, HTTP, and HTTPS;
Dionaea can also send all the activity of the system to online malware analysis services (like VirusTotal) and provide some visualization tools.
Even if it can simulate many services, each of them can listen only to one specific port. For this reason, it cannot be exploited to solve the multiple ports requests problem.

Figure 2.4: Dionaea muscipula

The Dianoea name comes from acarnivorous plant native to subtropical wetlands on the East Coast of the United States that It catches insects and arachnids with a trapping structure formed by the terminal portion of each of the plant's leaves (Fig. 2.4).

BitNinja

Figure 2.5: Bitninja Logo

BitNinja is a general purpose security-as-a-service server defense tool powered by a social defense system together with many active defense modules (Fig. 2.5).
This project includes some honeypot modules to prevent, detect and gather information about IT security issues.
The PortHoneypot module can set up to 100 honeypots on the server on random ports chosen among the 1.000 most popular ports. This module could detect if someone is performing a deep port scan on the analyzed server. The module captures the entire traffic on these honeypots and replies to the requests with some implemented chat scripts for different protocols (FTP, telnet, SMTP, IMAP,POP3).
The module does not bind on actual ports, but it binds on a port among 60.000 and it employs Iptables rules to forward the traffic from the actual ports.
If a daemon starts listening to a honeypot port, the module will automatically stop using that port as a honeypot. When the module starts, it also lists all the open sockets in listening mode and wont start honeypot on active ports. In this way the module will automatically avoid any collision with real services.

The main drawbacks of this system are two:
It is a proprietary code hold by the the server security companyBitNinja Kft therefore, intellectual property laws do not allow the disclosure of the source code for research propose.
In second hand, this system uses honeypot traps that are very easy to identify because they are simple chat scripts based on the requested protocol. Hackers can quickly discover the trap by analyzing the response received from the server.

Honeypot Architecture Project

The project entitled ”Scalable Honeypot Architecture for Identifying Malicious Network Activities” was presented to the International Conference on Emerging Information Technology and Engineering Solutions in 2015 made by the Department of Computer Science and Information Systems BITS, Pilani - Hyderabad Campus, Hyderabad, India.
The authors have deployed several honeypots in a virtualized environment to gather traces of malicious activities.
A distributed architecture has been deployed using only low-interaction honeypots that capture attack packets specific to a set of ports.
They spend a lot of attention making the system resilient to failures. If any of the honeypot VM is brought down by an attacker, the system still works with the help of other honeypots.
Even if a VM is entirely compromised, the hackers would not have sufficient access privileges to break into other VMs in the network because the hosts are mutually isolated among each other.
The server machine runs Ubuntu 12.02 upon which a hypervisor exists. The hypervisor runs five virtual machines, each one of them running a different honeypot. The traffic toward the virtual machines is controlled using a firewall and a NAT making easier to build a DMZ (demilitarized zone) with private IP addresses used for the machines.
The installed honeypots were: *Dionaea* (listening on ports 21, 69, 445, 1433, 3306, 5060 and 5061), *Glastopf* (listening on ports 80 and 8080), Kippo (listening on port 22), *HoneyD* (listening on ports 23, 25, 53, 110), *J-Honey* (listening on ports 135, 137, 138, 139, 1080, 1243, 12345, 12348, 27374 and 31337). The advantage of this system is that it is able to aggregate the best honeypots available on the internet (in term of patches and updates) and it makes difficult for the hackers to spot the trap because of the complex architecture.
The main disadvantage is that service requests not on well-known ports are lost and not tracked.

Chapter 3

Architecture

3.1 Architecture

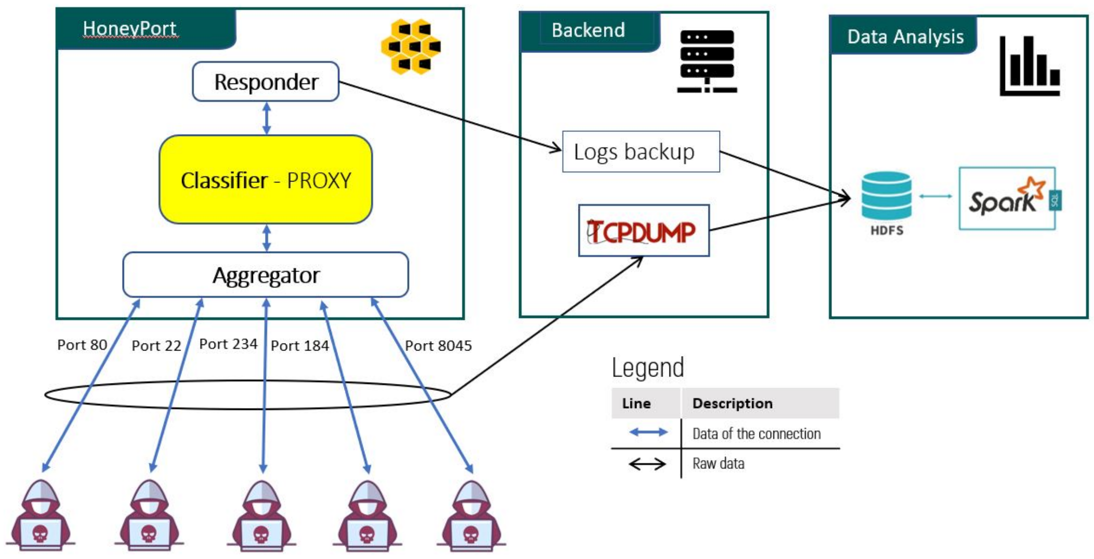


Figure 3.1: General architecture.

The architecture of the system developed my thesis can be summarized through the following points: (Fig. 3.1):

- **HoneyPort**: it aggregates, classifies and replies to all the incoming requests.

- **Host server**: it keeps track of connections with the attackers and and saves all the raw data exchanged during the communication.
- **Data Analisys**: it analyzes the traffic data.

Figure 3.2 shows a brief overview of the algorithm that is behind the honeypot system. When communication requests arrive at the system, they are aggregated into a single

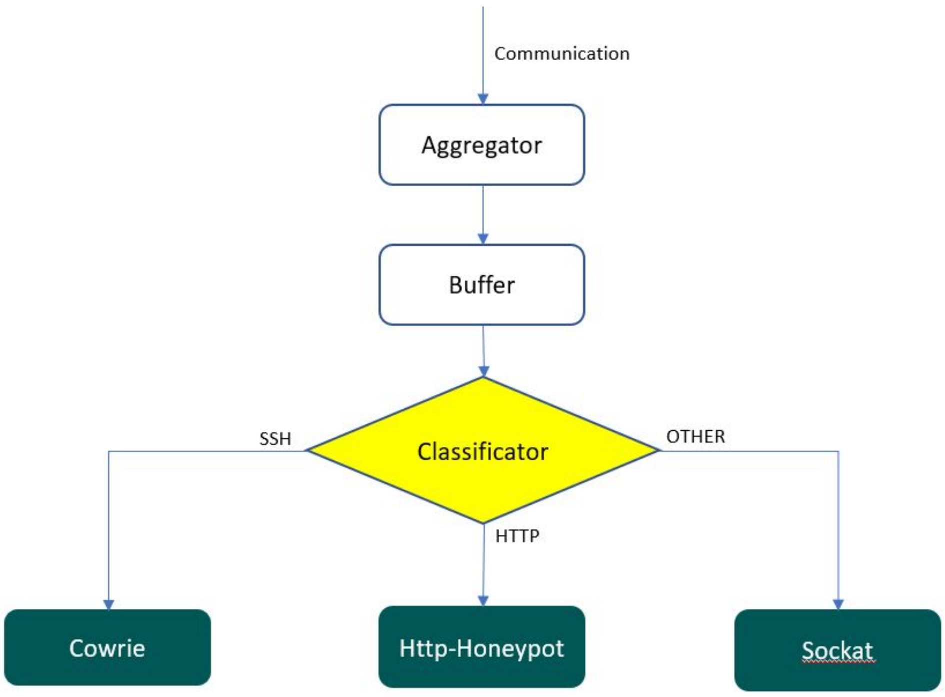


Figure 3.2: HoneyPort Diagram.

node in order to be easily examined by the service.
Each data flow is buffered to facilitate the task of identifying the required type of protocol performed by the classifier.
The classifier compares the incoming packets with typical protocol patterns to select the most suitable responder between Cowrie (SSH data flows) or HTTP-HoneyPot (HTTP data flows).
If any protocol between the supported patterns is found as matching, the Multi-Purpose responder Socat is able to establish a low-level connection.

This module must be installed in an isolated development environment because honeypots can be controlled by expert hackers for malicious actions against local network machines.

The Host server module acts as a packet sniffer, in order to capture all the data stream directed to the HoneyPort system and logs them in the internal storage.
This module behaves the backup function also for all the honeypots installed inside the HoneyPort. The Host server requires a secure environment against attacks and, in the meanwhile, it requires also a large amount of storage since it contains all the raw data gathered from the system.

The Data Analysis module aims to process the data recorded to the Host server.
The most suitable deployment hardware consist of a cluster with large calculation capacity.

3.2 HoneyPort

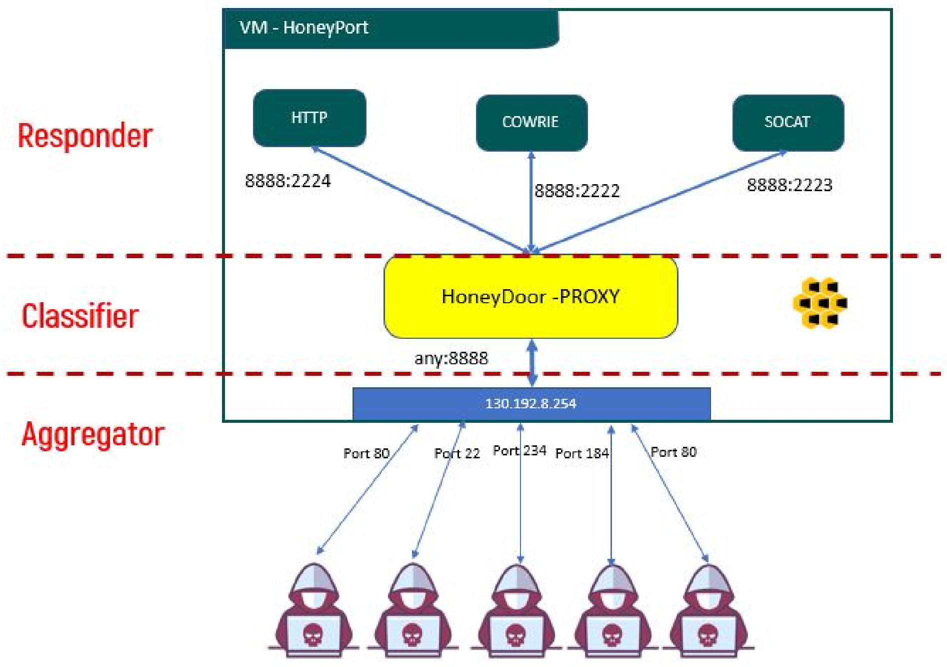


Figure 3.3: Architecture

The HoneyPort system is based on three main stages:

- **Aggregator**:It aggregates all incoming traffic through one specific socket port.
- **Classifier -Proxy**: It classifies the flows based on the extracted protocol and it redirects the packets to the appropriate Honeypot.
- **Responder**: It represents the set of installed honeypots.

3.3 Aggregator

The Aggregator is the software component that conveys all the requests on different ports in a single one. Its main purpose is to analyze the traffic through a single port by a listening process and to expose all the UDP/TCP endpoint occupied by a listening

service.
This microservice acts directly on the rules chain of the Linux kernel through the user-space utility called IPTABLES. The kernel consults the appropriate chain to find a match between the rules and the incoming packet. Once that the first match is found, the action specified in the rule's target is taken.

