ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Μελέτη και Εγκατάσταση Εργαλείων Εντοπισμού και Παγίδευσης Κακόβουλων Ενεργειών (Honeypots) σε Υποδομές Νεφοϋπολογιστικής (Cloud Infrastructures)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΑΙΚΑΤΕΡΙΝΗ ΣΑΡΛΟΥ

Επιβλέπων : Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Αθήνα Μάρτιος 2015

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Μελέτη και Εγκατάσταση Εργαλείων Εντοπισμού και Παγίδευσης Κακόβουλων Ενεργειών (Honeypots) σε Υποδομές Νεφοϋπολογιστικής (Cloud Infrastructures)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΑΙΚΑΤΕΡΙΝΗ ΣΑΡΛΟΥ

Επιβλέπων : Νεκτάριος Κοζύρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 23η Μαρτίου 2015

.....	.....	.....
Νεκτάριος Κοζύρης	Δημήτριος Φωτάκης	Γεώργιος Γκούμας
Καθηγητής Ε.Μ.Π	Επίκ. Καθηγητής Ε.Μ.Π	Λέκτορας Ε.Μ.Π

Αθήνα Μάρτιος 2015

.....

Αικατερίνη Σάρλου

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η παρούσα διπλωματική έχει ως στόχο την μελέτη διαφόρων ειδών εργαλείων παγίδευσης κακόβουλου λογισμικού (Honeypots), την εφαρμογή της υπάρχουσας τεχνολογίας στην υποδομή της υπηρεσίας ~okeanos (IaaS) και τέλος την εξαγωγή στατιστικών αποτελεσμάτων και συμπερασμάτων.

Αρχικά παρουσιάζονται και επεξηγούνται οι διαφορετικές κατηγορίες των Honeypots και με βάση ειδικά κριτήρια αξιολόγησης επιλέγεται το καταλληλότερο προς υλοποίηση. Στη συνέχεια, παρουσιάζονται οι τεχνικές που χρησιμοποιήθηκαν για την ορθή ρύθμιση των παραμέτρων του εργαλείου καθώς επίσης και τα συμπληρωματικά εργαλεία που χρησιμοποιήθηκαν, προκειμένου να καταστεί εφικτή η λειτουργία του εργαλείου και η εξαγωγή των αποτελεσμάτων. Τέλος, παρουσιάζονται και αξιολογούνται τα αποτελέσματα που παρήχθησαν.

Για την υλοποίηση της συγκεκριμένης διπλωματικής ελήφθησαν υπόψη οι όλο και αυξανόμενες απαιτήσεις ασφάλειας στα υπολογιστικά συστήματα σε παραγωγικά περιβάλλοντα, όπως αυτό της υπηρεσίας ~okeanos, και η αναγκαιότητα ύπαρξης πολλών και διαφορετικών μεθόδων αντιμετώπισης των εκάστοτε περιστατικών ασφαλείας λόγω της αυξανόμενης πολυπλοκότητας και συχνότητας εμφάνισης αυτών.

Για τους παραπάνω λόγους, η ολοκληρωμένη προσέγγιση που παρουσιάζεται στην παρούσα εργασία θεωρείται ότι καλύπτει τους παραπάνω στόχους και αποτελεί μία ιδανική λύση όσων αφορά τον εντοπισμό και την ερμηνεία κακόβουλων προσπαθειών επίθεσης προς τις υποδομές της υπηρεσίας ~okeanos.

Λέξεις κλειδιά

Honeypot, Honeyd, HoneyBot, ~okeanos, Nmap, TCP/IP, FTP, FTPS, ασφάλεια πληροφοριακών συστημάτων, cloud computing, cyber attacks, TLS/SSL, port scanning, web scanning, brute force attempts.

Abstract

The purpose of this diploma thesis is the examination of the different types of Honeypots, the deployment of the existing technology on ~okeanos (IaaS) platform and finally, the extraction of statistical results and conclusions about the types of cyber attacks that target ~okeanos service.

Firstly, we present and explain the different types of Honeypots and afterwards, based on specific evaluation criteria, we select the most suitable tool to implement. Then, we describe the techniques that were used for the appropriate configuration of the tool. Also, we outline the additional tools that were used for the successful operation of the tool and for the export of the statistical results. Finally, we present and evaluate the results that were produced.

The need for the implementation of the present diploma thesis was based on the increasing security requirements of the ~okeanos' production environment and the necessity of different incident handling methods due to the complexity and the frequency of the attacks that are reported from ~okeanos' incident response team (CERT).

For these reasons, we consider that this diploma thesis presents an integrated approach, which meets the above objectives as a solution for the identification and the interpretation of the malicious attempts that target ~okeanos service.

Key Words

Honeypot, Honeyd, HoneyBot, ~okeanos, cloud computing, Information Security, Nmap, TCP/IP, FTP, FTPS, cyber attacks, TLS/SSL, port scanning, web scanning, brute force attempts.

Στον Αντώνη

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τον καθηγητή μου κ. Νεκτάριο Κοζύρη για την εμπιστοσύνη που μου έδειξε, αναθέτοντας μου την παρούσα διπλωματική αλλά και για την υποστήριξή του καθ' όλη τη διάρκεια της εκπόνησης της.

Ακόμη, θα ήθελα ευχαριστήσω τον κ. Αλέξανδρο Ζαχαρή, Υπεύθυνο Ασφαλείας του ΕΔΕΤ, για την πολύτιμη καθοδήγησή του σε όλα τα στάδια εξέλιξης της υλοποίησης της διπλωματικής αλλά και για την προθυμία να επιλύσει οποιαδήποτε απορία μου παρουσιαζόταν στο μεταξύ.

Επίσης, οφείλω να ευχαριστήσω τον κ. Γεώργιο Καργιωτάκη, Διαχειριστή Δικτύου του ΕΔΕΤ, για την αμέριστη συνεισφορά του και την άμεση ανταπόκριση όσων αφορά στην επίλυση τεχνικών δυσκολιών που παρουσιάστηκαν κατά τη διάρκεια της εκπόνησης της διπλωματικής.

Ακόμη, να ευχαριστήσω τον κ. Κωνσταντίνο Τομπουλίδη, Μηχανικό Συστημάτων του ΕΔΕΤ, για τη συμβολή του σε καίρια σημεία της υλοποίησης και τον κ. Στάυρο Κρουστούρη, Προγραμματιστή Διαδικτυακών Εφαρμογών του ΕΔΕΤ, για την βοήθεια του όπου ήταν αναγκαίο.

Τέλος θα ήθελα να ευχαριστήσω τους κοντινούς μου ανθρώπους για την υπομονή και την συμπαράσταση τους όλο αυτό το διάστημα.

Contents

Chapter 1

Introduction

This chapter presents the incentives that led to the implementation of this diploma thesis. Furthermore, it outlines the overall structure in terms of its chapters and their respective contents.

1.1 Motivation

Nowadays, Information Security, a term which was not widely spread before the nineties, is becoming increasingly popular as the need for more secure communications grows.
Information Security, spans a rapidly growing domain of expertise of computer science, which is aimed, according to NIST "Glossary of Key Information Security Terms", *the protection of information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide confidentiality, integrity, and availability*. *Protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide* [1]:

- *integrity, which means guarding against improper information modification or destruction, and includes ensuring information non repudiation and authenticity;*
- *confidentiality, which means preserving authorized restrictions on access and disclosure, including means for protecting personal privacy and proprietary information*
- *availability, which means ensuring timely and reliable access to and use of information.*

Throughout the expansion of the Internet the need for secure systems is considered a vital part of their design and implementation. On a daily basis, thousands of incidents take place, intending to destabilize the foundations of telecommunication facilities. The following examples clearly illustrate the above. A very recent and severe incident was the "Anthem Data Breach" [2], resulting in the exposure of 80 million patients' medical records. According to Joseph Swedish, President and CEO of Anthem, the data stolen included client names, dates of birth, physical and email addresses, medical IDs and Social Security numbers. Another incident of equal importance, was the "Carbanak Apt: The Great Bank Robbery", which is still in progress, and led according to the Kaspersky Report [3], in the abduction of $1 billion, from more than 100 banks in 30 countries.

At this point, it should be stressed that every security incident, apart from causing severe issues regarding the functionality of computer networks, also induces detrimental economical and ethical effects to companies or individuals.

Another developing field, that indicates a security oriented way of thinking, concerns the Internet of Things (IoT) ecosystem. As the number of connected IoT devices constantly increase, security concerns are also exponentially multiplied. Moreover, the concepts of security should be redesigned [4] in order to fulfil the purposes of cloud-based infrastructures.

Last but not least, besides Internet of Things, Cloud Computing in general, demands more sophisticated security solutions, as hacking attacks become more elaborate. The European Union Agency for Network and Information Security (ENISA) in a recent report [5] indicates the importance of security reinforce of Cloud Computing Infrastructures, due to their critical services.

In particular, the following two cases could be distinguished:

- *Cloud computing services which are used by operators of critical infrastructure to support the delivery of their core services, in cases where the reliability of the underlying cloud technology is itself essential to the safe functioning of the critical service.*
- *Cloud computing services which are critical in themselves, i.e. if failing there would be a significant impact on health safety, security or economic well-being of EU citizens or the effective functioning EU governments.*

The above constituted a strong incentive in my desire to discover the challenges behind the implementation of a security mechanism, in a production Cloud Computing Environment, such as ~okeanos.

~Okeanos [6] is an Infrastructure as a Service (IaaS) platform provided by the Greek Research and Technology Network to the Greek Academic and Research Community. Okeanos enables the creation and management of virtual machines through an intuitive environment. ~Okeanos infrastructure counts thousands of active Virtual Machines, hosting numerous services of each kind. Therefore, monitoring and protection of the infrastructure is of paramount importance.

Greek Research and Technology Network, considering the need to ensure the security of its infrastructure, has proceeded to the creation of GRNET-CERT office. GRNET-CERT is the Computer Emergency Response Team of the Greek Research & Technology Network (GRNET). The activities of GRNET-CERT encompass a fairly wide area of applications in the computer security field. The main activity of the team is the effective response to security incidents involving GRENT's infrastructure. This is accomplished by acting as an intermediary between affected parties and offering, when required, technical advice leading to the resolution of the incident. Incidents are recorded, analyzed and monitored until they are considered resolved.

The purpose of this diploma thesis is the deployment of honeypots, in ~okeanos infrastructure, in order to investigate the types of attacks and how these attacks vary geographically. Furthermore, the conclusions extracted from honeypots will contribute to amelioration of the knowledge base of incident handling techniques.

Definition of honeypot

A honeypot is a decoy system configured to be intentionally vulnerable, deployed to gather information about attackers and their exploitation methods. While honeypots are not typically the target of highly sophisticated attacks, they are subject to many undefined attacks, and provide a window into the types of threats being launched against the cloud. [7]

1.2 Thesis Structure

This diploma thesis implements an integrated solution concerning an additional method for the security incident handling process within the scope of the ~okeanos service. Firstly, the ~okeanos platform was examined for open and presumably vulnerable services. Afterwards, those results were used for the appropriate configuration of the Honeypot tools. Then, the log file with the results collected by the Honeypot had to be transferred in a secure way, due to the risk of infrastructure compromise. Finally, the results were statistically processed and evaluated, offering an overview of the platform's status and its potential risks.

Throughout the process of the aforementioned implementation the main guideline was the maintenance of the security of the process and assurance of the stability of the productive environment of ~okeanos service.

The structure of the diploma thesis is presented below:

Chapter 2 outlines the theoretical background of the diploma thesis. It provides a detailed description of the ~okeanos platform, as well as the types of honeypots and explains the architecture of the honeypot implemented within the scope of this work and the reasons behind it.

Chapter 3 presents the details of the implementation of the Honeyd tool in the ~okeanos infrastructure and explains every step during the process as well as the tools and techniques used for the support of the implementation.

Chapter 4 presents the results obtained by two honeypots, namely Honeyd and HoneyBot, and explains the techniques that were used for the parsing of both their respective log files.

Chapter 5 outlines the conclusions derived through the evaluation of statistical interpretation of the obtained data and refers to future work which could lead to confrontation of specific incidents.

Chapter 2

Theoretical Background – Honeypots Analysis

This chapter provides a presentation of the ~okeanos platform, and deepens to the analysis of this specific use case in information security. Additionally, it outlines the types of Honeypots and their characteristics and evaluates their functionality using a number of certain criteria.

2.1 ~Okeanos Use Case

The goal of the ~okeanos platform is to deliver production-quality IaaS to GRNET's direct and indirect customers, i.e. IT departments of academic institutions and students/researchers respectively.

The ~okeanos service contains:

- Compute/Network Service (codename: cyclades)
- File Storage Service (codename: pithos+)
- Identity Management (codename: astakos)
- Image Registry (codename: plankton)
- Billing Service (codename: aquarium)
- Volume Storage Service (codename: archipelago)

The above are combined with a number of activities (monitoring, issue handling, helpdesk operations) to deliver the end-user experience. [8]

For the purposes of this diploma thesis, ~okeanos analysis will be focused on the security aspect, through the Computer Network Service "Cyclades". End users, during the creation of a Virtual Machine in "Cyclades", are able to assign an IPv4 to it and after its creation they have complete administrative rights on it, which means that they could install anything between a web server to some sort of malware, thereby potentially compromising the network infrastructure's security

CERT's reports for 2012-2013 show that in 2013 there has been a huge incident increase. (Figure 2.1) The overall number for incidents each period has a significant correlation with the number of activated users of ~okeanos service [9]. An additional way to gain an insight on the amounts of attacks targeting ~okeanos infrastructure, would be the deployment of a Honeypot and the statistical analysis of its log data. The different types of Honeypots were examined and evaluated against specific criteria, in order to determine the optimal choice, taking ~okeanos' unique features into account.

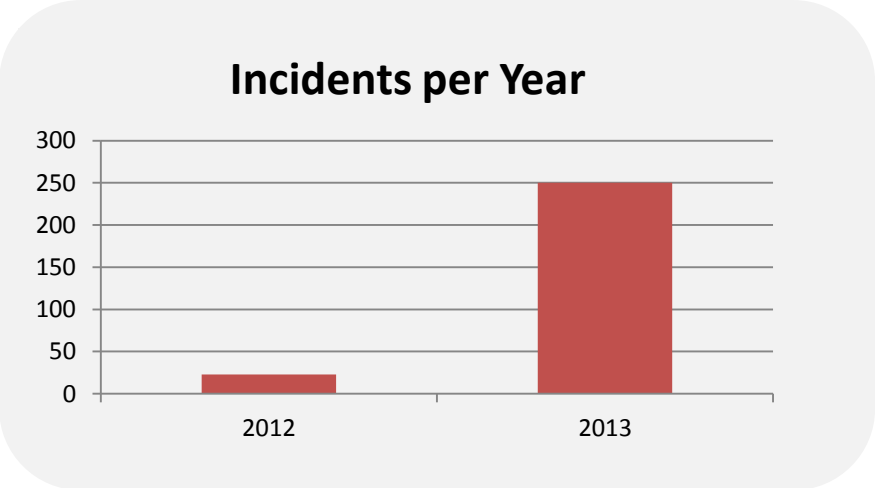

Figure 2.1: Incident increase (2012 – 2013)

2.2 Definition of Honeypot

A Honeypot tool is a security resource whose value lies in being probed, attacked, or compromised [10]. More concretely, a honeypot should be viewed as a decoy system configured to be intentionally vulnerable, deployed to gather information about attackers and their exploitation methods. While honeypots are not typically the target of highly sophisticated attacks, they are subject to many undefined attacks, and provide a window into the types of threats being launched against the cloud.

Honeypots allow researchers to:

- Collect new and emerging malware
- Identify the source of the attacks
- Determine attack vectors
- Build a profile of the target infrastructure if using specific infrastructure domains

2.3 Types of Honeypots

The different categories of honeypots are shown below (figure 2.2) and subsequently their properties are explained in detail:

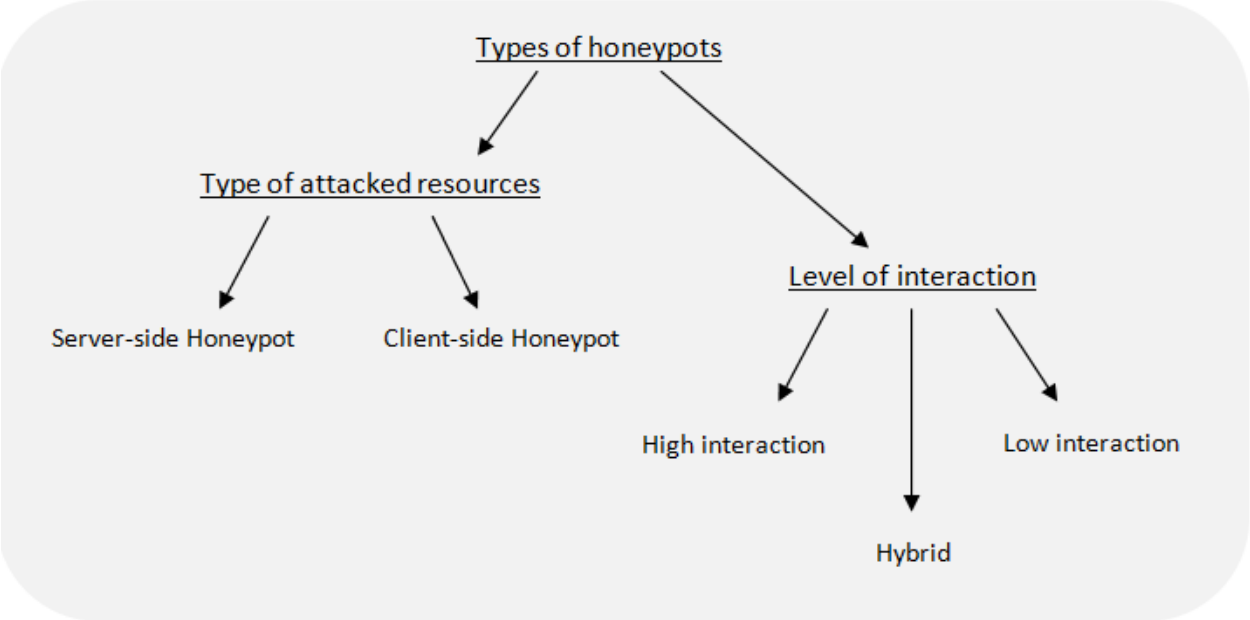

Figure 2.2: Types of honeypots

2.3.1 Server-side honeypots

Server-side honeypots are designed with the objective to detect attacks on network services. This particular type of Honeypots took its name by the fact that they emulate the behavior of a server. More specifically, server-side honeypots emulate the exposure of one or multiple open ports or whole applications and listen passively for incoming connections, established by remote (likely malicious) clients. Server-side honeypots have the ability to detect threats involving scanning methods aiming to identify potential victims to compromise. Server-side honeypots are widely used and for this reason most of the times the term Honeypot is by default associated with them.

2.3.2 Client-side honeypots

On the other hand client-side honeypots are designed to detect attacks on client applications. A client application is a software module that establishes a connection to a server and proceeds to interaction with it. Client-side honeypots function differently from server-side honeypots in that they detect malicious behavior of either the server or the content it serves by establishing active connections to services that the server hosts. The main targets of the most popular client-side honeypots are frequently-used applications such as web browsers, together with associated extensions and plugins, propagated via web pages. Moreover, client-side honeypots have the ability to examine various forms of attachments. Finally, the term that is generally used to describe this specific type of honeypots is “Honeyclients”.

2.3.3 Low interaction honeypots

The main characteristic of low-interaction honeypots is that they operate by emulating a set of resources. More precisely, server-side honeypots emulate services and honeyclients emulate client applications. Low-interaction honeypots adopt only a subset of characteristics of the resources that they try to mimic and therefore the interaction with an attacker is limited to a certain extent by the accuracy of emulation, in comparison to actual resources. The level of emulation is determined by the quality of the configuration of the low-interaction honeypot. The interaction between the attacker and the honeypot also depends on the degree of accuracy of the emulation that has been accomplished by the honeypot. In cases where the accuracy is poor, the adversary may identify the service as fake and terminate the before the actual malicious actions are been performed.

The greatest advantages of low-interaction honeypots are that their deployment is very straightforward and they are very simple to maintain. In addition, the emulation constrains the attack to a supervised environment and thus reduces the risk of the system becoming compromised. Moreover, low-interaction honeypots are very useful as the user has the ability to maintain the control over the attack and to determine its current stage. Low-interaction honeypots, however, have some disadvantages. The main disadvantage is that, in many cases, no matter how thorough a configuration was accomplished by the creators, emulated resources tend to behave in a different way than real ones. This behavior divergence could potentially allow the attacker to detect the targeted service as honeypot and end the attack

attempt prematurely. Finally, this specific type of honeypots has some inherent limitations which forbid emulating not-yet-known vulnerabilities namely, 0-day vulnerabilities.

2.3.4 High interaction honeypots

The prime difference between low-interaction honeypots and high-interaction honeypots is that the latter provide real operating systems and resources, both services and client applications, and they do not emulate them. In other words, high-interaction Honeypots are essentially actual systems with the only difference being that they are intended to serve as attack targets. High-interaction honeypots can use both bare metal or virtual environments for their purposes.

The dominant advantage of high-interaction honeypots is the fact that they exhibit realistic operating system and resources behavior, preventing the adversary from detecting the system's actual purpose and thereby minimizing the possibility of the attack ending before the malicious actions are performed. Also, they can detect the attacks on not-yet-known vulnerabilities, i.e. 0-day vulnerabilities.

Another difference between these two categories of honeypots is the quality of the collected data. High-interaction honeypots gather a great variety of data, which contain more information and details about the attack. However, the main characteristic of high-interaction honeypots, i.e. the fact that they provide real operating systems and resources, may also be considered as a drawback since their deployment and usage, including their configuration and management, requires significant effort and cost. In addition, their complexity makes it hard for the user to take control of the attack steps. This could increase the risk of compromising real systems, and losing control of the honeypot. Finally, determining which elements of system/application behavior are suspicious or malicious and which are benign is not straightforward. All disadvantages mentioned above stem from the fact that high-interaction honeypots require elaborated deployment.

2.3.5 Hybrid Honeypots

Hybrid honeypots, as their name describes, combine the characteristics of both low-interaction and high-interaction honeypots in order to be more effective. Hybrid honeypots achieve the amelioration of the threat detection performance as they make use of a high-interaction server-side honeypot to learn how to handle unknown traffic and then they redirect the traffic to low-interaction server-side honeypots to further analyze the attack.

2.4 Evaluation of different types of Honeypots

The need to evaluate the different available Honeypots, in order to choose the most appropriate to implement, according to the needs of the present diploma thesis, led to the introduction of the following criteria based on which they will be assessed:

- Cost (preferable under BSD license)
- Ease of use and setting up
- Usefulness for CERTs

Based on the above criteria and in accordance with the tables of the European Network and Information Security Agency (ENISA) [11], it was concluded that among honeypots of high and low interaction the latter category was preferable. Low interaction honeypots have the great advantage of being simpler with respect to configuration and maintenance. Additionally, the majority of low interaction honeypots are open source which makes them a low-cost solution.

Moreover, the choice of the right tool within the low interaction category was once more established based on the above criteria. Concretely, more particular importance was given the final characteristic, i.e. usefulness for CERT. One of the predominant choices was Honeyd, a tool that satisfies all three criteria, mentioned earlier. Honeyd was finally chosen because it contains a wide range of possibilities, which makes it very useful for CERTs, providing the capability of implementing scripts management and simulation of complex network topologies.

The next section elucidates the characteristics and properties of Honeyd tool, focusing on the potential adjustment of its parameters.

2.5 Honeyd

Honeyd [12] is a honeypot developed by security researcher Niels Provos and it is available as open source software, licensed under GNU General Public License v2.0, as part of Honeynet program [13]. Honeyd was ground-breaking in that it could create multiple virtual hosts on the network, as opposed to just using a single physical host. The honeypot can emulate various operating systems (which differ in how they respond to certain messages) and services. Since Honeyd emulates operating systems at the TCP/IP stack level, it can fool even sophistic network analysis tools such as Nmap. Upon attack, Honeyd can passively attempt to identify the remote host [14].

Honeyd supports the creation of a template bound with an IPv4 address, which determines the behaviour of virtual hosts and networks. The Honeyd tool, is simple to implement as all parameters are adjusted in a simple text file (text file: honeyd.conf), resulting in requiring minimal effort.

Honeyd is able to handle virtual honeypots on multiple IPv4 addresses simultaneously, in order to populate the network with numerous virtual honeypots simulating different operating systems and services. To increase the realism of the simulation, the framework is able to simulate arbitrary network topologies. To simulate address spaces that are topologically dispersed and for load sharing, the framework also needs to support network tunnelling. Honeyd is designed to reply to network packets whose destination IPv4 address belongs to one of the simulated honeypots. For Honeyd to receive the correct packets, the network needs to be configured appropriately. There are several ways to do this, e.g., creation of special routes for the virtual IPv4 addresses that point to the Honeyd host, use of Proxy ARP, or use of network tunnels.