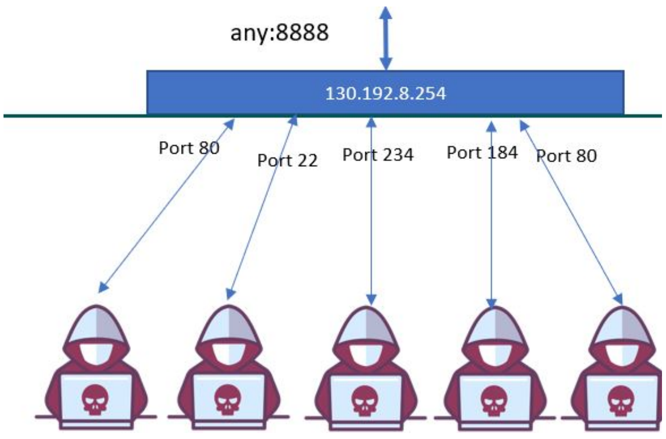


Figure 3.4: Aggregator

The heart of this microservice is composed of four rules:
The first two rules (ref Listing: 3.1) disable outgoing ICMP and RST requests from the server.

```
iptables -A OUTPUT -p tcp -m tcp --tcp-flags RST RST -j DROP
iptables -A OUTPUT -p icmp -m icmp --icmp-type 3 -j DROP
```

Listing 3.1: Iptables rules part 1

When an unexpected TCP packet arrives at a host, that machine usually responds by sending a reset packet back on the same connection. A reset packet is simply one with no payload and with the RST bit set in the TCP header flags.
Likewise, when an Internet Control Message Protocol (ICMP) ping request arrives at a host, immediately an ICMP message will be sent back.
During a port scanning, many open-source network scanner (like Nmap ref:2) send multiple SYN packets to analyze the replies. A RST packet is indicative of the absence of listener and ICMP reply can give further information.

In the second part of the firewall configuration (ref Listing: 3.2), we modify the prerouting chain, that is used to make any routing related decisions before sending any packets. NAT automatically tracks the active connections so it can forward return packets back to the correct host.

```
iptables -t nat -A PREROUTING -p tcp -m set --match-set
    exported_ports dst -j REDIRECT --to-ports 8888
iptables -t nat -I OUTPUT -p tcp -o lo -m set --match-set
    exported_ports dst -j REDIRECT --to-ports 8888
```

Listing 3.2: Iptables rules part 2

All the packets that are sent or received through the ports specified in the Ipset list called exported_ports are mapped to the internal server port 8888.
The Ipset is an extension of Iptables that we used to create firewall rules that match the entire ”sets” of addresses at once. The IP sets are stored in indexed data structures, making lookups very efficient, even when we have to deal with large sets.
I write two python programs to manage the Ipset list:

- **Ipset_generation.py**: it adds all the port from 0 to 65534 into the ipset *exported_ports* (the last port 65535 is used as back-door for server configuration).
- **Ipset_monitor.py**: it checks the services listening on the server's ports. The program removes the ports on the ipset *exported_ports* that are already occupied by another service and it adds the free ports. This process must run in the background for the entire test time.

These two programs allow us to analyze all the packages by a service listening on a specific port (Fig. 3.4, specifically port number 8888).

3.4 Classifier - HoneyDoor Proxy

The honeypot Proxy is the hearth of my system (Fig. 3.5).
The goal of my system was to create a structure able to recognize on the fly the service requested by the attacker to address it internally to the most suitable honeypots.
So the code must comply with some specific requirements:

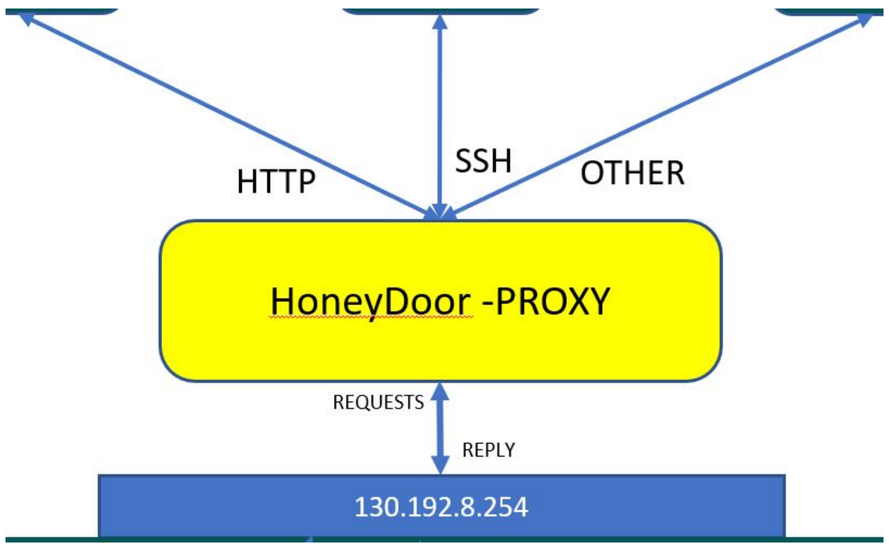


Figure 3.5: Honeyport-Proxy

- **Interoperability**: the system must interact with external requests that come through the port 8888 and the internal honeypots. After the classification of the protocol, it has to create a hidden communication channel between the hacker and the selected honeypot.

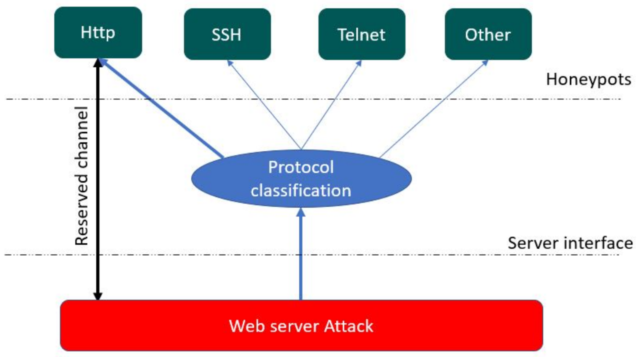


Figure 3.6: interoperability

- **Performance**: The system must be able to handle the requests as quickly as possible avoiding to reveal the trap to hackers. One of the most important performance indicator is the Server response time. In case of an Http request, the server response time is the amount of time required to load the HTML document of a website from a server so that the client can begin rendering the page. Other key points are the production of reports that must be done once a day.

- **Capacity**: To provide an acceptable service level, the minimum number of concurrent users that the system could deal with at any moment is 4096.
  This number is usually the hard limit of maximum file descriptors (number of the socket opened at the same time) in a Linux based server.
  The system shall refuse the next concurrent user if the maximum capacity level is reached to not affected too much the performances.
  To increase the capacity of the system the service must have some strategies to manage the data flows and clean up memory space when it is not strictly required by the protocol.

- **Scalability**: The system has to be scalable to accommodate the installation of new honeypots and their new patterns to recognize the protocol used in the classification process.

- **Portability**: The system is designed to run on Linux distribution.

- **Reliability**: The system must continue to operate even when the classification of the protocol does not recognize the type of request protocol. All unclassified requests can be redirected to a default low-level honeypot to at least set up the TCP / IP connection through the three-way handshake process.
  The TCP three-way handshake is the method used by any protocol that rides over TCP to set up and negotiate the parameters of connection over anInternet Protocol-basednetwork.
  Usually, after setting the connection, the client sends packets containing the first application-level data that can give an idea of the type of protocol used (Fig. 3.10) analyzing the Tcpdump output.

After a technology analysis to find the programming language that best suited my needs and design constraints, I decided to use the Python programming language and implement two modules called *socket* and *thread*.

3.4.1 Sockets

Sockets are software abstractions that represent the ends, also called nodes, of virtual communication channels between devices.
They are local resources specific to one node in which a process can send and receive data through them (Inter-Process Communication).
Sockets, in the IP/TCP protocol, are associated with the Socket Addresses (IP address:port) which are essential to discriminate a data stream (see Fig. 3.7).

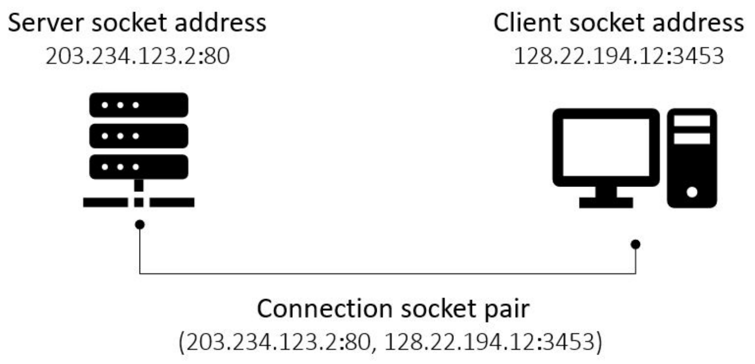


Figure 3.7: Socket connection example

I adopted the Socket module contained in the Standard Library of Python 3.6 to handle the different data packets.

```python
dock_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
dock_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
dock_socket.bind(('', settings[2])) #settings[2] = client port 8888
dock_socket.listen(5)
while True:
    client_socket = dock_socket.accept()[0]
```

Listing 3.3: HoneyPort Proxy - Socket

I created the Socket object by setting different parameters:

- The *AF_INET* is useful to work with IPv4 type addresses.
- The *SOCK_STREAM* allows setting the TCP socket type.
- The *SO_REUSEADDR* allows binding a socket to an address that is already in use.

The *bind()* function connects the socket to the machine address (' ' is equivalent to the localhost address) and to the designated listening port 8888 where all requests from the Aggregator arrive. The TCP port 8888 was chosen at random between 1024 and 65535 to avoid the well-known ones.

3.4.2 Thread

Concurrent programming, also called multithreading, is used when an application requires the execution of many tasks independent from one another, as well as potentially parallelizable operations.
Web servers are a typical example of multithreading application because they have to handle activities of multiple simultaneous users and provide data access to many web browsers and other services at once.
Generally, a parent process manages and generates the subprocesses called threads that run in parallel to achieve a common purpose.
All the threads could share the same address space provided by the parent process, then each thread keeps its state parameters and register values separate from the other threads.
There are many libraries offered by programming languages for concurrency programming but, often high-level implementations deal are hidden to solve some synchronization problems with multi-core processors.
The HoneyPort Proxy is not likely to scale up to many simultaneous clients very well. The queue of the requests may increase exponentially if the system is programmed in single-thread because each connection could be delayed for any number of reasons.
If no other client will be able to obtain service until the blockage is repaired, the performance constraints illustrated previously might not be respected. For instance: the classification of the protocol could be difficult, the response of the honeypot may be slow, the client may not be ready to receive data or simple the network can be interrupted.

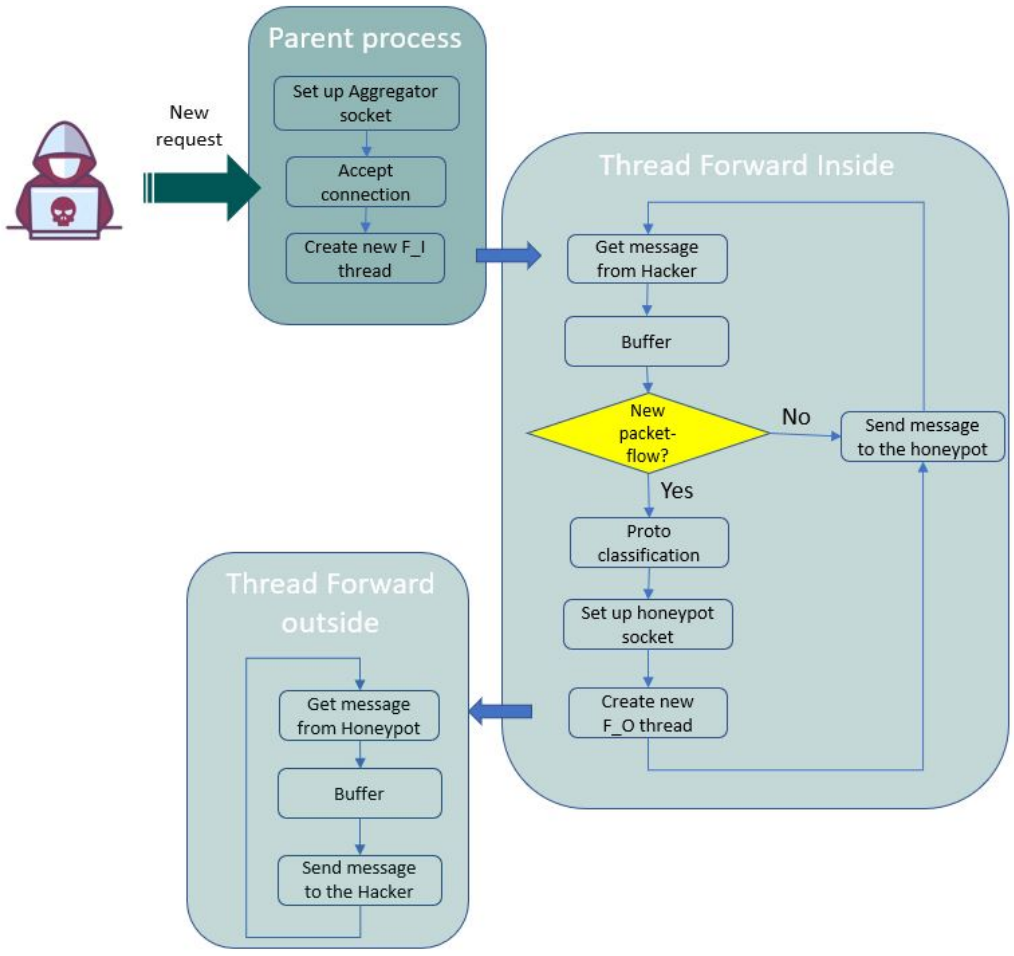


Figure 3.8: Proxy diagram

3.4.3 Proxy details

The HoneyPort proxy is a multi-threading process.
When a new service request comes from the Aggregator, the proxy creates 2 concatenate thread:

- The *Forward-inside* thread: it handles and classifies the messages sent by the client to the honeypot.
- The *Forward-outside* thread: it handles the messages sent by the honeypot toward the client.