2.5.1 Architecture of Honeyd

The Honeyd's architecture [15] is presented in Figure 2.3 and contains the above components:

- a configuration database
- a central packet dispatcher
- protocol handlers for ICMP, TCP and UDP protocols
- a personality engine
- an optional routing component

Firstly, the incoming packet is processed by the central packet dispatcher. Before it can process a packet, the dispatcher must query the configuration database to find a honeypot configuration that corresponds to the destination IPv4 address. If no specific configuration exists, a default template is used. Then the packet dispatcher examines the length of the packet and verifies its checksum. If the packet belongs to the category of the major internet protocols (ICMP, TCP and UDP) honeyd knows how to handle it. Otherwise, the packet is logged and then discarded.

Afterwards, the packet and corresponding configuration is forwarded to the protocol-specific handler. The ICMP protocol handler supports most ICMP requests. By default, all honeypot configurations respond to echo requests and process destination unreachable messages. For TCP and UDP, the framework can establish connections to arbitrary services. More specifically, Honeyd contains a simplified TCP state machine, in which the three-way handshake for connection establishment and connection teardown via FIN or RST is fully supported, but receiver and congestion window management is not fully implemented. In addition, UDP datagrams are passed directly to the application, a process that is described in the next paragraph. When the framework receives a UDP packet for a closed port, it sends an "ICMP port unreachable" message unless instructed otherwise by the configured personality. In sending "ICMP port unreachable messages", the framework allows network mapping tools like traceroute to discover the simulated network topology.

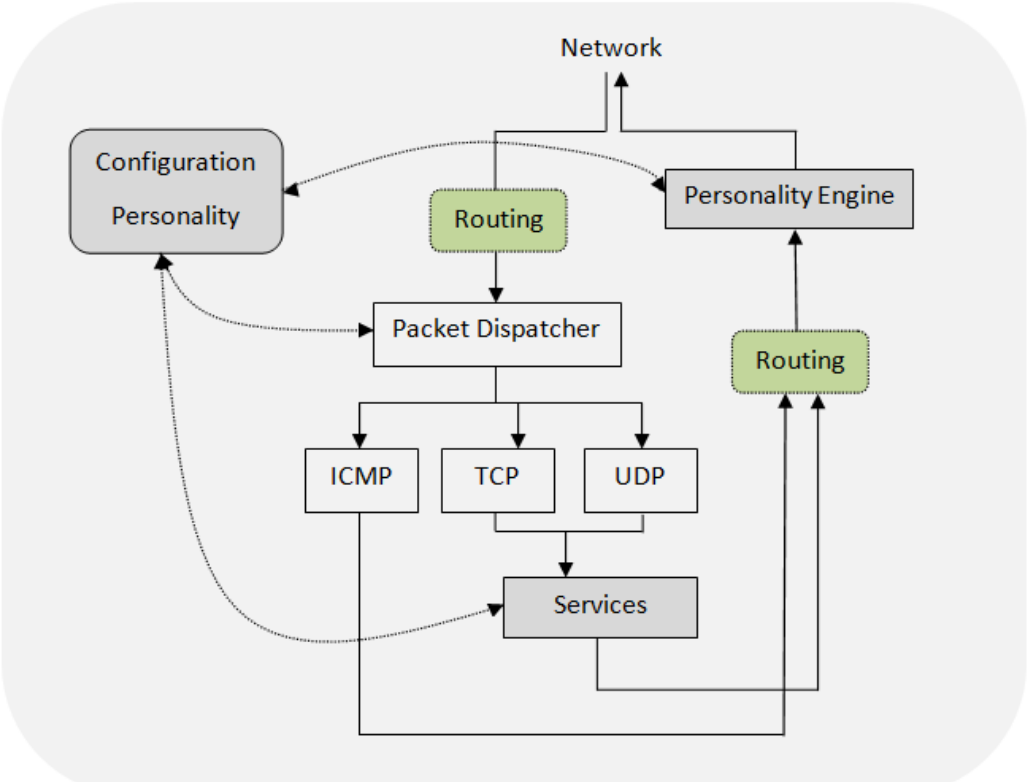

Figure 2.3: Architecture of Honeyd

As mentioned before, for each packet, Honeyd can create connections to arbitrary services. The behaviour of a service depends entirely on the external application. When a connection request is received, the framework checks if the packet is part of an established connection. In that case, any new data is sent to the already started service application. If the packet contains a connection request which is not part of an existing connection, a new process is created to run the appropriate service. Instead of creating a new process for each connection, the framework supports subsystems and internal services.

A subsystem is an application that runs in the name space of the virtual honeypot. The subsystem-specific application is started when the corresponding virtual honeypot is instantiated. A subsystem can bind to ports, accept connections, and initiate network traffic. While a subsystem runs as an external process, an internal service is a Python script that executes within Honeyd. Internal services require even less resources than subsystems but can only accept connections and not initiate them.

In every case, before a packet is sent to the network, it is processed by the personality engine. The personality engine adjusts the packet's content so that it appears to originate from the network stack of the configured operating system.

2.5.2 Honeyd's Personality Engine

Honeyd uses the term personality to refer to the network stack behaviour of a virtual honeypot. This characteristic is used when an adversary tries to scan an infrastructure with a fingerprinting tool. The main purpose of this is to gather information about a target system. The most commonly utilized fingerprinting tools are Nmap and Xprobe. It is important that honeypots do not stand out when fingerprinted and therefore Honeyd simulates the network stack behaviour of a given operating system, to make them appear real to a probe.

The personality engine makes a honeypot's network stack behave as specified by introducing changes into the protocol headers of every outgoing packet so that they match the characteristics of the configured operating system. The framework uses Nmap's fingerprint database as its reference for a personality's TCP and UCP behaviour and Xprobe's fingerprint database as a reference for a personality's ICMP behaviour. Also, Honeyd can have a different personality for each template. The user can configure Honeyd to impersonate more than 130 systems, including different versions of Windows, Mac OS, Linux, Digital UNIX, Sun Microsystems' Sun Solaris, FreeBSD, NetBSD, OpenBSD and many network infrastructure devices. For example, Honeyd can be configured to respond to a port scan where port 80 is open and also respond to an attempt to get header information from a Web server by returning a standard head request reply that includes information such as the server type. Honeyd also enables the creation custom personalities to extend its functionality.

In the next paragraph, we clarify how the information provided by Nmap's fingerprints, is used to change the characteristics of a honeypot's network stack. Each Nmap fingerprint has a format similar to the example shown in Figure 2.4 [15]:

```
Fingerprint IRIX 6.5.15m on SGI O2
TSeq(Class=TD%gcd=<104%SI=<1AE%IPID=I%TS=2HZ)
T1(DF=N%W=EF2A%ACK=S++%Flags=AS%Ops=MNWNNTNNM)
T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=N%W=EF2A%ACK=O%Flags=A%Ops=NNT)
T4(DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(DF=N%W=0%ACK=S%Flags=AR%Ops=)
PU(Resp=N)
```

Figure 2.4: An example of an Nmap fingerprint

The string after the Fingerprint word describes the personality name. The lines after the name show the different tests that Nmap performs in order to determine the operating system of a remote host. The first test determines how the network stack of the remote operating system creates the initial sequence number (ISN) for TCP SYN segments. Nmap indicates the difficulty of

predicting ISNs in the Class field. Predictable ISNs pose a security problem because they allow an adversary to spoof connections.

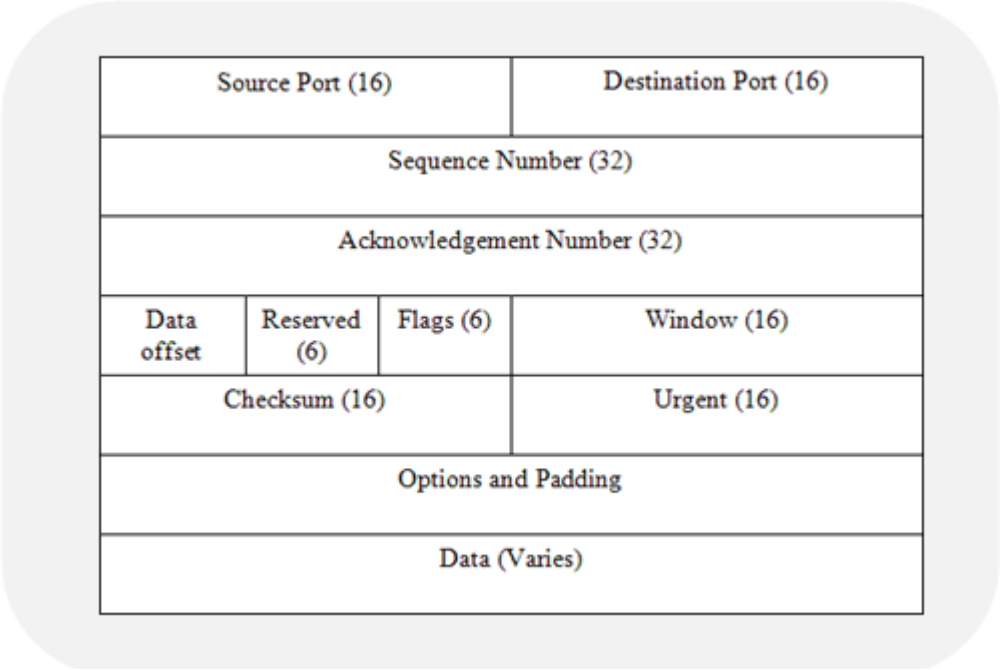

Figure 2.5: Structure of TCP heade

The meaning of each test is presented below:

- Tseq: TCP sequence ability test
- T1: SYN packet with a number of TCP options to open port
- T2: NULL packet with a number of TCP options to open port
- T3: SYN/FIN/URG/PSH packet with a number of TCP options to open port
- T4: ACK packet with a number of TCP options to open port
- T5: SYN packet with a number of TCP options to closed port
- T6: ACK packet with a number of TCP options to closed port
- T7: SYN/FIN/URG/PSH packet with a number of TCP options to closed port
- PU: UDP packet to a closed port

The framework keeps track of every virtual honeypot that is created. Every honeypot record includes information about ISN generation, the boot time of the honeypot and the current IP packet identification number. This is necessary, because the subsequent ISNs generated by the personality engine should follow the distribution specified by the fingerprint. Nmap's fingerprinting is mostly concerned with an operating system's TCP implementation.

The framework uses the fingerprint to determine the frequency with which TCP timestamps are updated. For most operating systems, the update frequency is 2 Hz. Generating the correct distribution of initial sequence numbers is not straightforward. Nmap obtains six ISN samples and analyzes their consecutive differences. Nmap recognizes several ISN generation types: constant differences, differences that are multiples of a constant, completely random difference, time dependent and random increments.

To differentiate between the latter two cases, Nmap calculates the greatest common divisor (gcd) and standard deviation for the collected differences. The framework keeps track of the last ISN that was generated by each honeypot and its generation time. For new TCP connection requests, Honeyd uses a formula that approximates the distribution described by the fingerprint's gcd and standard deviation. In this way, the generated ISNs match the generation class that Nmap expects for the particular operating system.

For the IP header, Honeyd adjusts the generation of the identification number. It can either be zero, increment by one, or random. For ICMP packets, the personality engine uses the PU test entry to determine how the quoted IP header should be modified using the associated Xprobe fingerprint for further information. As the packet leaves the Honeyd process, the personality engine modifies the content of the packet headers to mimic the desired operating system. This is the limit of interaction that the adversary has with the system since there are no actual services to interact with but only scripts. Also, some operating systems modify the incoming packet by changing fields from network to host order and as a result quote the IP and UDP header incorrectly. Honeyd introduces these errors, if necessary, into the quoted IP header to match the behaviour of network stacks. Figure 2.6 shows an example for an ICMP destination unreachable message

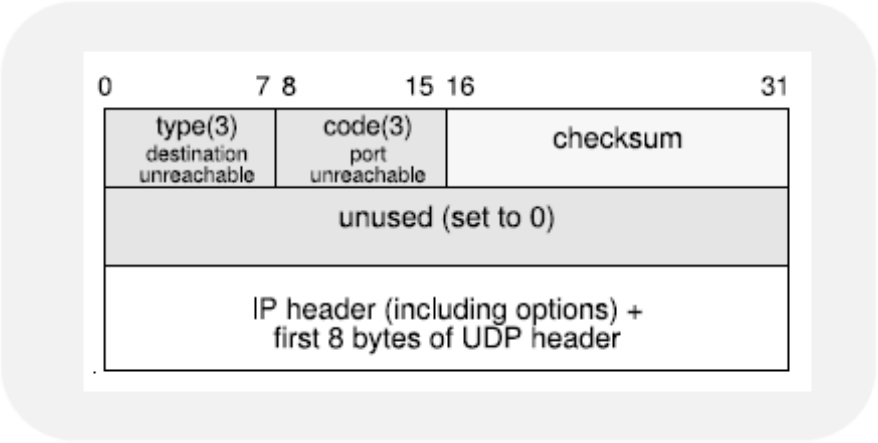

Figure 2.6: The diagram shows the structure of an ICMP port unreachable message.