When the system starts, the parent process set the socket of the Aggregator (TCP port 8888) where all the incoming message will pass through.
When a new request arrives, the connection is accepted and the parent process creates a new instance of Forward-Inside thread.
The Forward-Inside thread gets the message request and it checks if this data packet belongs to an ongoing connection or if it is related to a new service request.
In case of an in-progress connection, the message is forwarded directly to the previously selected honeypot. In case of a new service request, the script selects the most suitable honeypot based on the TCP protocol.
The protocol classification relies on pattern matching between the firsts data packets from the client and some data patterns stored in the system that are typically related to a specific protocol.
For instance, The Secure Shell (SSH) Transport Layer Protocol defines that "When the connection has been established, both sides MUST send an identification string. This identification string MUST be: SSH-protoversion-softwareversion SP comments CR LF"[19] and this information can characterize a SSH connections.
Once that the most suitable honeypot is selected, then the Forward-Inside thread sets up the socket to the trap and creates a new instance of Forward-Outside thread.
Finally, the thread redirects the client message to the honeypot and listens for new packets. Forward-Outside thread simply waits for the reply messages from the honeypot and it redirects that traffic to the correct client.
Those two loop ends when the communication is closed by the client or by the honeypot.
This approach of two nested threads was undertaken to optimize the number of opened sockets for each connection because the thread opens the socket only when it is necessary, instead of set up sockets for all the honeypots before the protocol classification.

3.5 Responder

The Responder is the software component that links all the honeypots installed in the system.
This component is modular because it is composed of units that can be added, deleted,

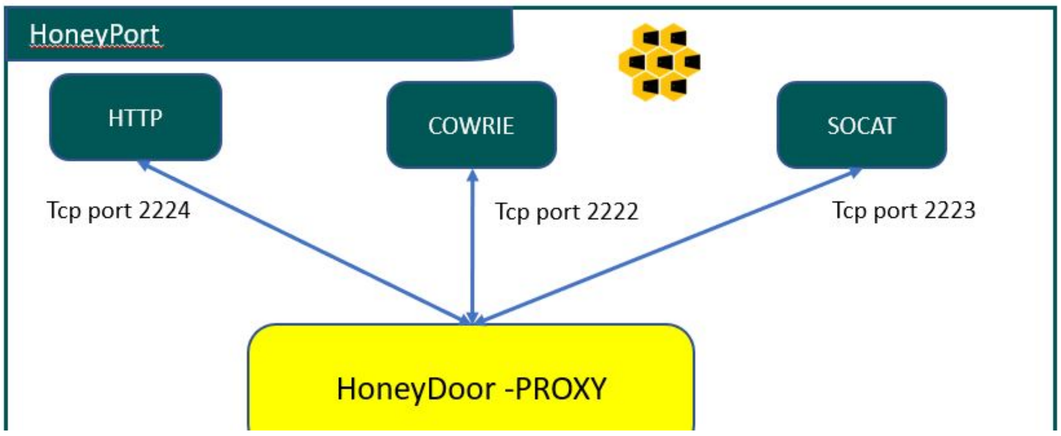


Figure 3.9: Honeyport-Responder

modified without affecting the rest of the system.
The installed units are:

- *Cowrie*: honeypot that simulates SSH protocol
- *Http honeypot server*: honeypot that simulates HTTP web server
- *Socat*: low-level honeypot utilized in the not classified protocol requests.

A brief explanation of the honeypots details will follow.

Cowrie

The operation of this SSH honeypot was presented in chapter 2. I decided to run in an instance of this software through a docker container at the TCP port 2222 (ref 3.4).

```
sudo docker run -p 2222:2222 cowrie/cowrie
```

Listing 3.4: Cowrie docker instantiation

Docker container allows to package up an application with all the parts it needs, like libraries and other dependencies, and ensure that every instance of the application behaves in the same way.
The TCP port 2222 was chosen at random between 1024 and 65535 to avoid the well-known one.

Http honeypot server

To create a website that is as close as possible to a real web resource, I decided to use a Web scraping software that can access the World Wide Web directly using the Hypertext Transfer Protocol to extract data from an online website.
The choice of web harvesting software fell on HTTrack that is widely used and available for both Windows and Unix systems.
”HTTrack is best suited for an exploratory acquisition of a small number of sites. It modifies the links in retrieved content to create a self-consistent set of files that can be directly viewed without the need of a separate viewing tool.”[20]

```
httrack https://smartdata.polito.it/ -O .
```

Listing 3.5: Httrack command

In the HoneyPort implementation at Politecnico di Torino, I extracted the Website data from my research group SmartData (ref: 3.5).
All the data gathered by HTTrack are then served on port 2224 through the http.server module installed in python3.6 (ref: 3.6).

```
python3.6 -m http.server 2224
```

Listing 3.6: http.server by python3.6

It creates and listens at the HTTP socket, dispatching the requests to a handler and reply with the stored data.

Socat

Socat, also known as ”Swiss army knife”, is a very popular multi-purpose networking tool developed by Gerhard Ridger.
It includes many networking functionalities, such as Chat servers, Revers shells, Key generators, Certificate makers and Packet crafting.
During the protocol classification of a new requests, if any protocol between the supported patterns is not found as matching, the Multi-Purpose responder Socat is able to establish a low-level connection anyway.

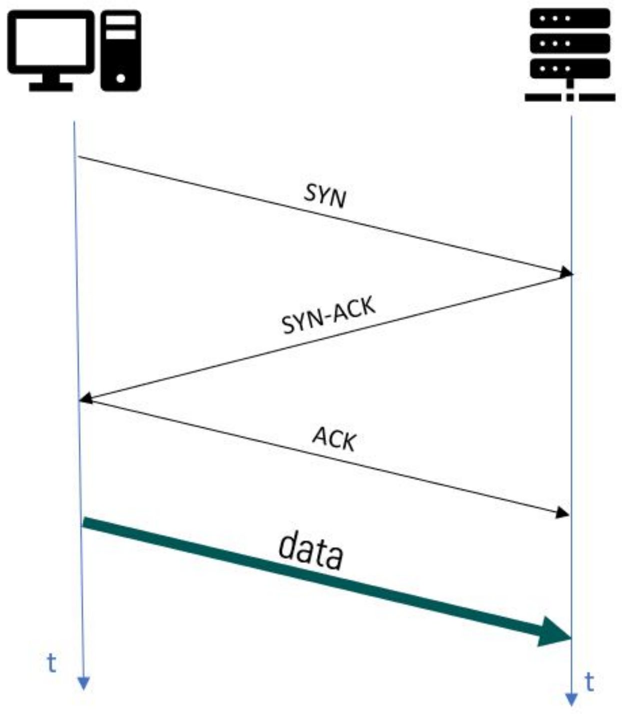


Figure 3.10: TCP three-way handshake

In this way, the Socat is able to proceed with the connection by the three-way hand-shaking and the honeyport system can sniff the consequently first data packets from the suspicious users (Fig. 3.10).
Those captured data can have a great impact to understand what are the potential protocols honeypot that need to be implemented in the future.

Chapter 4

Implementation

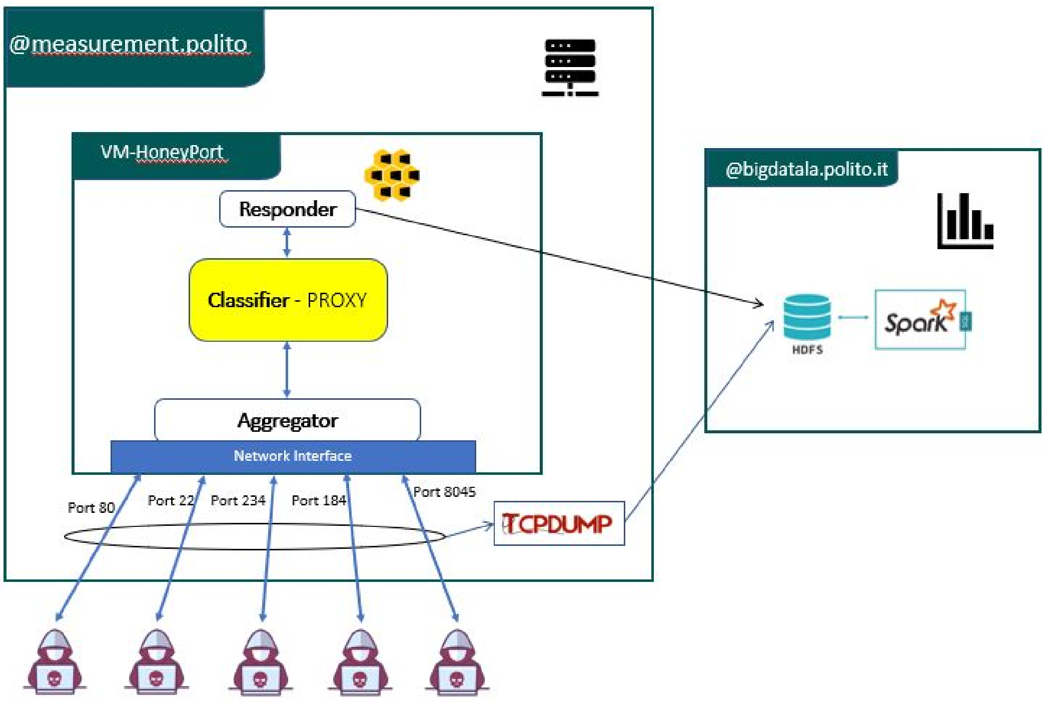


Figure 4.1: Architecture

The following section will present the IT infrastructure granted by the BigData lab research group for the implementation of the HoneyPort system and data analysis of sniffed data packets. This structure will include the description of the following three properties for each module:

- **Name**: the name of the module.
- **Responsibilities**: the role of the module in the system.
- **Implementation information**: details related to the modules implementation that is relevant for managing its development.

The main modules are *VM-HoneyPort*, *measurement.polito.it* and *bigdata.polito.it*.

4.0.1 VM-HoneyPort

Here I set up the HoneyPort system and the main tasks of this module are:

- Running an instance of HoneyPort system.
- Network endpoint for the hacker trap.
- Security management.

I have created an Oracle virtual machine with the Linux operating system Ubuntu 18.04.2 LTS (GNU / Linux 4.15.0-55-generic x86_64), 10 GB of memory allocation, Intel (R) Xeon (R) E5-2640 CPU 0 @ 2.50GHz and a gigabit ethernet interface.
I set a public IP (130.192.8.254) previously used in the Darknet network so, the traffic directed to this address was not filtered by any firewall.
I opted for a virtual machine because the instances of the honeypots inside the HoneyPort system shall be controlled by an expert hacker who, by discovering the trap, could perform malicious actions. Moreover, for this reason, the virtual ethernet interface was isolated from the local area network in order to avoid possible attacks on the internal systems of the Turin Polytechnic.