2.5.3 Honeyd's counter measures

The following paragraph presents an overview of the counter measures used by Honeyd tool. [16]:

- *Faked OS personality*: When an OS is scanned it activates a personality, which can be viewed as a defensive measure. The faking of OS consumes the time of attackers when they perform scanning. This is a passive measure as there is no intent to tackle the attacker. This is static since the OS personalities once set do not change.

- *Faked network topology*: When the topology is scanned the fake network topology is activated, which can be viewed as a defensive measure. The faking of network topology consumes the time of attackers when they perform scanning. This is a passive measure as there is no intent to combat the attacker. This is static since the topology once set does not change.

- *Faked TCP/IP stack*: Honeyd activates TCP/IP handshake sequence only when scanned, which is also a defensive measure. This manipulates the TCP/IP sequence responses sent to the attacker, making it an active counter measure. This is dynamic as TCP/IP stack changes on the initiation of each operating system scan. Hence it helps in misleading the attacker of the fake operating system as active fingerprinting used by attackers will receive information of the operating system emulated by honeyd.

- *Faked services*: The faked services inform the attacker about the potential targets making it an active. Services are exposed when they are scanned. Since it informs the attacker of services and weakness that doesn't really exist, it is effective in deception.

- *Faked buffer overflow*: Fake buffer overflow is activated only when it is initiated through attacks. The attacker sees the system as penetrable and thus their time as well as effort is consumed. This is a passive measure as there is no intent to combat the attacker. Buffer overflow is triggered only when the correct attack takes place. This acts and appears as a system crashing down and forces the attacker to believe that it is exploitable.

2.6 HoneyBot

The data collected from Honeyd will be compared with the results of another honeypot, HoneyBOT [17], which is already implemented in the ~okeanos platform.

HoneyBOT is a Windows-based low-interaction honeypot solution, which emulates vulnerabilities in network services. HoneyBOT works by opening many UDP and TCP listening sockets on the honeypot machine and then emulates vulnerabilities on these ports. When an attacker connects to these services they are "tricked" into thinking they are attacking a real server. The honeypot safely captures all communications with the attacker and logs these results for future analysis.
HoneyBOT can be configured in two modes [18].

- Mode 1 honeypot: In this mode the HoneyBot listens on all TCP and UDP ports. Each connection attempt is established and timed out. In case of TCP protocol usage, when an external host sends a SYN packet to the honeypot, the honeypot replies with a SYN/ACK. Hence, it is possible to record the initial exchanged bytes.
- Mode 2 Blackhole: In this mode, the HoneyBot is completely passive and simply captures the packets, received by the network. In this case, the initial interactions cannot be recorded due to the missing established handshake.

Figure 2.7 shows a sample of HoneyBot's log file. The HoneyBot has the ability to select the export logs to CSV (Comma Separated Value) option to create a daily extract of the log file as a CSV file.

```
24,5,2014-11-01,2:15:59,758,+3:00,118.121.38.181,47975,83.212.101.19,23,TCP,RX,FF,1
25,5,2014-11-01,2:16:0,367,+3:00,118.121.38.181,47975,83.212.101.19,23,TCP,RX,FC18,2
26,5,2014-11-01,2:16:2,207,+3:00,118.121.38.181,47975,83.212.101.19,23,TCP,RX,FF,1
27,5,2014-11-01,2:16:2,831,+3:00,118.121.38.181,47975,83.212.101.19,23,TCP,RX,FC20,2
28,5,2014-11-01,2:16:4,423,+3:00,118.121.38.181,47975,83.212.101.19,23,TCP,RX,FF,1
29,5,2014-11-01,2:16:5,31,+3:00,118.121.38.181,47975,83.212.101.19,23,TCP,RX,FC23,2
30,5,2014-11-01,2:16:6,591,+3:00,118.121.38.181,47975,83.212.101.19,23,TCP,RX,FF,1
31,5,2014-11-01,2:16:7,215,+3:00,118.121.38.181,47975,83.212.101.19,23,TCP,RX,FC27,2
32,5,2014-11-01,2:16:8,978,+3:00,118.121.38.181,47975,83.212.101.19,23,TCP,RX,FIN,0
33,6,2014-11-01,2:20:54,536,+3:00,68.36.73.233,2663,83.212.101.19,8080,TCP,RX,SYN,0
```

Figure 2.7: HoneyBot's log file

The log file includes information about the date, the time and the time zone. Also, presents the remote IPv4 and Port that were used by the adversary and the local IPv4 and Port that were used by the tool. Moreover, it gives information about the size of each received packet and about the type of the connection.

Chapter 3

Honeypots Implementation - ~Okeanos Use Case

This chapter provides a detailed description of the implementation of the Honeyd tool on the ~okeanos infrastructure and explains every step during the process as well as the tools and techniques used for the support of the implementation.

3.1 Solution Architecture

The architecture that was decided to be implemented, considering Honeyd's special characteristics and ~okeanos' needs, is presented in figure 3.1:

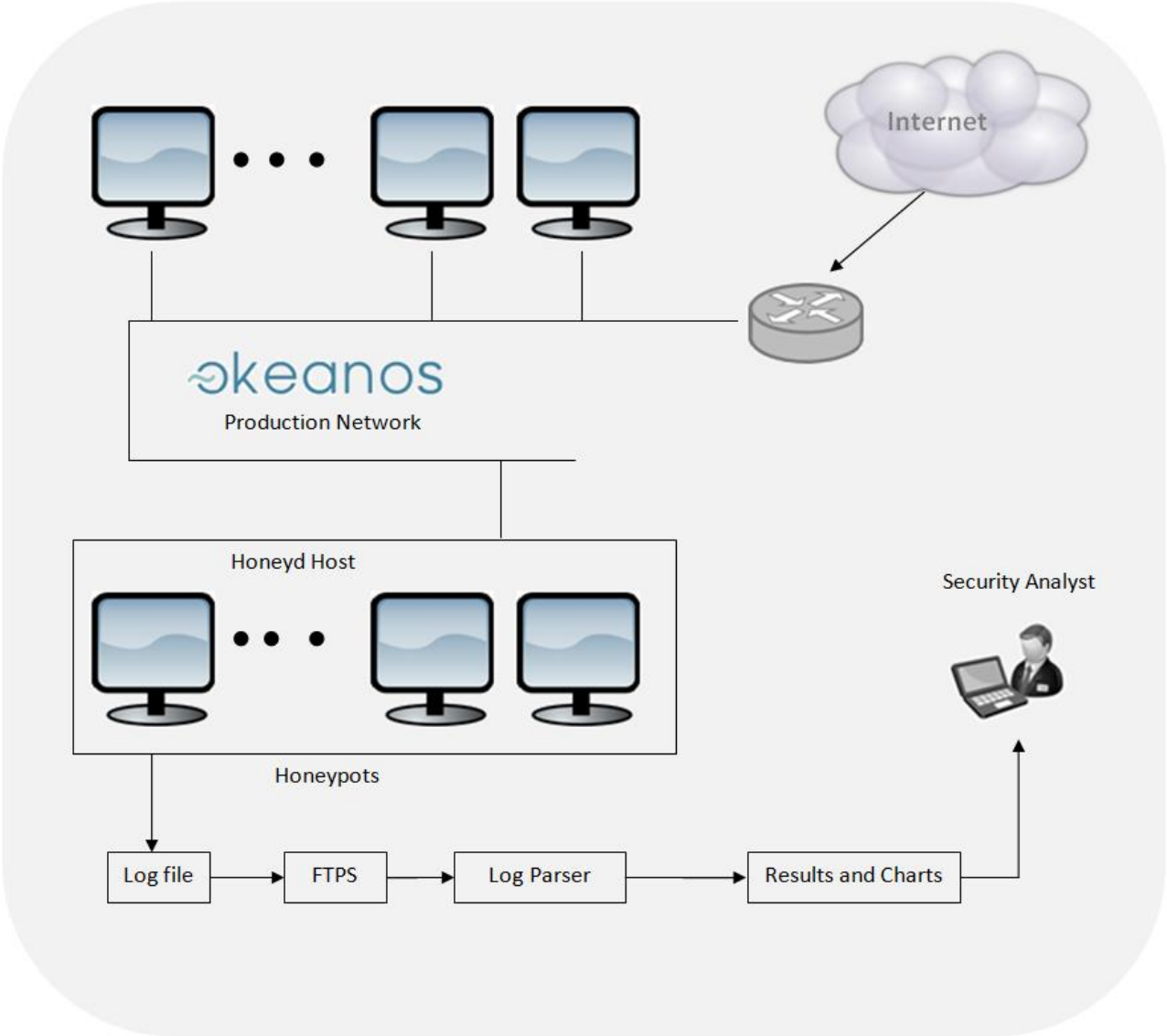

Figure 3.1: System Architecture

The main idea behind the above architecture was creating a system that gathers data about the variety of services having been installed in ~okeanos infrastructure and then using these data in order to configure the Honeyd tool appropriately in for it to "fake" them convincingly. Subsequently, Honeyd's logs are securely transferred to another Virtual Machine. There, a log parser analyses data and enables the extraction of statistical results. Finally, all information extracted from the Honeyd tool is analyzed by a security analyst so as to plan for the actions to meet the CERT demands.

3.2 Nmap ~okeanos port scanning

Nmap ("Network Mapper") is a free and open source utility for network discovery and security auditing, designed to rapidly scan large networks. Nmap uses raw IP packets in novel ways to determine what hosts are available on the network, what services (application name and version) those hosts are offering, what operating systems (and OS versions) they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics [19].

The purpose of using Nmap was to discover open ports in ~okeanos infrastructure. The results of Nmap scanning indicate the appropriate design of Honeyd's configuration. More specifically, the ports found open in ~okeanos will determine which ports and services, Honeyd should preferably emulate in order to be highly convincing in simulating a real host.

The command executed against ~okeanos' hosts is shown below (Figure 3.2):

```
nmap -T4 -A -v 83.212.96-127.0-255
```

Figure 3.2: Nmap command

The next paragraph explains the syntax of the Nmap command [19]:

- Timing template: -T

  Nmap offers a simpler approach for timing controls, with six timing templates. They can be specified with the -T option and their number (0–5) or their name. The template names are paranoid (0), sneaky (1), polite (2), normal (3), aggressive (4), and insane (5). The first two are for IDS evasion. Polite mode slows down the scan to use less bandwidth and target machine resources. Normal mode -T3, is the default mode and does not offer any extra functionality. Aggressive mode speeds scans up by making the assumption that scanned network is reasonably fast and reliable. Finally insane mode assumes that the network is extraordinarily fast or the user is willing to sacrifice some accuracy for speed. These templates allow the user to specify how aggressive they wish to be, while leaving Nmap to pick the exact timing values. The templates also make some minor speed adjustments for which fine-grained control options do not currently exist. For example, -T4 prohibits the dynamic scan delay from exceeding 10 ms for TCP ports and -T5 caps that value at 5 ms. Templates can be used in combination with fine-grained controls, and the fine-grained controls specified will take precedence over the timing template default for that parameter. Using -T4 when scanning reasonably modern and reliable networks, is highly recommended. Also, using -T4, is suggested If examination is on a decent broadband or Ethernet connection. Option -T2 is suggested rarely because the scan may take ten times longer than a default scan. Machine crashes and bandwidth problems are rare with the default timing options (-T3) and so is normally recommend that for cautious scanners. While -T0 and -T1 may be useful for avoiding IDS alerts, they will take an extraordinarily long time to scan thousands of machines or ports, so for such a long scan, is better to set manually the exact timing values you need rather than rely on the canned -T0 and -T1 values.

- Aggressive scan options: -A

  This option enables additional advanced and aggressive options. Presently this enables OS detection (-O), version scanning (-sV), script scanning (-sC) and traceroute (--traceroute). The purpose of this option is to enable a comprehensive set of scan options without people having to remember a large set of flags. However, because script scanning with the default set is considered intrusive, –A option should not be used against target networks without permission. This option only enables features, and not timing options (such as -T4) or verbosity options (-v). Options which require privileges (e.g. root access) such as OS detection and traceroute will only be enabled if those privileges are available.

- Increase verbosity level: -v

  Choosing this option increases the verbosity level, causing Nmap to print more information about the scan in progress. Open ports are shown as they are found and completion time estimates are provided when Nmap scan endures more than a few minutes. It could be used twice or more for even greater verbosity: -vv, or alternatively a verbosity level could be set directly, for example -v3. Most changes only affect interactive output, and some also affect normal and script kiddies output. The other output types are meant to be processed by machines, so Nmap can give substantial detail by default in those formats without fatiguing a human user. However, there are a few changes in other modes where output size can be reduced substantially by omitting some detail.

3.2.1 Results of Nmap port scanning

The following tables represent some general information produced by Nmap scanning tool against ~Okeanos Infrastructure that took place on 11/11/2014.Table 3.1 illustrates the total number of Ports found open, the amount of IP addresses which have been scanned as well as the number of active Hosts. The most frequently Open Ports categorized by type are presented in Table 3.2.

	Total Number
Open Ports	14258
IP addresses	8192
Hosts Up	4630

Table 3.1: Nmap scan summary

At this point, it is necessary to be mentioned that the above results concerning active hosts, could be considered a good estimation of the real situation of ~okeanos infrastructure, as shown in ~okeanos main page (https://okeanos.grnet.gr/home/). Thus, they are considered valuable information for the configuration of Honeyd tool.

The table below shows that the most commonly open port is port 22 TCP, which is used for secure logins (ssh), file transfers (scp, sftp) and port forwarding. The next most often open port is Port 80 used by Hypertext Transfer Protocol (HTTP), the foundation of data communication for the World Wide Web.

The pie chart below constitutes an alternative representation of the data outlined in Table 2.

Port Type	Open Port Number
22: Tcp	2888
80: Tcp	1685
3389: Tcp	1616
445: Tcp	798
135: Tcp	787

Table 3.2: Top 5 Open Ports in actual numbers

For the creation of the pie chart, the following statistical formulas were implemented in Python:

- Top 5 open ports:

$$Port\ Number\ (\%) = \frac{P_f}{P_t} * 100$$

- Rest ports:

$$Rest\ Port\ Number\ (\%) = \frac{P_t - P_m}{P_t} * 100$$

Where,

- P_t : represents the total number of open ports
- P_f : represents the number appearances of each open port
- P_m : represents the sum of the number of appearances of the five most frequently open ports

Figure 3.3 shows the five most frequently appeared ports among the total ports found open from Nmap scanning.

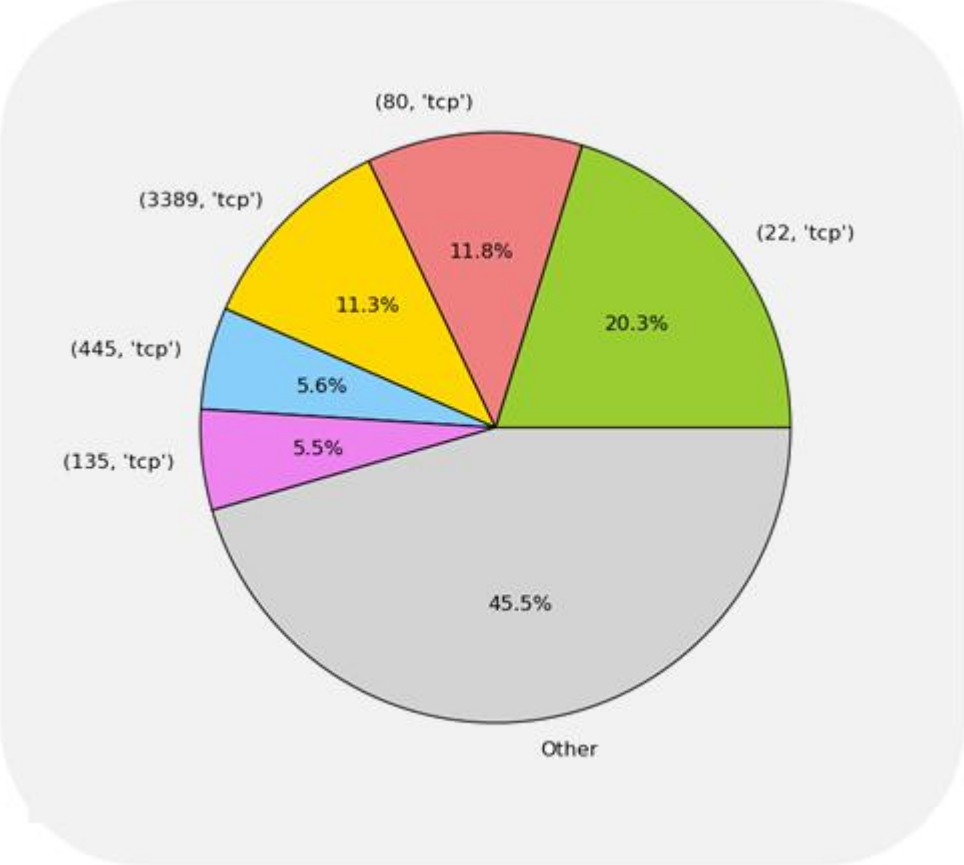

Figure 3.3: Pie chart of Open Ports

The pie chart above and the two tables as well, were created from a python script parsing Nmap outputs, written exclusively for diploma thesis purposes. The source code can be found in the appendix.

3.3 Honeyd Implementation

The honeyd host was installed on an Ubuntu 12.04 LTS virtual machine. The fake wired network and routing topology was created by writing a configuration file that honeyd used. A number of virtual systems were created. Each virtual system was shown as if it ran a particular operating system. The virtual systems were also provided with multiple scripts that ran on various ports. These imitated the services on the virtual systems and would help in deceiving the attacker. A part of the configuration file with the list of templates is presented in the Table 3.3 (the whole file can be found in the appendix).

3.3.1 Honeyd Configuration file Analysis

The configuration of Honeyd was achieved with the creation of templates inside the appropriate configuration file. The template was formed from a set of commands, explained below.

- **Create**: creates new templates.
- **Set:** assigns a personality from the Nmap fingerprint file to a template
- **Add:** specifies which services are remotely accessible and what application should run on each port
- **Bind:** assigns a template to an IP address

More specifically, the set command assigns a personality from the Nmap fingerprint file to a template. The set command also defines the default behaviour for the supported network protocols. The default behaviour is one of the following values: block, reset, or open. Block means that all packets for the specified protocol are dropped by default. Reset indicates that all ports are closed by default. Open means that they are all open by default. The latter settings make a difference only for UDP and TCP. The add command specifies the services that are remotely accessible.

In addition to the template name, Honeyd needs to specify the protocol, port and the command to execute for each service. Instead of specifying a service, Honeyd also recognizes the keyword proxy that allows forwarding network connections to a different host. The framework has the ability to expand the following four variables for both the service and the proxy statement: $ipsrc, $ipdst, $sport, and $dport. Variable expansion allows a service to adapt its behaviour depending on the particular network connection it is handling. It is also possible to redirect network probes back to the host that is doing the probing.

The bind command assigns a template to an IP address. If no template is assigned to an IP address, the default template is activated. Honeyd also enables the ability of routing topology creation, with the command route. However, this ability lies beyond the scope of the present diploma thesis mainly due to complexity of ~okeanos infrastructure concerning network configuration.

```
### Default Template
create default
# Set default behavior

set default default tcp action block
set default default udp action block
set default default icmp action  block

#### Linux Suse 8.0 template
create suse80
set suse80 personality "Linux 2.4.7 (X86)"
set suse80 default tcp action reset
set suse80 default udp action  block
set suse80 default icmp action open
set suse80 uptime 79239
set suse80 droprate in 4
add suse80 tcp  port 21  "sh scripts/unix/linux/suse8.0/proftpd.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp  port 22  "sh scripts/unix/linux/suse8.0/ssh.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp  port 23  "sh scripts/unix/linux/suse8.0/telnetd.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp  port 79  "sh scripts/unix/linux/suse8.0/fingerd.sh $ipsrc $sport $ipdst  $dport"
add suse80 tcp  port 80  "sh scripts/unix/linux/suse8.0/apache.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp  port 110  "sh scripts/unix/linux/suse8.0/qpop.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp  port 143  "sh scripts/unix/linux/suse8.0/cyrus-imapd.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp  port 515  "sh scripts/unix/linux/suse8.0/lpd.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp  port 3128  "sh scripts/unix/linux/suse8.0/squid.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp  port 8080  "sh scripts/unix/linux/suse8.0/squid.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp  port 8081  "sh scripts/unix/linux/suse8.0/squid.sh $ipsrc $sport $ipdst $dport"
add suse80 udp port 514 "sh scripts/unix/linux/suse8.0/syslogd.sh $ipsrc $sport $ipdst  $dport"
set suse80 ethernet "aa:36:29:2e:2e:2a"
bind 83.212.86.185  suse80
```

Table 3.3: Honeyd configuration file (sample)

3.3.2 ~Okeanos configuration

This section presents and explains the need for the configuration of ~okeanos platform, after the configuration of Honeyd was completed. ~Okeanos platform implements in a different way the routing rules of the packets that each virtual machine receives. Each virtual machine is assigned with a unique MAC address. However this address does not appear at each packet received from ~okeanos as the source address of the VM. Instead, the MAC address of the virtual machine is connected with the MAC address of its host machine via APP Proxy protocol, i.e. one host answers ARP requests intended for another machine. Each host uses an ARP Proxy in order to send a packet from a specific virtual machine on the Internet and respectively to receive a packet from the Internet about a specific virtual machine. Figure 3.4 gives more details about this specific implementation.

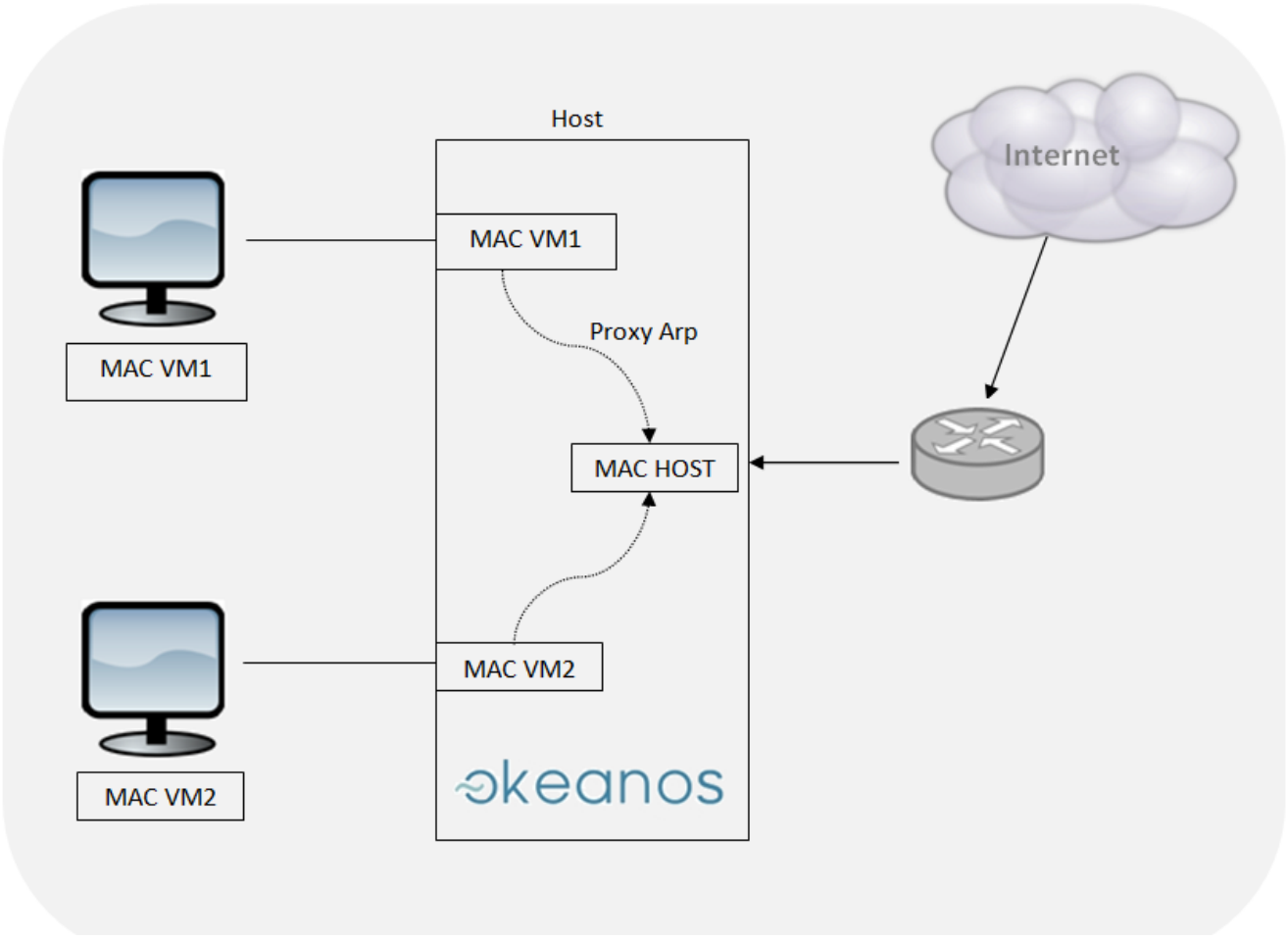

Figure 3.4: Illustration of ARP Proxy in ~Okeanos Infrastructure

Honeyd creates virtual hosts and produces a random MAC address for each one of them. Thus, without the appropriate configuration of ~okeanos, the Honeyd will fail to run properly. For the purpose of the present diploma thesis, the ~okeanos account that was used had several IPv4 addresses from the range 83.212.84.0/22 assigned to it. The virtual machine that hosted the