4.0.2 measurement.polito.it

In this machine, I deploy the backend functions and the main tasks of this module are:

- Hosting the VM.
- Daily backup of the HoneyPort logs.

- Packet Sniffing.

This machine is used for many purposes within the BigData Lab research group. I had the chance to use this hardware as a host for the VM-HoneyPort virtual machine.
I created a bash backup script to extract the virtual machine logs from the *VM-HoneyPort* system to *measurement.polito.it* server through the SSH backdoor on the port 65535 (ref: 4.1).

```
# m h  dom mon dow   command
0 10 * * * rsync --remove-source-files --exclude 'cowrie.json' --
    exclude 'cowrie.log' -rv -e "ssh -p65535" root@130.192.8.254:/
    home/cowrie/cowrie/var/log/cowrie/ /home/rootbigdata/honeypot/ >>
     /tmp/cronout.log 2>&1
```

Listing 4.1: Backup script on Crontab

The backdoor connection allows us to avoid false-positive during the data analysis of the SSH connections.
In order to automatically run the script every day at 10 o'clock, I used the Crontab scheduling service of Linux operative system.
The *measurement.polito.it* networks interface can intercept all the data traffic intended for the HoneyPort system installed in the Virtual Machine.
We decided to sniff the virtual network interface through an instance of Tcpdump in the *measurement.polito.it* server for further data analysis (Fig. 4.2).
Tcpdump is a common packet analyzer that can make a dump of the network interface and save all the records in a special format called Pcap files.

4.0.3 bigdata.polito.it

Within this cluster,we analyzed the data and the main tasks of this module are:

- Big Data analysis.
- nDpi analysis.

The amount of data deriving from Tcpdump and the honeypot log files can grow considerably and a simple database or SQL programming language may not be able to handle this amount of information.

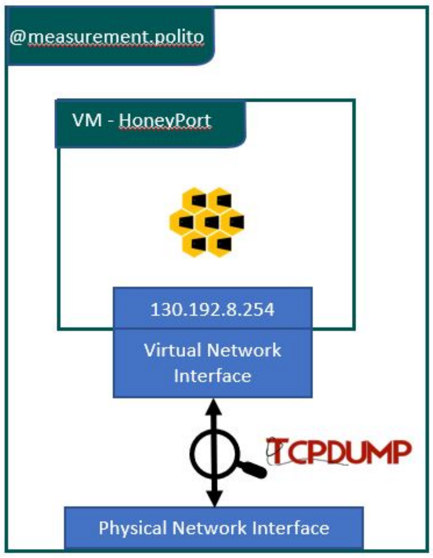


Figure 4.2: Network Interfaces

We have therefore decided to move the data to an HDFS database (ref chap:2) installed on the bigdata.polito.it machine.
I did the Big Data analysis thorugh a Python code employing the PySpark APIs that exposes the Spark programming model to Python language.

Chapter 5

Results

In the following chapter, we will analyze the data gathered by the HoneyPort system based on two levels:

- *Raw packet analysis*: Deep investigation about the sniffed communication packets.
- *Honeypots analysis*: High-level investigation about the honeypot logs installed in the HoneyPort.

The analysis was made comparing the data collected by:

- *Darknet Analysis*: an examination of the traffic received at an IP address selected from a pool of addresses reserved for the investigation about the darknet implemented at the Politecnico di Torino.
- *Honeypot Analysis*: an examination of the data connection received at the *measurement.polito.it* Host where was installed an implementation of Cowrie (SSH honeypot) and a Web server responder.
- *HoneyPort Analysis*: an examination of the data connection received at the *measurement.polito.it* Host where was installed an implementation of **HoneyPort** system.

During June 2019, were gathered the data related to Darknet and HoneyPort systems. The honeypot system gathered the communication data during August 2019.

5.1 Raw packet analysis

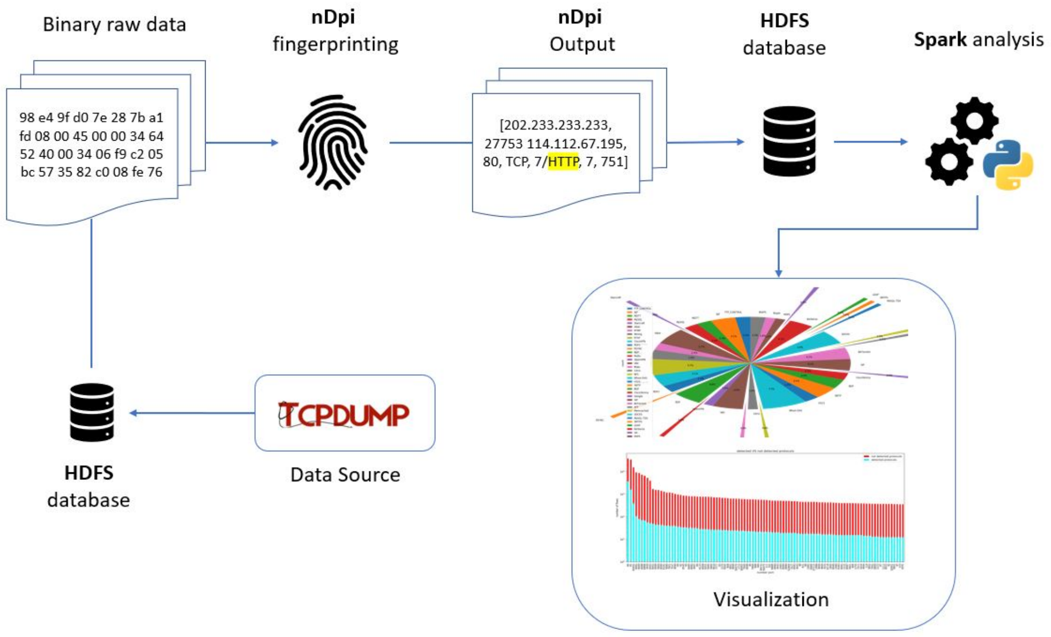

Binary raw data
98 e4 9f d0 7e 28 7b a1 fd 08 00 45 00 00 34 64 52 40 00 34 06 f9 c2 05 bc 57 35 82 c0 08 fe 76
nDpi fingerprinting
nDpi Output
[202.233.233.233, 27753 114.112.67.195, 80, TCP, 7/HTTP, 7, 751]
HDFS database
Spark analysis
TCPDUMP
Data Source
HDFS database
Visualization

Figure 5.1: nDpi Analysis Workflow

All the raw data comes from an instance of Tcpdump that sniffed the traffic through the network interface of the HoneyPort Virtual machine and stored them by the pcap format within the distributed HDFS database (Fig. 5.1).
Pcap files are not human-readable, so they were processed by the deep packet inspection nDpi to extract useful information for each TCP stream. Such as IP source address, IP destination address, transport layer model, Internet protocol type, and others.
Given the amount of data to be processed, we adopted the Apache Hadoop MapReduce Streaming process in order to parallelize the work in the BigDataLab cluster.
The Map method splits the input data into chunks and maps them to different workers who process the information simultaneously. I decided not to apply the Reduce function in order to avoid huge output files and to speed up the task.
To get a better visualization of the results, I created a Pyspark-based script that allowed me to have a Python interface to the Apache Spark environment.

System	Number of attack flows
Darknet	118168
Honeypot	233249
HoneyPort	306202

Table 5.1: comparison about Number of attacks

The most contacted system was HoneyPort, followed by honeypot and darknet, as evident from Tab 5.1. In particular, the honeypot system had an increase of 97% of service requests compared to the darknet and the HoneyPort system had an increase of more than 159% of attacks compared to the darknet.

From Fig. 5.2 to Fig. 5.4 it can be seen the number of TCP streams received in specific ports, for space reasons only the 50 most contacted ports are displayed. can be observed on the ordinates the number of TCP connection flows and the relative server ports contacted by the client on the abscissa.