Honeyd tool used one IP and the rest of them were used for the virtual hosts created by the tool. The default file of Honeyd (/etc/default/honeyd) was configured as shown below:

- INTERFACE = eth1
- NETWORK = 83.212.84.0/22

Afterwards, the remaining IPv4 were used for the configuration of the honeyd via bind command. Figure 3.4 and figure 3.5 present the commands that were executed throughout the process of ~okeanos' configuration.

```
ip route add 83.212.86.XXX dev tapYY table ZZ
```

Figure 3.4: Network configuration

Where,

- tapYY = tap device established in the host referring to the VM interface

- tableZZ = routing table which enables the routing of honeyds' network from the side of the host. E.g. iptables -I FORWARD -i tapYY -s 83.212.86.XXX -j ACCEPT

```
iptables -I FORWARD -i tapYY -s 83.212.86.XXX -j ACCEPT
```

Figure 3.5: Network configuration

This configuration enables ~okeanos to process the packets send/received from Honeyd's virtual hosts without permitting its firewall rules to block them.

3.3.3 Honeyd Running command

After, the appropriate changes were made in the ~okeanos network infrastructure, Honeyd was able to run by executing the following command (Figure 3.4). The description of each attribute of Honeyd running command is given in Table 3.5.

```
/usr/bin/honeyd -d -f /etc/honeypot/honeyd.conf -l /var/log/honeypot/honeyd.log -p
/etc/honeypot/nmap.prints -a /etc/honeypot/nmap.assoc -0 /etc/honeypot/pf.os -x
/etc/honeypot/xprobe2.conf -u 0 -g 0 --disable-webserver -i eth1 83.212.86.0/24
```

Figure 3.5: Honeyd running command

It should be mentioned that Honeyd reads the data stored in the configuration file that the user chooses with the option –f. Also, the user can determine the directory, where log file data can be stored with the –l option. Option –0 enables Honeyd to read the database for passive fingerprinting. The names of the operating systems specified in file p0f-file are recognized by Honeyd's parser and can be used for dynamic templates. With –x Honeyd reads xprobe style fingerprints. This file determines how honeyd reacts to ICMP fingerprinting tools. The remaining options are clarified in Table 4.

Option	Description
-d	Enable verbose debugging messages
-f file	Read the configuration in file
-l logfile	Log packets and connections to the logfile specified by logfile
-a assoc	Read the file that associates nmap style with xprobe style fingerprints
-0 p0f-file	Read the database for passive fingerprinting
-x xprobe	Read xprobe style fingerprints
-u uid	Set the UID that Honeyd is running as
-g gid	Set the GID that Honeyd is running as
--disable-webserver	Disables the builtin webserver
-i interface	Listen on interface

Table 3.4: Honeyd command options

3.4 FTPS Deployment

The **File Transfer Protocol** (**FTP**) is a standard network protocol used to transfer computer files from one host to another host over a TCP-based network, such as the Internet. The objectives of FTP are 1) to promote sharing of files (computer programs and/or data), 2) to encourage indirect or implicit (via programs) use of remote computers, 3) to shield a user from variations in file storage systems among hosts, and 4) to transfer data reliably and efficiently. [20]

FTP is built on client-server architecture and uses separate control and data connections between the client and the server. FTP users may authenticate themselves using a clear-text sign-in protocol, normally in the form of a username and password, but can connect anonymously if the server is configured to allow it. For secure transmission that protects the username and password, and encrypts the content, FTP is often secured with SSL/TLS (FTPS).

The security extensions to FTP offer a comprehensive set of commands and responses that can be used to add authentication, integrity, and confidentiality to the FTP protocol. The TLS protocol is a popular (due to its wholesale adoption in the HTTP environment) mechanism for generally securing a socket connection. [21]

SSL/TLS offers some of the following positive attributes:

- Flexible security levels. SSL/TLS can support confidentiality, integrity, authentication, or some combination of all of these. During a session, this allows clients and servers to dynamically decide on the level of security required for a particular data transfer.

- Ability to provide strong authentication of the FTP server.

- Formalized public key management. By use of well established client identity mechanisms (supported by TLS) during the authentication phase, certificate management may be built into a central function.

- Co-existence and interoperation with authentication mechanisms that are already in place for the HTTPS protocol. This allows web browsers to incorporate secure file transfer using the same infrastructure that has been set up to allow secure web browsing.

The choice of implementing FTPS for the transfer of Honeyd Log file was based on the idea of securing ~okeanos infrastructure, in case of a compromise of the Virtual Machine hosting Honeyd.

The steps for FTPS Server deployment in a virtual machine are presented below. Additionally, the implementation of FTP client which was installed in the Honeyd machine is explained.

FTPS Server Side

The FTP daemon, chosen to be installed in ~okeanos virtual machine was ***vsftpd***. Afterwards, the configuration file was updated with the following lines.

Step 1: Basic Setup

- Anonymous_enable = NO
- Local_enable = YES
- Write_ enable = YES

The first security measure was taken to restrict user administrative privileges and "jail" them in a specific directory. Constraining the users to a certain disk space reassures that a potential adversary will not have the ability to overwrite important system data. The following changes were also made to the configuration file.

Step 2: Chroot users

- Chroot_local_users = YES
- Chroot_list_ enable = NO

The second security measure was the activation of TLS/SSL option. At this point, it should be mentioned that vsftpd uses the certificate Ubuntu creates upon its installation the "snake-oil" certificate.

Step 3: TLS/SSL/FTPS

- ssl_enable=YES
- allow_anon_ssl=NO
- force_local_data_ssl=YES
- force_local_logins_ssl=YES
- ssl_tlsv1=YES
- ssl_sslv2=YES
- ssl_sslv3=YES

*Note: the configuration file of **vsftpd** can be found in the appendix.*

The next step for securing the FTPS connection is that users are permitted to connect only if they possess a username and a password. A virtual user was created to fulfill this purpose. The followed actions are described below.

Step 4: Create a new group of FTP access for FTP users

- sudo groupadd ftpaccess

Step 5: Change the config file /etc/ssh/sshd_config, replacing the "Subsystem sftp /usr/lib/openssh/sftp-server" line with the following:

- Subsystem sftp internal_sftp
- Match group ftp access
- Chroot Directory %h
- X11 Forwarding no
- Allow TcpForwarding no
- Force Command internal_sftp

Also the next line should be commented

- #UsePAM yes

The final stage is the creation of a user with a username and a password that will have access to the directory "/home/HoneyLogs/www".

Step 6: Create user

- sudo useradd –m HoneyLogs –g ftpaccess –s /usr/bin/nologin
- sudo passwd **********
- sudo chow root /home/HoneyLogs
- sudo mkdir /home/HoneyLogs/www
- sudo chown HoneyLogs : ftpaccesss /home/HoneyLogs/www

FTPS Client Side

Once the deployment of FTPS Server was completed, the client side of the FTPS connection was implemented. The tool that was selected for this purpose was FileZilla, an open source software distributed free of charge under the terms of the GNU General Public License [22].

FileZilla was chosen because it is easy to use, with an ergonomic graphical user interface and completely compatible with FTPS connections.

The functionality of ftp client was controlled by a few settings made at FileZilla menu. The FileZilla was installed in Honeyd machine. The host attribute was completed by the Ipv4 of the virtual machine hosting FTPS server. The protocol chosen was the SFTP- SSH File Transfer Protocol. The specific directory was chosen from FileZilla directory menu and the username and the password were supplemented in the appropriate field. (Figure 3.6)

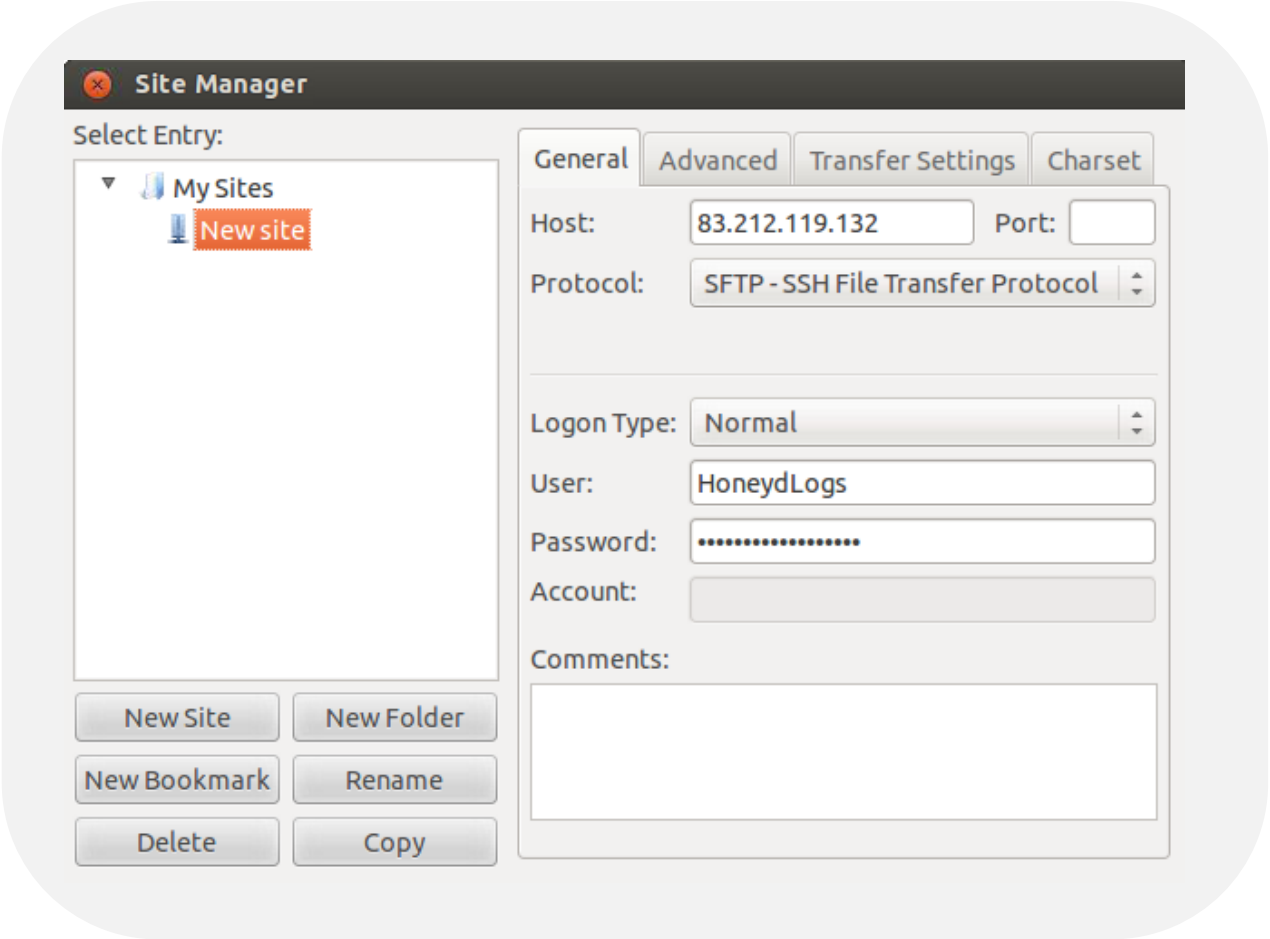

Figure 3.6: FTPS client -Connection setup

After the completion of the FileZilla setup, the transmission of the log file from Honeyd Virtual Machine to FTPS Server Machine was feasible. The file log.out was selected from directory menu and was sent to the selected directory of FTPS server. Figure 3.7 shows that the transmission was successful.

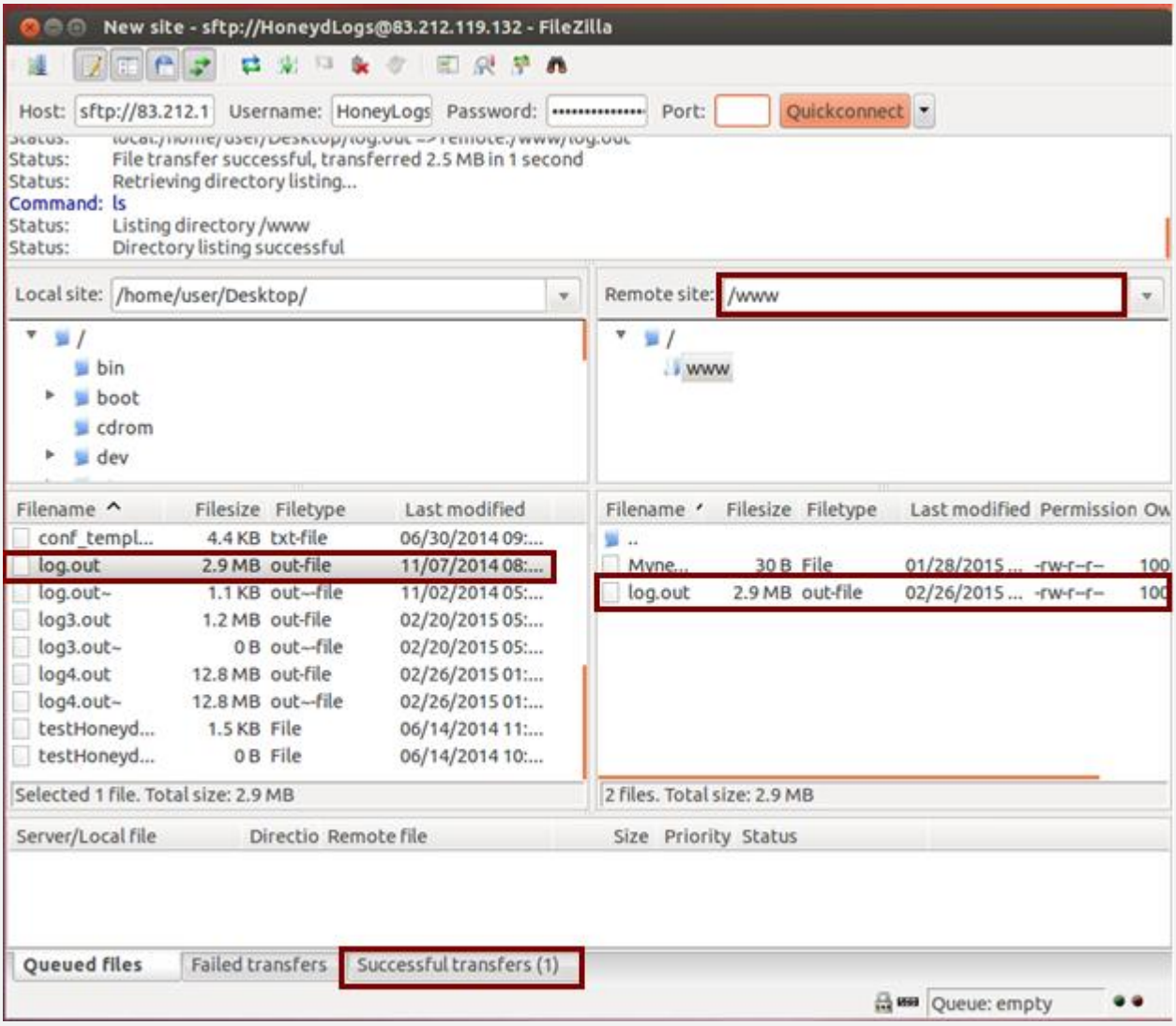

Figure 3.7: Transmission process through FTPS

Figure 3.8 illustrates that a user with restricted administrative privileges, cannot have access to any directory other than the one he is “jailed” into. The system prohibits the file transfer, as it knows that the user does not have the privileges to write in this area.

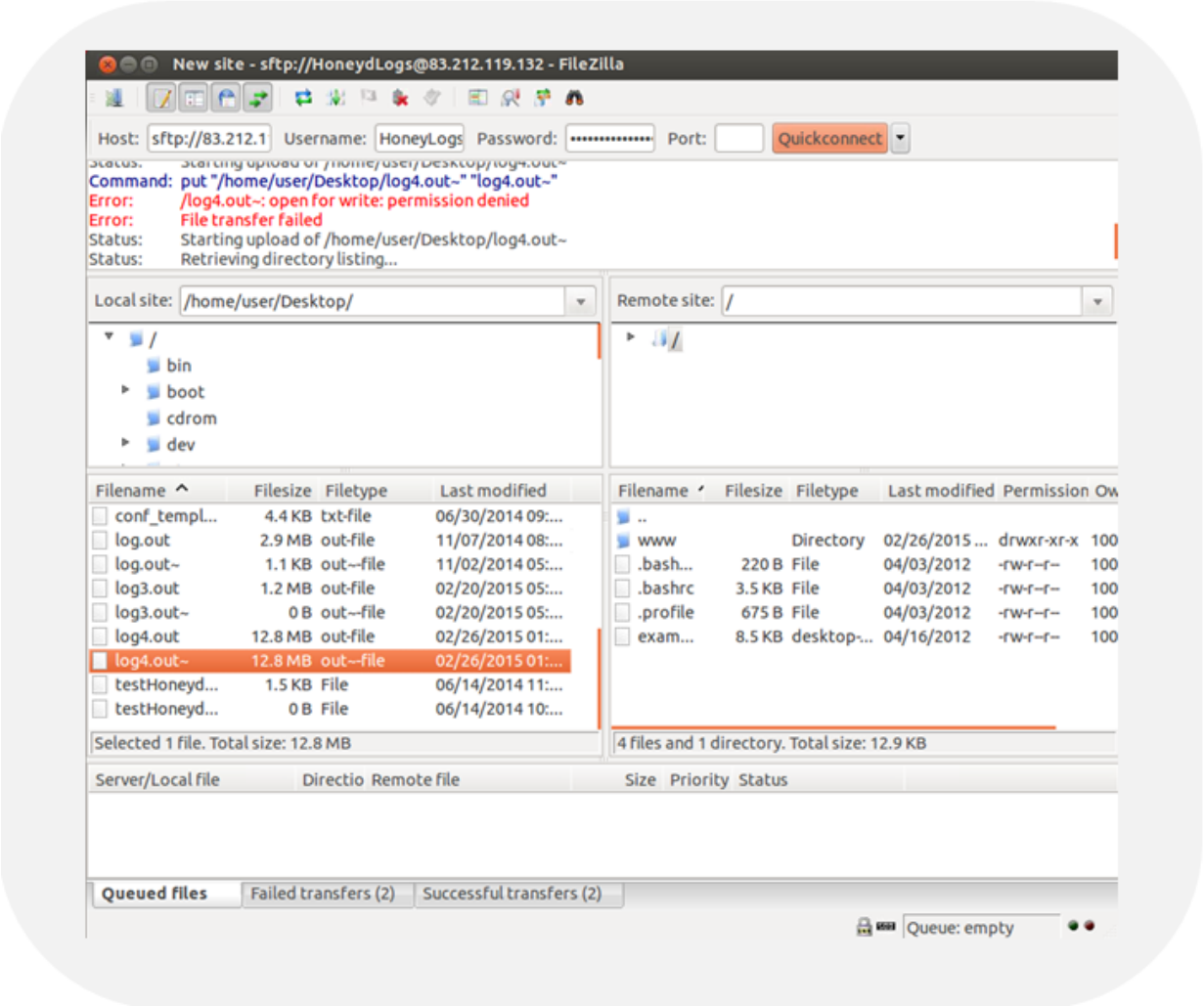

Figure 3.8: Example of an unsuccessful transmission due to restricted administrative privileges

3.5 Analysis of Log file

In this section, the analysis of the results taken from Honeyd's log file is described. The original log file created by honeyd contains the records in the format shown in figure 3.9.

```
2014-11-06-13:27:11.8123 honeyd log started ------

2014-11-06-13:27:21.4979 icmp(1) - 83.212.84.72 10.0.0.1: 8(0): 84

2014-11-06-13:52:10.7885 tcp(6) S 192.168.0.2 42532 10.0.0.2 5900

2015-02-20-00:54:38.0262 tcp(6) - 116.231.232.184 60601 83.212.86.184 3389: 48 S [Windows XP SP1]
```

Figure 3.9: Honeyd Record Format

The structure of the records is described in Table 3.5. The First field is the timestamp of the packet and the second shows the protocol type along with the number. The next field identifies whether the packet is starting the connection (S), is intermediate (-) or ending the connection (E). The next two fields store the IPv4 address and port number where the packet originated, i.e. the source IPv4 and the source port. In addition, the next field holds the IPv4 address and the port number of the honeypot where the packet was headed to, i.e. the destination IPv4 and the destination port. Next field provides the size of the packet. Then, in the case of a TCP packet, the flag (SYN, ACK, FIN, RST etc.) that appears in the header of the packet is recorded after the packet size. The last field provides information regarding the operating system of the source machine.

Description	Example
Date	2015-02-20-0054:38.0262
Protocol	tcp(6)- Icmp(1)-udp(17)
T	S / - / E
Source IP	116.231.212.184
Source Port	60601
Destination IP	83.212.86.184
Destination Port	3389
Info	48 packet size S Tcp flag
Comment	[Windows XP SP1]

Table 3.5: Description of record contents

The information that is stored in each record of the log file generated by honeyd should be parsed with an analysis tool. The following paragraph describes, briefly, the algorithms that were implemented for the statistical processing of the log files which were created by honeyd. The source code can be found in the appendix.

The required information for extracting statistical results is the Source IPv4 and the Destination Port. The destination port is needed for the determination of the attack type. The field of the Source IPv4 is equally essential, since it contains the information about the country where the attack originated from.

The algorithm takes as an input each line of the log file and then it calculates the total number of the unique ports and IPs found in the file. Afterwards, it identifies the country of origin of each IPv4 address, by using the *geolite2* python library.

The algorithm that was implemented for the statistical interpretation of HoneyBot's log files was based on the same principles that were described above.

Chapter 4

Representation – Analysis – Comparison of Honeyd and HoneyBot Statistical Results

This chapter presents the results obtained by two honeypots, namely Honeyd and HoneyBot, and explains the techniques that were used for the parsing of their respective log files.

4.1 Honeyd Results

This section presents the results obtained from Honeyd. More specifically, it shows the number and the type of ports which were attacked in ~okeanos platform. In addition, the log file provides the information about source IPv4 addresses and consequently, the information about the country where the attack originated from.

Figure 4.1 shows the number of ports that were abused in a period of a half month.