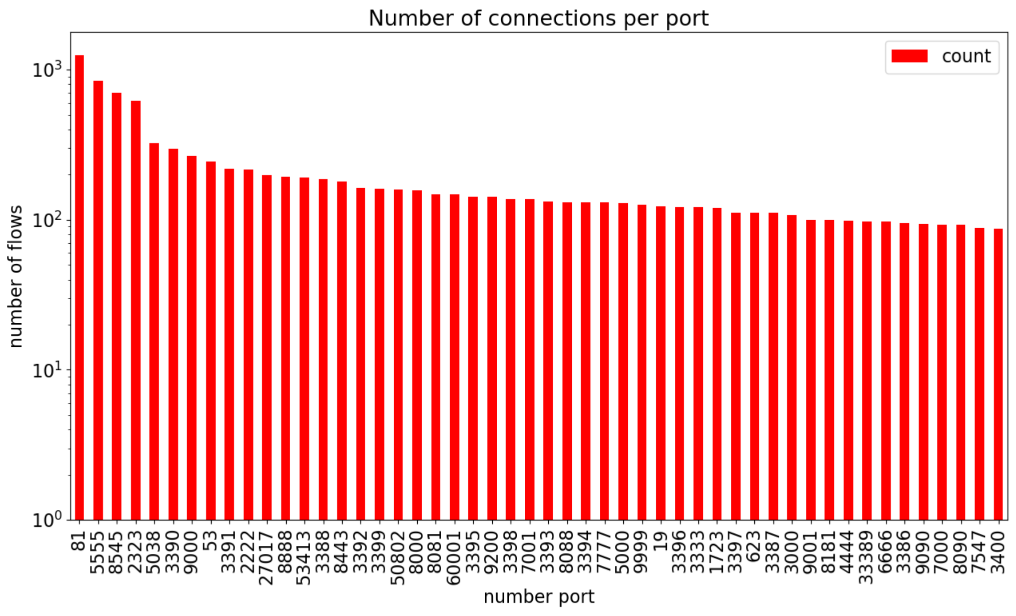


Figure 5.2: Protocol-Port attacks darknet

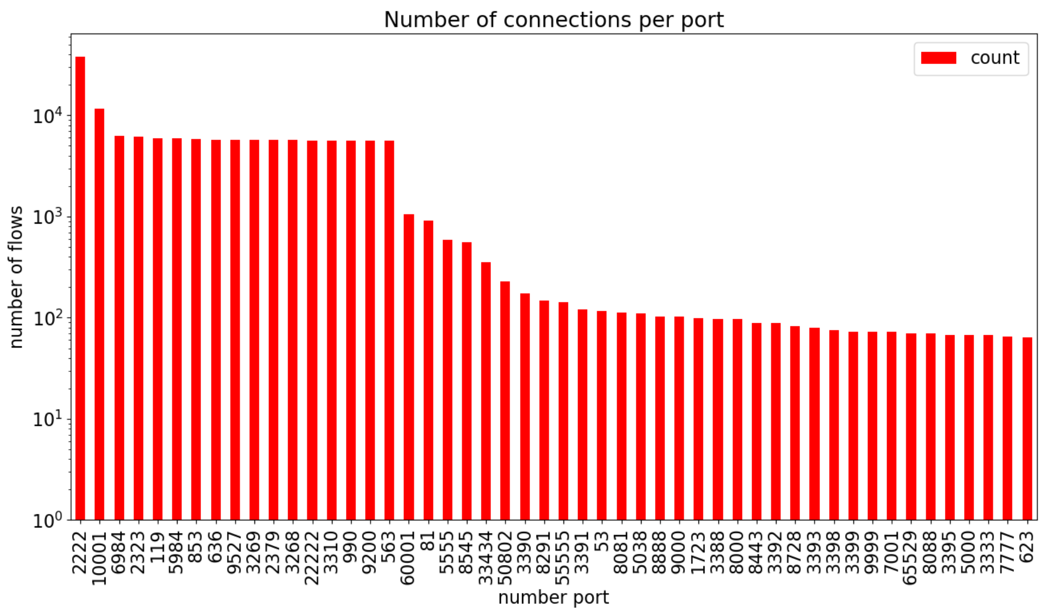


Figure 5.3: Protocol-Port attacks honypot

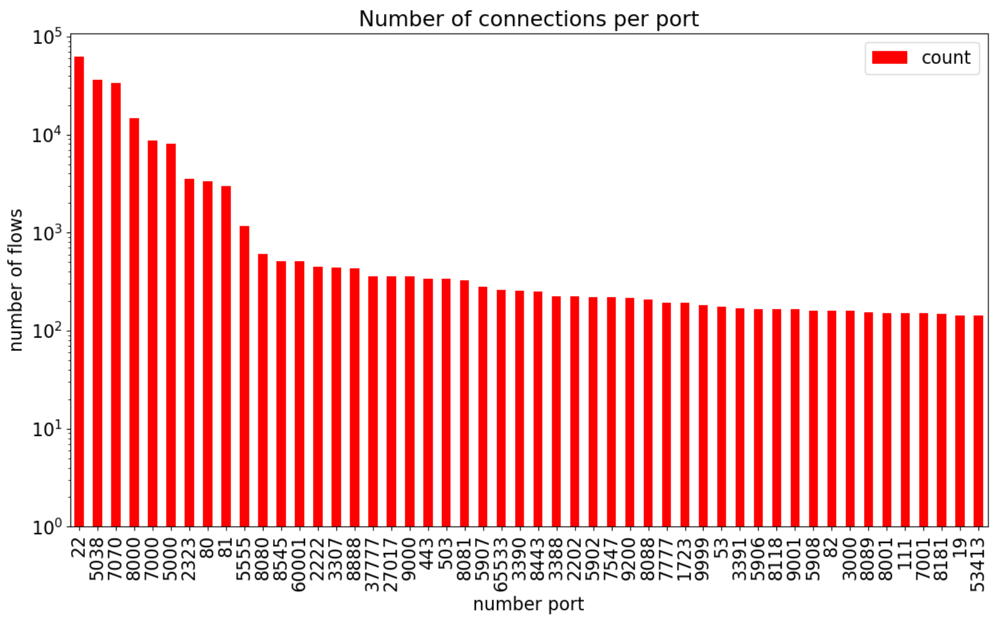


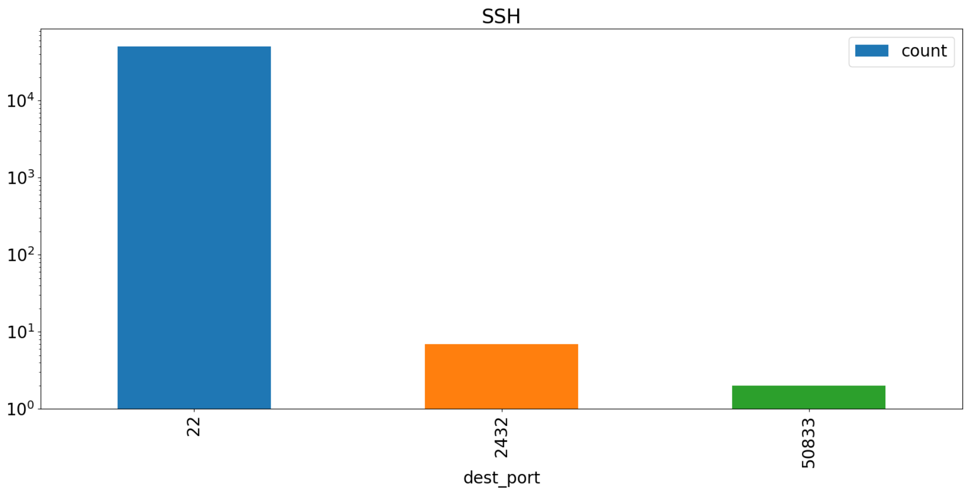
Figure 5.4: Protocol-Port attacks HoneyPort

All the requests per port do not overcome 1000 flows in the case of the darknet environment (Fig. 5.2), except for port 81 where could be observed a little more than 1000 flows.
Interesting to see so many request addressed to an IP address not used by any device, a sign that the client are probably enumerating possible vulnerabilities to exploit and scientific researches based on Darknets.
As already mentioned, the machine employed as a host for the Honeypot test marked a considerable increase in attempts. The most contacted port was 2222 in which the SSH Cowrie honeypot was listening with more than 10^5 connections.
Remarkable is the amount of requests on port 1001 which, based on some malware collected, could be linked to some known cyber-attack such as the remote access trojan called *Backdoor.Zdemon.126* which targets windows systems. It is clear that much additional work is required before a complete understanding can be reached.

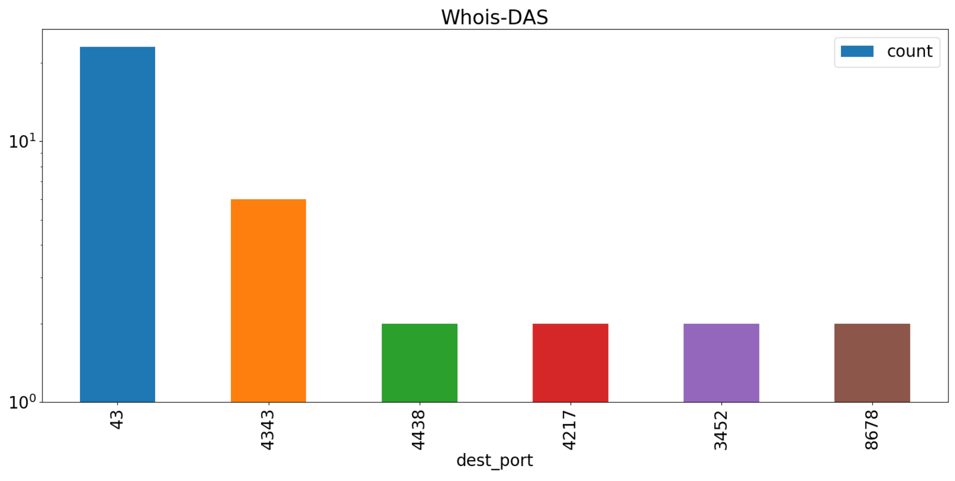
After the implementation of the HoneyPort system, the number of ports contacted by clients has increased remarkably, as can be seen in Fig. 5.4 and this is a confirmation of the goodness of the strategy acquired by the HoneyPort.
These graphs can contain many research ideas, for example, very impressive is the number of activities on the TCP port 60001.
The range 49152 to 65535 contains TCP dynamic or private ports that can be used by private and customized services or for temporary reasons, so we expect a modest number of connections on this endpoint.
however, TCP ports of numbers as big as 60001 and 37777 were the targets of many attacks. Looking for some malware repository information, I found that a Trojan called Backdoor.Linux.Trinity largely attacked many Linux-based systems through TCP port 60001.
I quote the EventTracker KB source: ”This malware can be manipulated remotely to control infected machines and launch what constitutes a denial of service (DoS) attack against systems running Linux or Unix. This malware also continually attempts to connect to certain IP addresses, causing huge network traffic and infected systems to slow down.”[21]
Protocol identification is more effective when the HoneyPort system is in operation, which means that the system handles a greater variety of service requests and the num-

ber of packets exchanged for each communication is greater than traditional monitoring methods.

I obtained other information about the relationship between a specific application protocol and the TCP port contacted, as illustrated in figure 5.5.



(a) SSH



(b) Whois-Das

Figure 5.5: nDpi protocols detail

The SSH requests come not only on the well-known port 22 but also on both 2432 and 50833, which are probably the SSH default ports of some specific system.
Usually, the hackers have to supply some identifying information to set up a domain for his or her command and control server so, this may be that case. That data typically contains some personal information about the entity responsible for maintaining the registry such as the registrant's name, the e-mail address and other contact data.
The *WHOIS* protocol allows hackers to exploit this registration data for a given domain and re-use this information to hide his identity. This method to gather sensible information is still applied, as indicated in figure 5.5b.
For example, some attack strategies plan to send a crafted request to port 4343 TCP in order to execute arbitrary code on the system with Web user privileges.
Select the right listening port for the honeypot implementation is crucial.

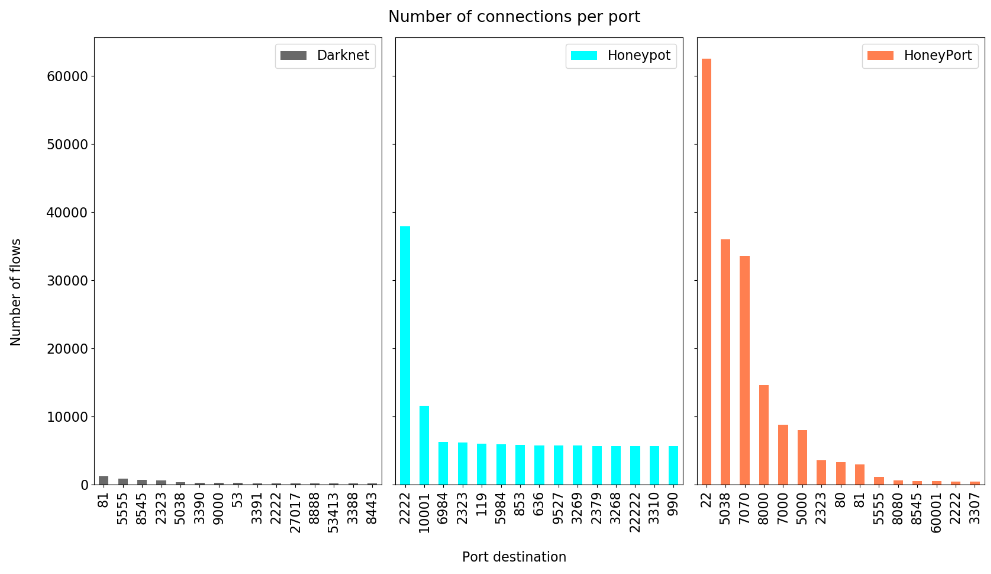


Figure 5.6: Comparison top 15 contacted ports

we decide to implement the honeypot Cowrie at the port 2222 during the honeypot environment test because the most contacted port by SSH requests, over the Darknet data analysis, between 2222 and 22 was 2222 one, as can see in Fig. 5.6.
Therefore, during the honeypot environment, the port 2222 received a lot of SSH requests. However, this decision was wrong because the HoneyPort system implementation received much more SSH requests on port 22 than port 2222.

This means that the HoneyPort system might facilitate the deployment phase optimizing the choice of listening port service.

The following graphs, from 5.7 to 5.9, are a comparison made to emphasize the type of protocols classified by the nDpi requested by the clients.

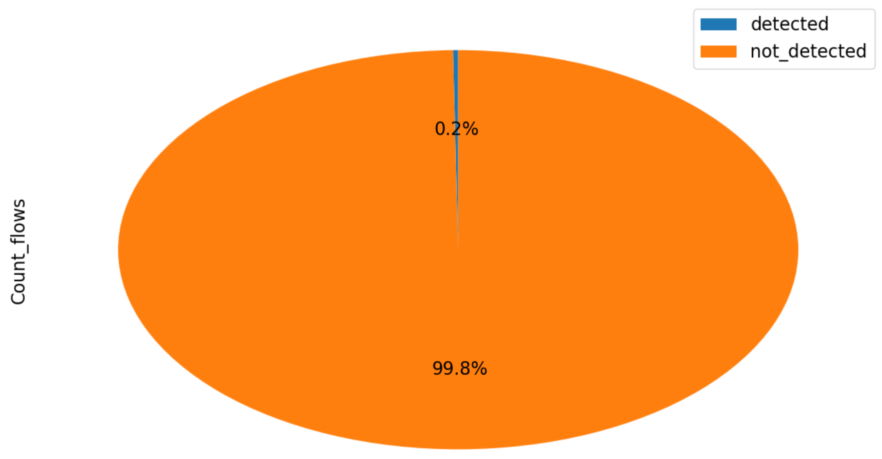


Figure 5.7: Detected VS Not Detected Protocol darknet

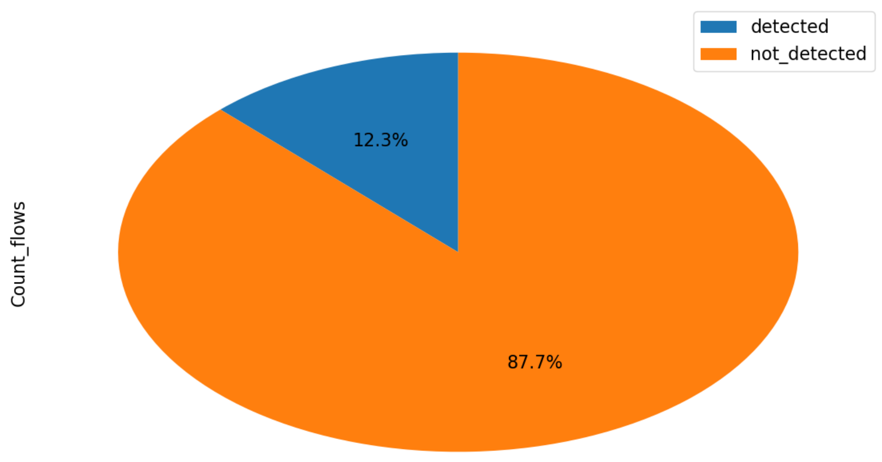


Figure 5.8: Detected VS Not Detected Protocol honeypot

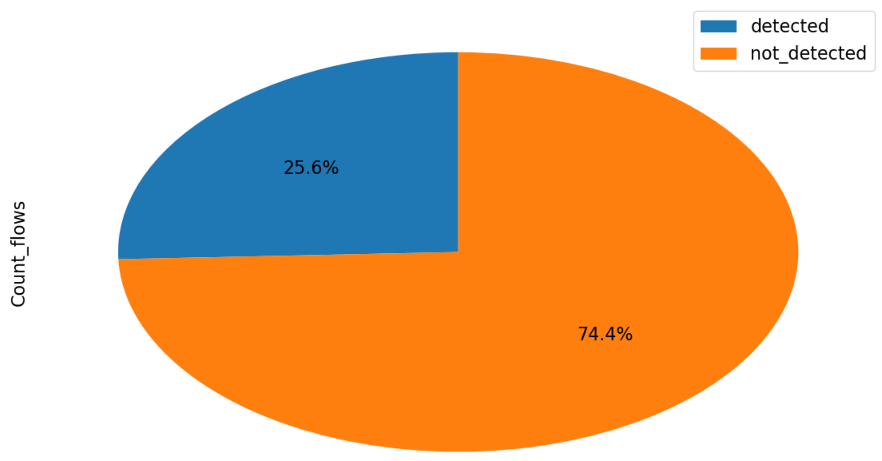


Figure 5.9: Detected VS Not Detected Protocol HoneyPort

nDpi information extraction can be much more effective if the system is able to retain the clients for as long as possible in the trap because there are more data packages to be analyzed.
The extracted data by nDpi on the darknet environment are almost negligible and have been classified only 0.2% of the communications, as illustrated in Fig. 5.7.
The classification of the protocol reaches 12.3% of all the analyzed communication In case of honeypot environment, as shown in Fig. 5.8 Nevertheless, the more significant improvement in the amount of classified communication protocol is related to the HoneyPort system, which reaches about 26% of the analyzed traffic flow.
The HoneyPort has doubled the effectiveness of nDpi's protocol recognition with respect to commonly used honeypots, as evident from the figure 5.9.

The graphs from Fig. 5.10 to Fig. 5.12 provide the analysis of the type of protocol detected per each environment. This information can be useful to decide honeypots to implement in the HoneyPort updates.

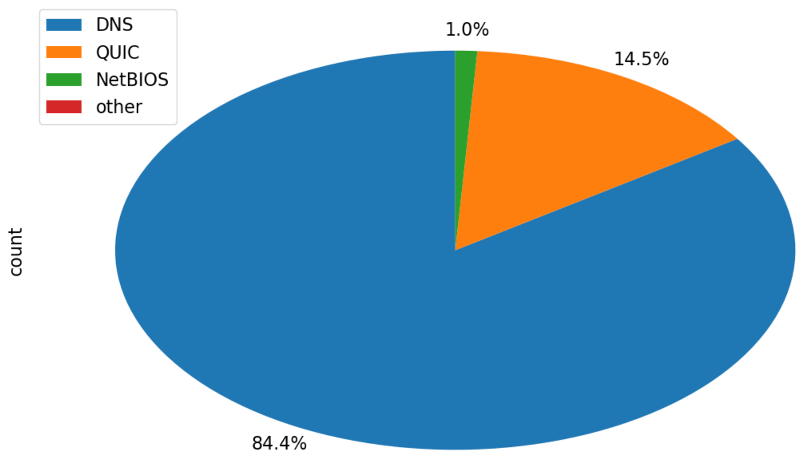
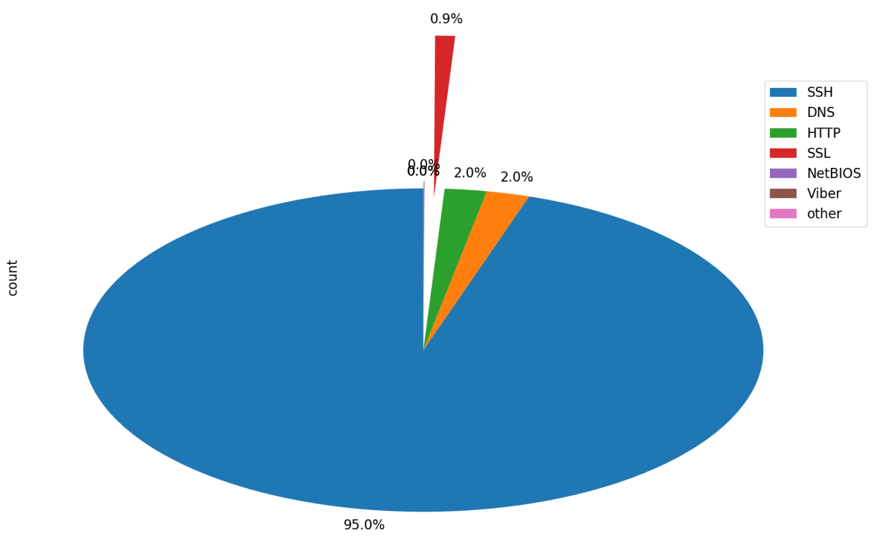
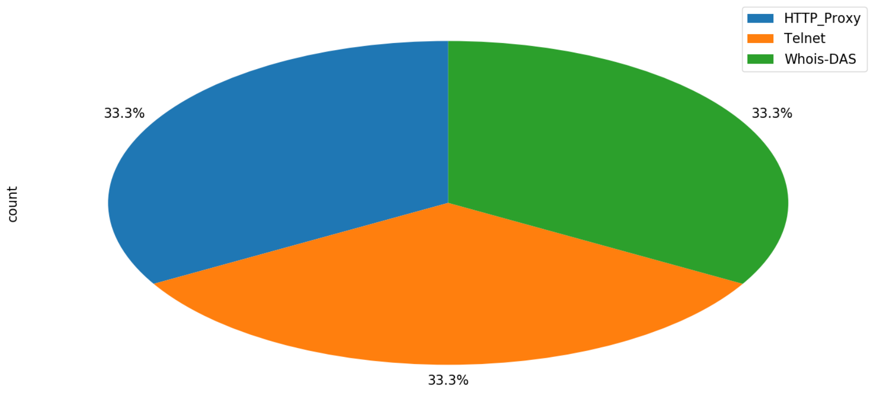


Figure 5.10: Protocol requests darknet

The majority of the classified protocol of Darknet environment are DNS requests



(a) Protocol requests Honeypot

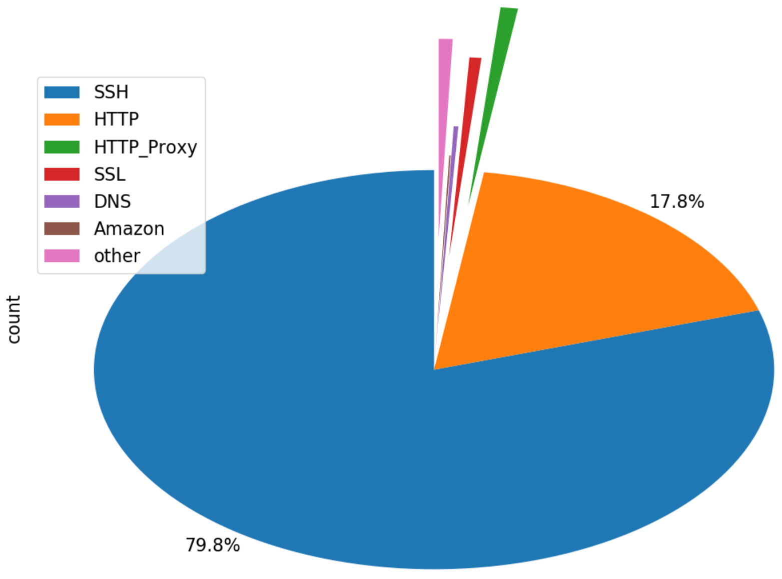


(b) Other Protocol requests Honeypot

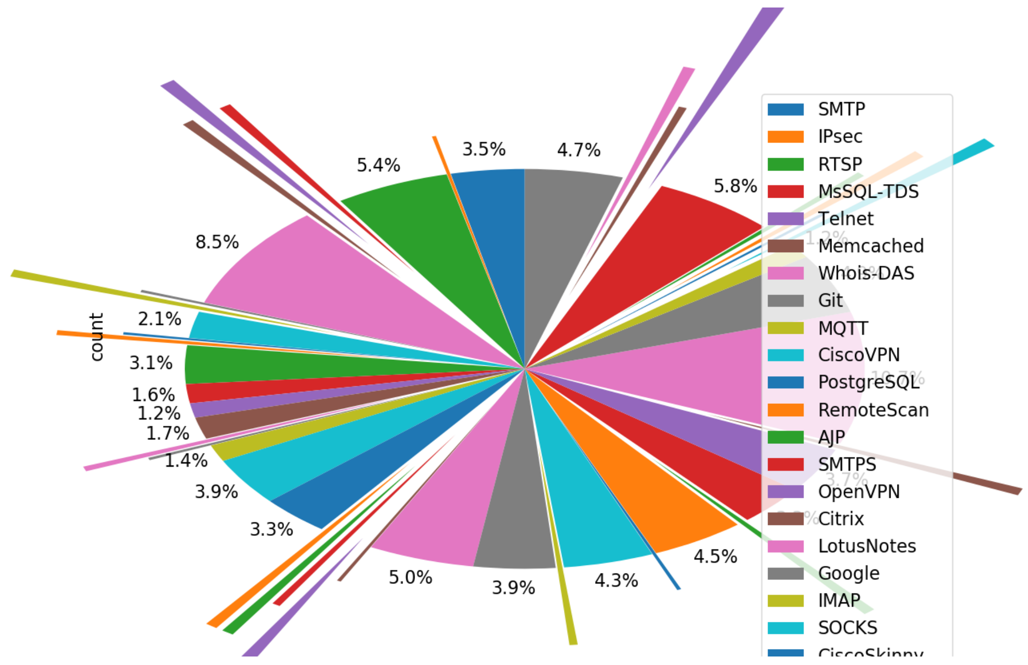
Figure 5.11: Protocol requests HoneyPort

and data related to the protocol designed by Google called QUIC (Fig. 5.10).
The amount of classified flow protocol in the Darknet is very small so, this means that it cannot be representative of all the service requests on this type of environment.
As expected, in the honeypot environment the majority of the flow connection belongs

to the SSH connection (Fig. 5.11) since the principal implemented system was Cowrie.
There are 9 different types of detected protocol like HTTP and DNS requests.
Much more interesting are the result of the HoneyPort system, that shows a tremendous increase in the type of service requests (Fig. 5.12).
The SSH and HTTP remain the most common communication data with 43% and 75% respectively but, we recorded more than 50 types of different service requests.



(a) Protocol requests HoneyPort

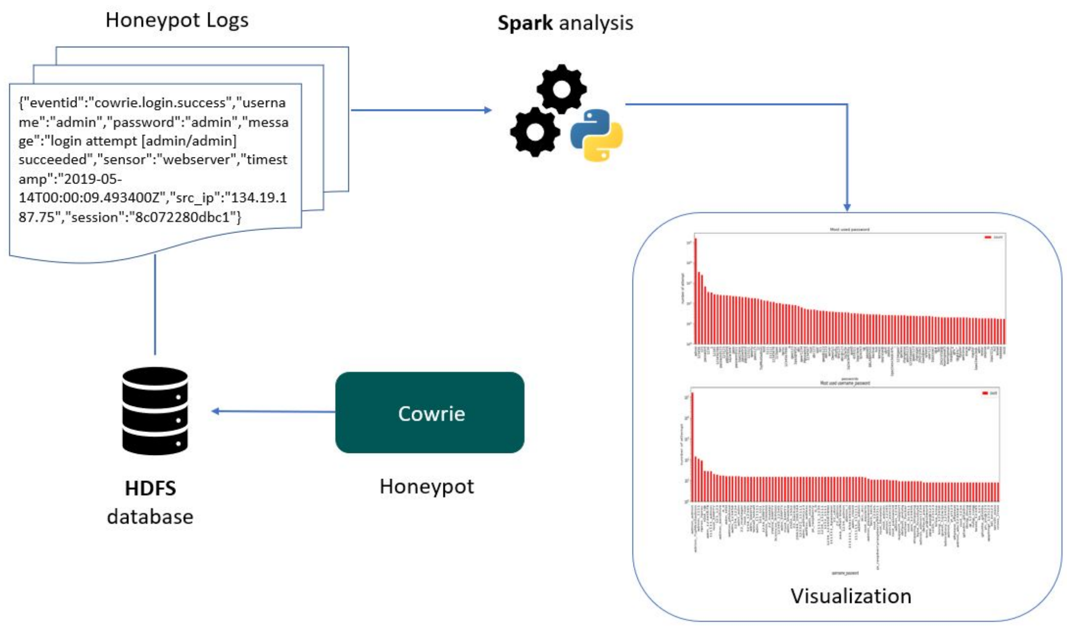


(b) Other Protocol requests HoneyPort

Figure 5.12: Protocol requests HoneyPort

5.2 Honeypots analysis

In this section, I analyzed the output coming from the Responders. The most significant results are related to the implementation of Cowrie for the SSH service. The



```
{"eventid":"cowrie.login.success","username":"admin","password":"admin","message":"login attempt [admin/admin] succeeded","sensor":"webserver","timestamp":"2019-05-14T00:00:09.493400Z","src_ip":"134.19.187.75","session":"8c072280dbc1"}
```

Figure 5.13: Cowrie Analysis Workflow

methodology used (Fig. 5.13) for data analysis is similar to that illustrated in the previous chapter, using the Cowrie output in JSON format to optimize the number of steps. The Cowrie log files are very detailed for each session, so after storing them within the distributed HDFS database, I processed the data through a script based on Pyspark with the aim of obtaining useful information. Figures 5.14,5.15 and 5.16 portray in order: Username, Password and Username/Password most attempted by the attackers. Most of the clients tried ”admin” as Username with more than 10^5 attempts, followed by ”tests” and ”root” with almost 10^3 tries.

Data in Fig. 5.15 suggest that the three main attempted passwords are ”admin” (with more than 10^5 tries), ”12345” (with almost 5000 tries) and ”123” (with more than 10^3 tries).

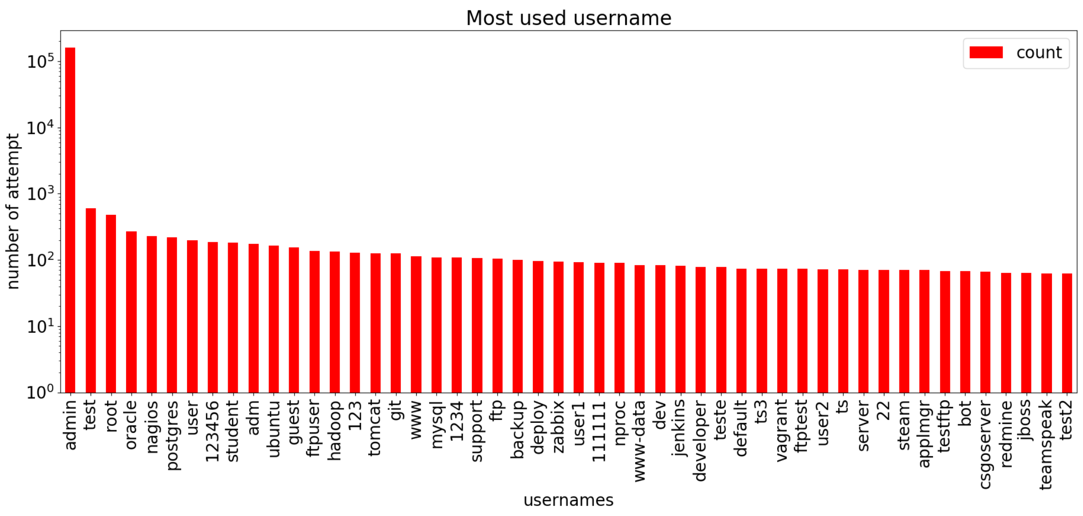


Figure 5.14: SSH Username attempts

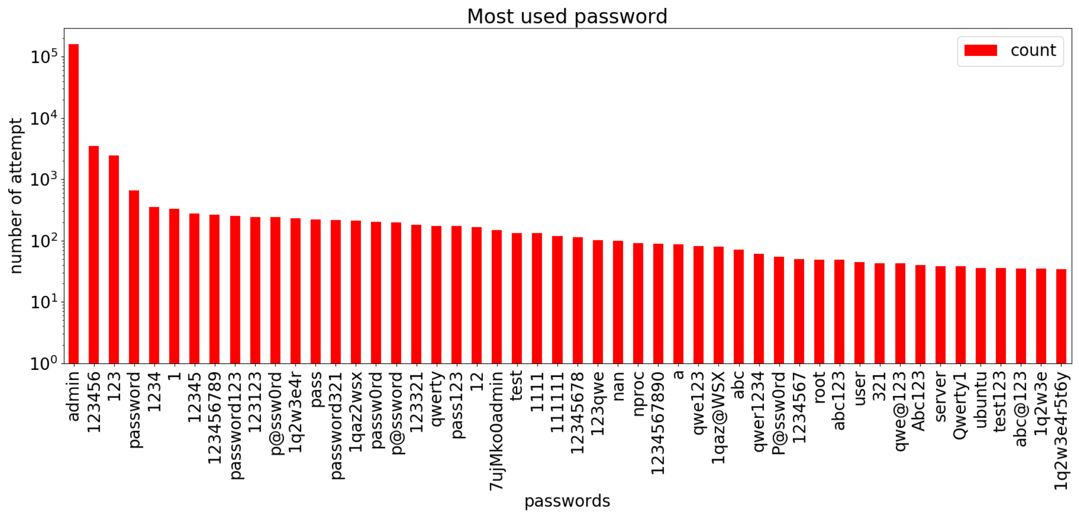


Figure 5.15: SSH Password attempts

The most common way to infect devices is the dictionary attack. These credentials can be generic as ”admin/admin” or addressed to particular hardware such as ”admin/7ujMko0admin”, as listed in Fig. 5.16.
The most effective way to defend against these type of attacks is to know which are the exploited ”username/passwords” and change as soon as possible this credentials.

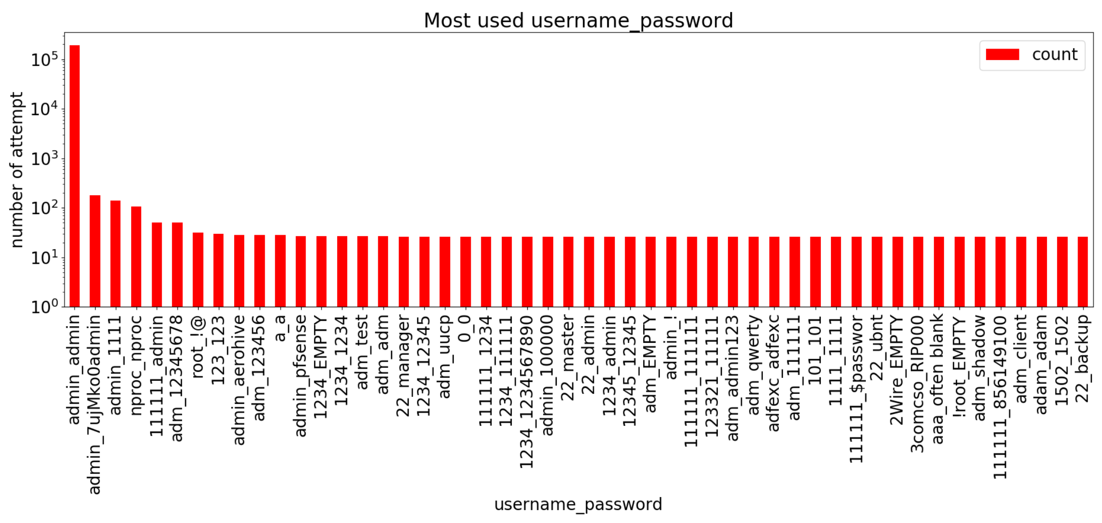


Figure 5.16: SSH Username-Password attempts

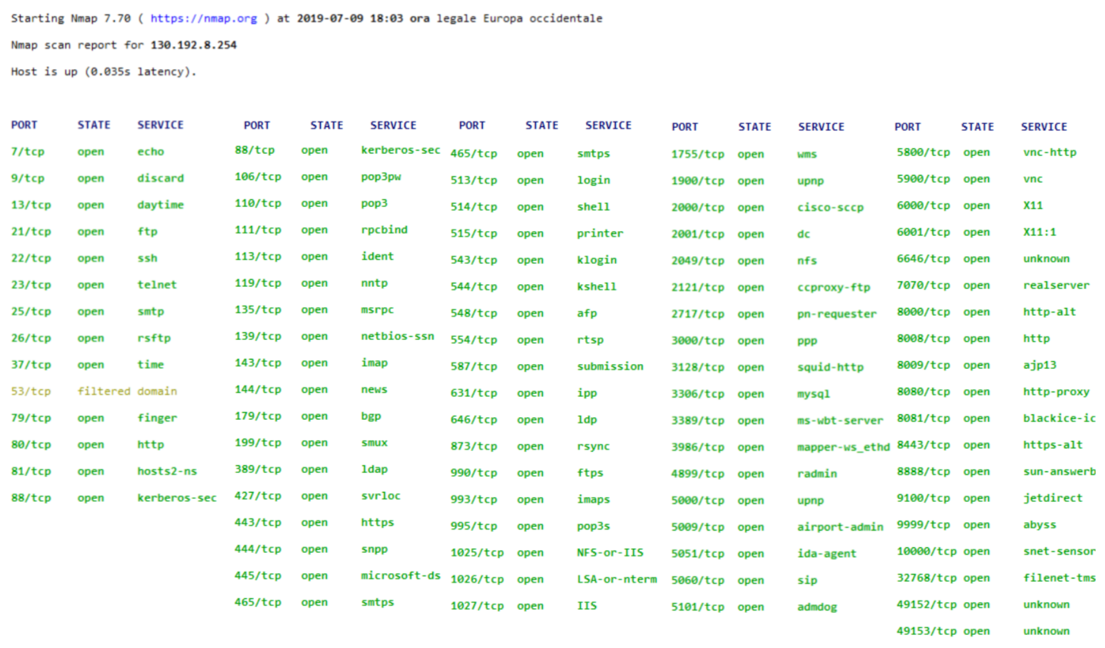
Chapter 6

Conclusions and future work

6.1 Conclusion

In conclusion, the thesis posed the problem of low visibility that affects the majority of the honeypot project in the literature and offered a new methodology to solve it using a system able to analyze the requests and redirect them to the most suitable honeypot. Prior work has documented the effectiveness of gathering information through the application of a honeypot system concerning new cyberattacks that could affect the privacy and functionality of many devices sold on a large scale.
However, these studies have relied on the port assigned to specific server services by the Internet Assigned Numbers Authority (IANA) constraint that which in today's applications are no longer reasonable.
In this study, I decided to inherit the well-established honeypot functionalities and add to them the possibility of dealing with requests that address not conventional UDP/TCP communication endpoints.

The possibility of classifying the protocol for each communication and redirect them to the most suitable honeypot that allows the HoneyPort system considerably increases the level of flexibility and the number of attacks that are traced with respect to the implementation of a single trap, as reported by the analyzes.
A key point of HoneyPort is the ability to exposing all the communication endpoint open as if there is a listening service at every TCP/UDP port (Fig. 6.1).
The analysis of the communication data indicates that the benefits gained from the ex-



```
Starting Nmap 7.70 ( https://nmap.org ) at 2019-07-09 18:03 ora legale Europa occidentale

Nmap scan report for 130.192.8.254

Host is up (0.035s latency).

PORT      STATE    SERVICE       PORT      STATE  SERVICE       PORT      STATE  SERVICE       PORT      STATE  SERVICE          PORT       STATE  SERVICE
7/tcp     open     echo          88/tcp    open   kerberos-sec  465/tcp   open   smtps         1755/tcp  open   wms              5800/tcp   open   vnc-http
9/tcp     open     discard       106/tcp   open   pop3pw        513/tcp   open   login         1900/tcp  open   upnp             5900/tcp   open   vnc
13/tcp    open     daytime       110/tcp   open   pop3          514/tcp   open   shell         2000/tcp  open   cisco-sccp       6000/tcp   open   X11
21/tcp    open     ftp           111/tcp   open   rpcbind       515/tcp   open   printer       2001/tcp  open   dc               6001/tcp   open   X11:1
22/tcp    open     ssh           113/tcp   open   ident         543/tcp   open   klogin        2049/tcp  open   nfs              6646/tcp   open   unknown
23/tcp    open     telnet        119/tcp   open   nntp          544/tcp   open   kshell        2121/tcp  open   ccproxy-ftp      7070/tcp   open   realserver
25/tcp    open     smtp          135/tcp   open   msrpc         548/tcp   open   afp           2717/tcp  open   pn-requester     8000/tcp   open   http-alt
26/tcp    open     rsftp         139/tcp   open   netbios-ssn   554/tcp   open   rtsp          3000/tcp  open   ppp              8008/tcp   open   http
37/tcp    open     time          143/tcp   open   imap          587/tcp   open   submission    3128/tcp  open   squid-http       8009/tcp   open   ajp13
53/tcp    filtered domain        144/tcp   open   news          631/tcp   open   ipp           3306/tcp  open   mysql            8080/tcp   open   http-proxy
79/tcp    open     finger        179/tcp   open   bgp           646/tcp   open   ldp           3389/tcp  open   ms-wbt-server    8081/tcp   open   blackice-ic
80/tcp    open     http          199/tcp   open   smux          873/tcp   open   rsync         3986/tcp  open   mapper-ws_ethd   8443/tcp   open   https-alt
81/tcp    open     hosts2-ns     389/tcp   open   ldap          990/tcp   open   ftps          4899/tcp  open   radmin           8888/tcp   open   sun-answerb
88/tcp    open     kerberos-sec  427/tcp   open   svrloc        993/tcp   open   imaps         5000/tcp  open   upnp             9100/tcp   open   jetdirect
                                 443/tcp   open   https         995/tcp   open   pop3s         5009/tcp  open   airport-admin    9999/tcp   open   abyss
                                 444/tcp   open   snpp          1025/tcp  open   NFS-or-IIS    5051/tcp  open   ida-agent        10000/tcp  open   snet-sensor
                                 445/tcp   open   microsoft-ds  1026/tcp  open   LSA-or-nterm  5060/tcp  open   sip              32768/tcp  open   filenet-tms
                                 465/tcp   open   smtps         1027/tcp  open   IIS           5101/tcp  open   admdog           49152/tcp  open   unknown
                                                                                                                                49153/tcp  open   unknown
```

Figure 6.1: Nmap result on the HoneyPort Host

posure of all the ports may increase the number of recorded cyber attacks.
In addition, the great variety of requests received and the amount of data collected through the Honeyport provide compelling evidence that the Proxy algorithm implemented to the system could improve the port coverage of the existing honeypot projects.
Although the experiment's environment was quite realistic, specific stress tests are still required to evaluate the stability of the implementation.

6.2 Future work

Possible improvement of the HoneyPort Proxy is to change the software design pattern multi-thread into thread pool approach (Fig. 6.2).
In this case, the number of thread is fixed a priori by the parent process.

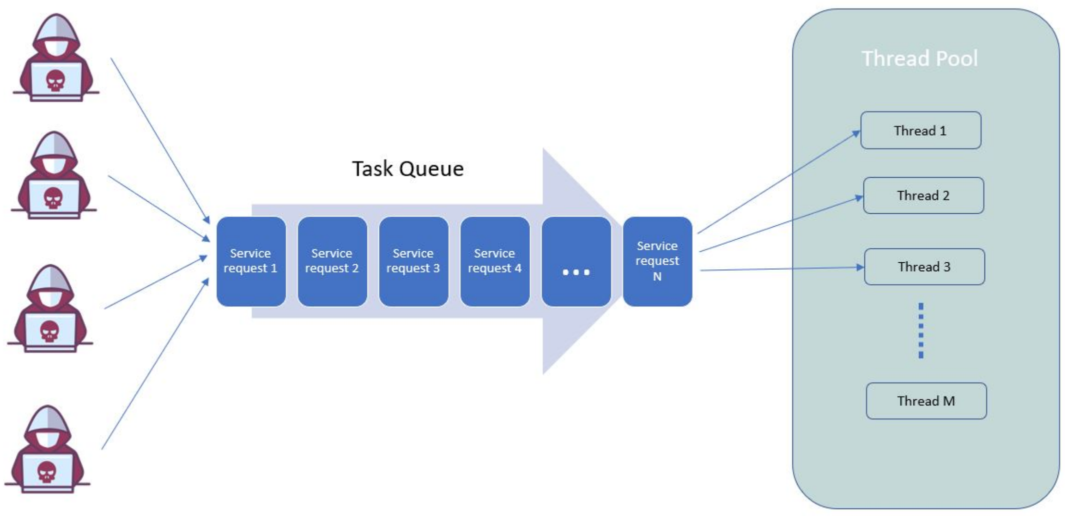


Figure 6.2: Thread pool approach

The main thread is still responsible for accepting connections, but all the requests are placed into a data structure like an array.
Then each worker threads pulls requests out of the buffer according to some scheduling algorithm.
By maintaining a pool of threads, the system increases performance and avoids latency in execution due to frequent creation and destruction of threads for short-lived tasks.
The responder module can be improved increasing the number of installed honeypot units based on the data analysis. In particular, a more sophisticated honeypot to handle the HTTP requests could catch more suspicious users, such as Glastopf (web application honeypot founded by Lukas Rist) or its successor Snare/Tanner.
Moreover, finding a better classification system may improve the selection of the most suitable honeypot when a request comes. I didn't choose the already described nDpi system because it must be able to analyze the traffic in real-time. Bringing the proxy

features to the Honeypot projects represents a significant step forward in security research because it can solve the problem of the listening port selection during the Honeypot installation.
These results highlight the fact that the HoneyPort is a promising proof of concept that, through further improvement, could represent a new frontier in Honeypots development.
The HoneyPort system is now available as an open-source project on Github.com to letthe computer science community supports software development.

Bibliography

[1] Cisco Systems, Inc. Global_2020_forecast_highlights, 2016.

[2] Stephan Haller, Stamatis Karnouskos, and Christoph Schroth. The Internet of Things in an Enterprise Context. In John Domingue, Dieter Fensel, and Paolo Traverso, editors, *Future Internet FIS 2008*, volume 5468, pages 14–28. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[3] Dave Evans. How the Next Evolution of the Internet Is Changing Everything. *CISCO white paper*, page 11, 2011.

[4] O. Arias, J. Wurm, K. Hoang, and Y. Jin. Privacy and Security in Internet of Things and Wearable Devices. *IEEE Transactions on Multi-Scale Computing Systems*, 1(2):99–109, April 2015.

[5] Ping Wang, Lei Wu, Ryan Cunningham, and Cliff C. Zou. Honeypot detection in advanced botnet attacks. *International Journal of Information and Computer Security*, 4(1):30, 2010.

[6] Biagio Botticelli 1212666. IoT Honeypots: State of the Art. September 2017.

[7] Claude Fachkha and Mourad Debbabi. Darknet as a Source of Cyber Intelligence: Survey, Taxonomy, and Characterization. *IEEE Communications Surveys & Tutorials*, 18(2):1197–1227, 2016.

[8] Francesca Soro, Idilio Drago, Martino Trevisan, Marco Mellia, Joao Ceron, and Jose J Santanna. Are Darknets All The Same? On Darknet Visibility for Security Monitoring, 2018.

[9] Eray Balkanli and A. Nur Zincir-Heywood. On the analysis of backscatter traffic. In *39th Annual IEEE Conference on Local Computer Networks Workshops*, pages 671–678, Edmonton, AB, Canada, September 2014. IEEE.

[10] F Pouget and M Dacier. Honeypot-based Forensics. page 15, 2004.

[11] Christopher Hecker, Kara L Nance, and Brian Hay. Dynamic Honeypot Construction. *University College*, page 8, 2006.

[12] Yaser Alosefer and Omer Rana. Honeyware: A Web-Based Low Interaction Client Honeypot. In *2010 Third International Conference on Software Testing, Verification, and Validation Workshops*, pages 410–417, Paris, France, April 2010. IEEE.

[13] Tatu Ylonen and Chris Lonvick. The Secure Shell (SSH) Protocol Architecture.

[14] Jacob Wurm, Khoa Hoang, Orlando Arias, Ahmad-Reza Sadeghi, and Yier Jin. Security analysis on consumer and industrial IoT devices. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 519–524, Macau, January 2016. IEEE.

[15] Gordon Fyodor Lyon. *Nmap network scanning: official Nmap project guide to network discovery and security scanning*. Insecure.Com, LLC, Sunnyvale, CA, 1st ed edition, 2008. OCLC: ocn297178138.

[16] HDFS Architecture Guide.

[17] Abdul Ghaffar Shoro and Tariq Rahim Soomro. Big Data Analysis: Apache Spark Perspective. *Global Journal of Computer Science and Technology*, February 2015.

[18] Luca Deri, Maurizio Martinelli, Tomasz Bujlow, and Alfredo Cardigliano. nDPI: Open-source high-speed deep packet inspection. In *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 617–622, Nicosia, Cyprus, August 2014. IEEE.

[19] Tatu Ylonen and Chris Lonvick. The Secure Shell (SSH) Transport Layer Protocol. 2006.

[20] J. L. Marill, A. Boyko, M. Ashenfelder, and L. Graham. Tools and techniques for harvesting the world wide web. In *Proceedings of the 2004 joint ACM/IEEE conference on Digital libraries - JCDL '04*, page 403, Tuscon, AZ, USA, 2004. ACM Press.

[21] EventTracker KB –Port No 60001 Service Name Backdoor.Linux.Trinity RFC Doc 0 Protocol TCP.

Acknowledgement

I would first like to thank my supervisor, Prof. Marco Mellia, whose expertise was invaluable in the formulating of the research topic and methodology. He let me be independent, yet gently guided me at the same time.
I would also like to thank my co-supervisor, Dr. Idilio Drago, for his valuable guidance. You provided me with the tools that I needed to choose the right direction and complete my dissertation and sometimes even reassuring words.
I am grateful to all the Postdoctoral researchers and Ph.D. students of Smart data@Polito for their wonderful collaboration. You were always willing to help me during the working hour and good buddies during break times.
I would like to thank my fantastic roommates Nicola and Enrico, who are always very patient with my disorder and help me a lot during the thesis writing.
I want to warmly thank my wonderful girlfriend Irene, who patiently and lovingly supported me in all circumstances.
I would also like to thank my closest Monregalesi friends Luca, Berru and Sara, who were always ready to help me in case of need.

And last, but by no means least, I would like to thank my family, my mother Margherita, my father Matteo, my uncle Marco and my beautiful grandmother Teresa for the great moral support that you gave to me every day of my life.

Without YOU, I would never have been able to finish my studies, so I want to dedicate this thesis to you all as a small sign of gratitude.

```
   /|
  / |                 .' '.            __           Made by
 /__|______  .        .   .           (__\_             Eros Filippi
|  __  __  |  .         .         . -{{_(|8)        s243888@polito
| |  ||  | |    ' .  . ' ' .  . '     (__/
| |__||__| |
|  __  __()|
| |  ||  | |        __        ___      __   __   __   _____
| |  ||  | |  |__| /  \ |\ | |__  \ / |__) /  \ |__)    |
| |__||__| |  |  | \__/ | \| |___  |  |    \__/ |  \    |
|__________|
```