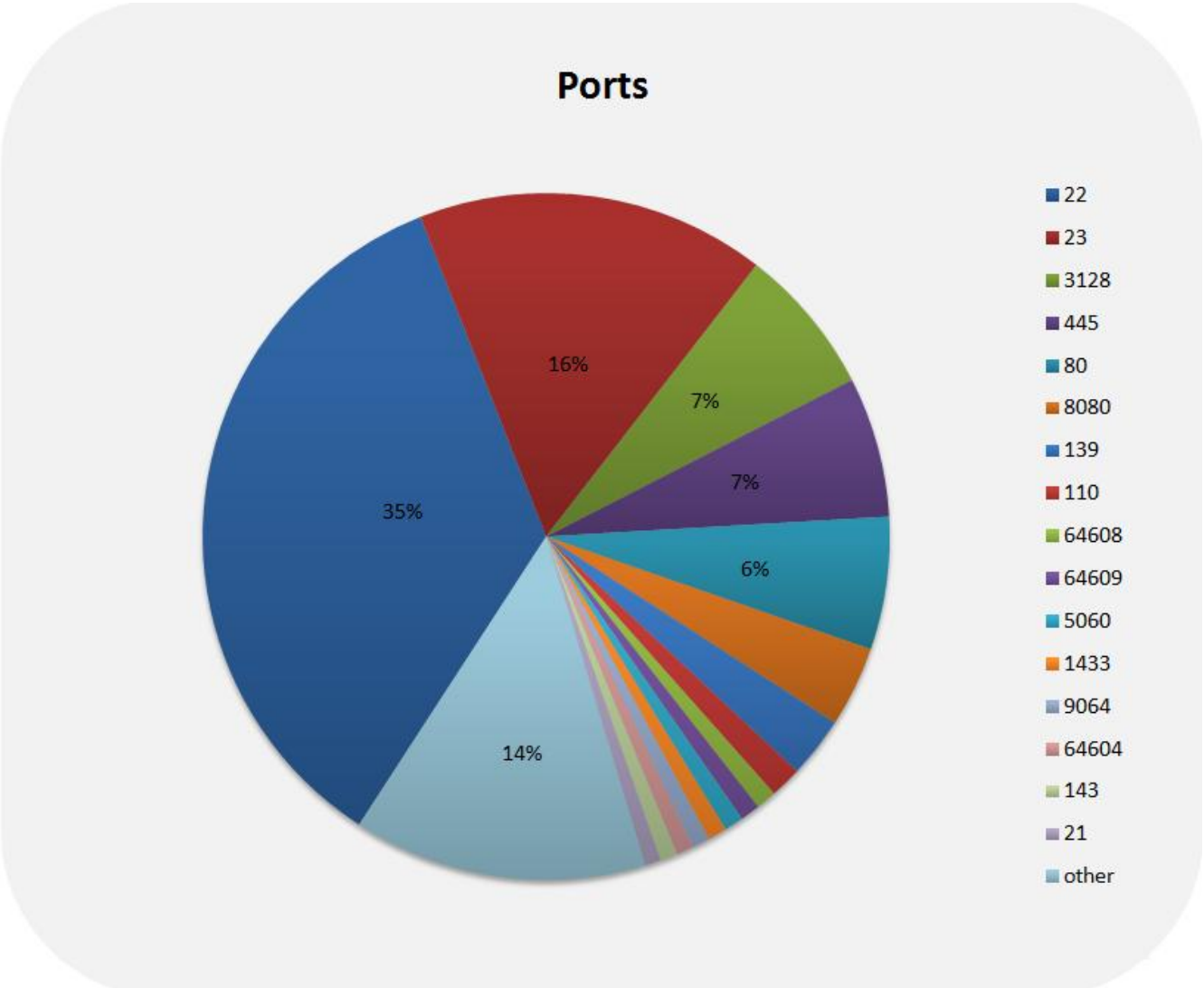

Figure 4.1: Most frequently attacked ports – Honeyd log file

From the information given in the above chart, it can be easily concluded that the majority of the attacks were on ports 22 and 23. These ports represent SSH and TELNET services respectively and this chart outlines that over 50% of the attack volume was targeting these two ports.

The type of the attack, *BRUTE FORCE* attempt, can be identified from the type of service (SSH, TELNET). In addition, a large number of attack attempts were found on port 3128, which hosts Web caches (Squid) service. Scans on port 3128 usually look for badly configured proxy servers in order to use them to hide further intrusion attempts or to bypass company (or country wide) firewall rules restricting access to certain web sites. These scans usually come in sets that scan several ports frequently used by proxies (80, 8080 etc.) Port 3128 is usually used by 'squid', a very popular web proxy server that is also able to proxy other protocols (e.g. ftp). Another target was port 445 for Microsoft-DS SMB file sharing service and port 110 Post Office Protocol v3 (POP3), third version of a widespread method of receiving email. Finally, another observation was the attacks on WEB ports like 80 and 8080 amounting to about10% of the total attack attempts. The type of the aforementioned attacks, were PORT and WEB SCANNING.

By analysing the origin of the attack traffic, 132 countries were identified, attacking or crawling/scanning the Honeypot. The country with the greatest attack rate was China (CN) as 24% of all attack traffic originated from there. The second country was the United States (US) with a rate of 18%. Other high rate attacking countries were South Korea (KR), Brazil (BR), Russian Federation (RU), France (FR), Taiwan (TW) and Turkey (TR) (Figure 4.2). The map below visualizes the above results, for a more comprehensive representation. (Figure 4.3)

It should be mentioned that from the total amount of countries that were examined, only the first fifty had a number of attacks greater than 10. Therefore, the percentages may not seem that great, but that is only because the numbers of countries with a small rate of attacks are quite a lot.

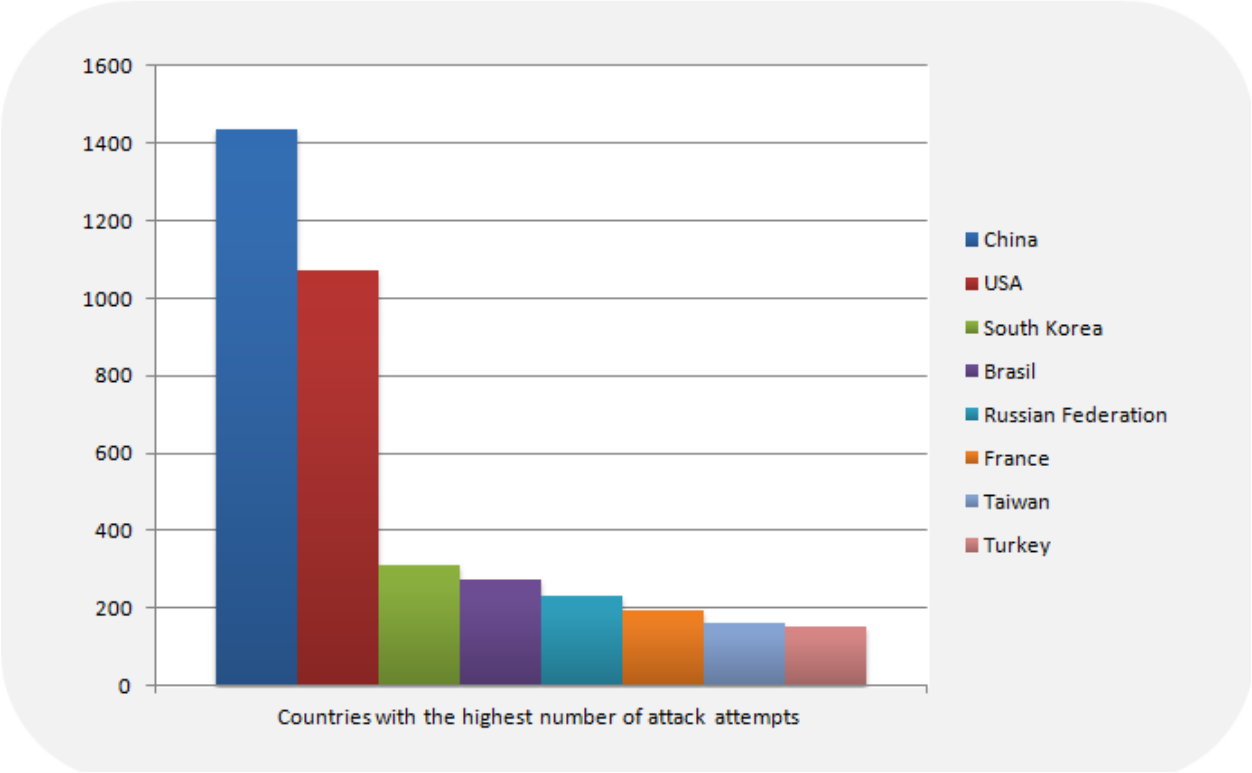

Figure 4.2: Countries with the highest number of attack attempts – Honeyd log file

Figure 4.3: Visualization of Attacks per Country

4.2 HoneyBot Results

This section presents the results obtained from HoneyBot. More specifically, it shows the number and the type of ports which were attacked in ~okeanos platform. In addition, the log file provides the information about source IPv4 addresses and consequently, the information about the country where the attack originated from.

Figure 4.4 presents the number of ports that were been attacked in a period of two months.

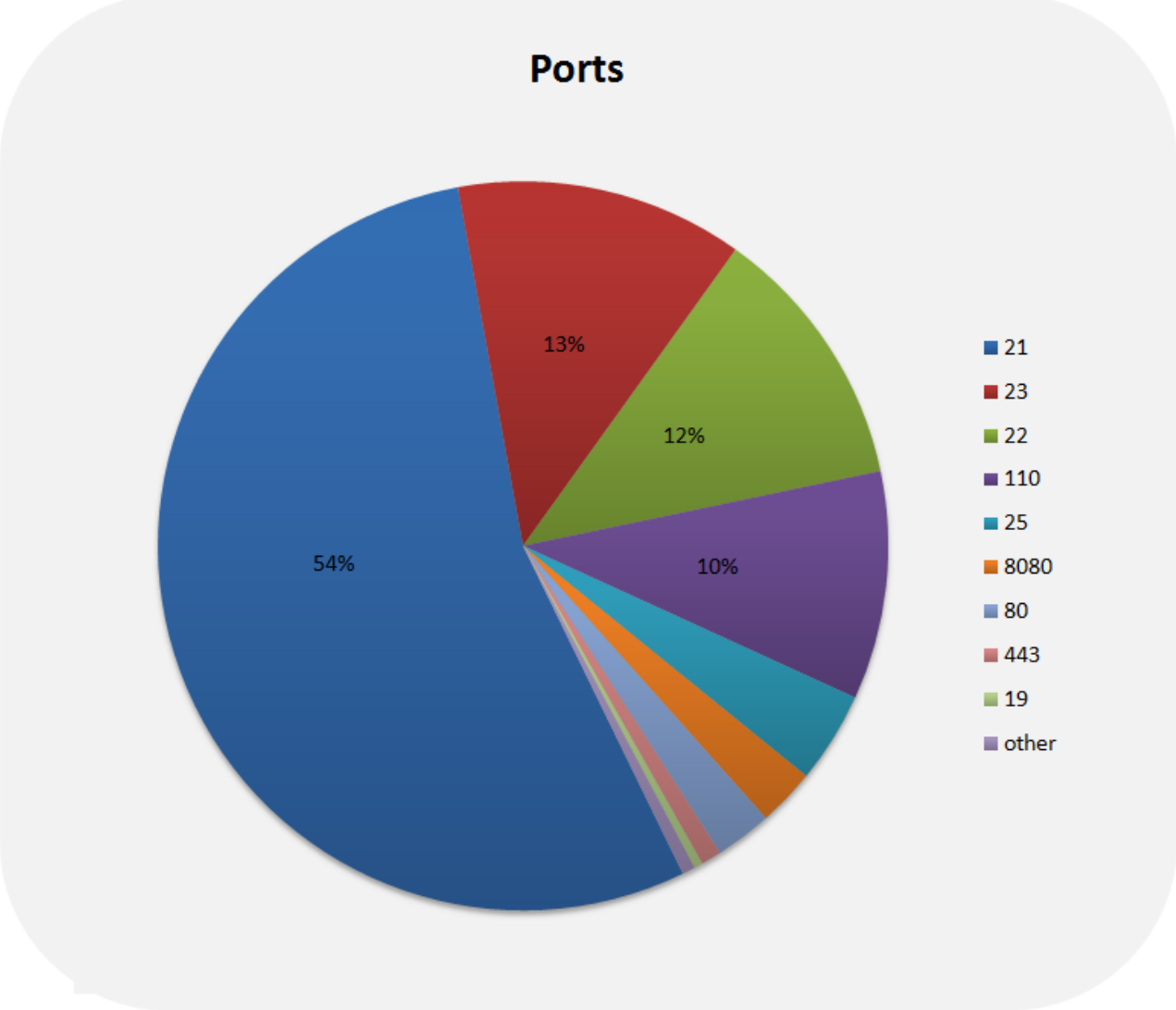

Figure 4.4: Most attacked ports – HoneyBot log file

From the information given in the above chart, it can be easily concluded that the majority of the attacks were on ports 21, 23 and 22. These ports represent FTP, TELNET and SSH services respectively and this chart shows that over 75% of the attack volume was targeting these three ports. The type of the attack, BRUTE FORCE attempt, can be identified from the type of service (FTP, TELNET, SSH). In addition, a large number of attack attempts were found on port 110, 110 Post Office Protocol v3 (POP3), third version of a widespread method of receiving email. Another

observation was the attacks on WEB ports like 8080, 443 and 80 with a total of around 6%. The type of the aforementioned attacks, were PORT and WEB SCANNING.

By analysing the origin of the attack traffic, 103 countries were identified, attacking or crawling/scanning the Honeypot. Again China (CN) and the United States (US) were the countries with the greatest attack rate, 36% and 12% respectively. Other countries with high attacking rate were Germany (DE), Turkey (TR), South Korea (KR), the Netherlands (NL), and France (FR) (Figure 4.5). The map below visualizes the above results, for a more qualitative representation (Figure 4.6).

Once more, it should be mentioned that from the total amount of countries that were examined, only the first forty had a number of attacks greater than 10. Therefore, the percentages may not seem that great, but that is only because the numbers of countries with a small rate of attacks are quite a lot.

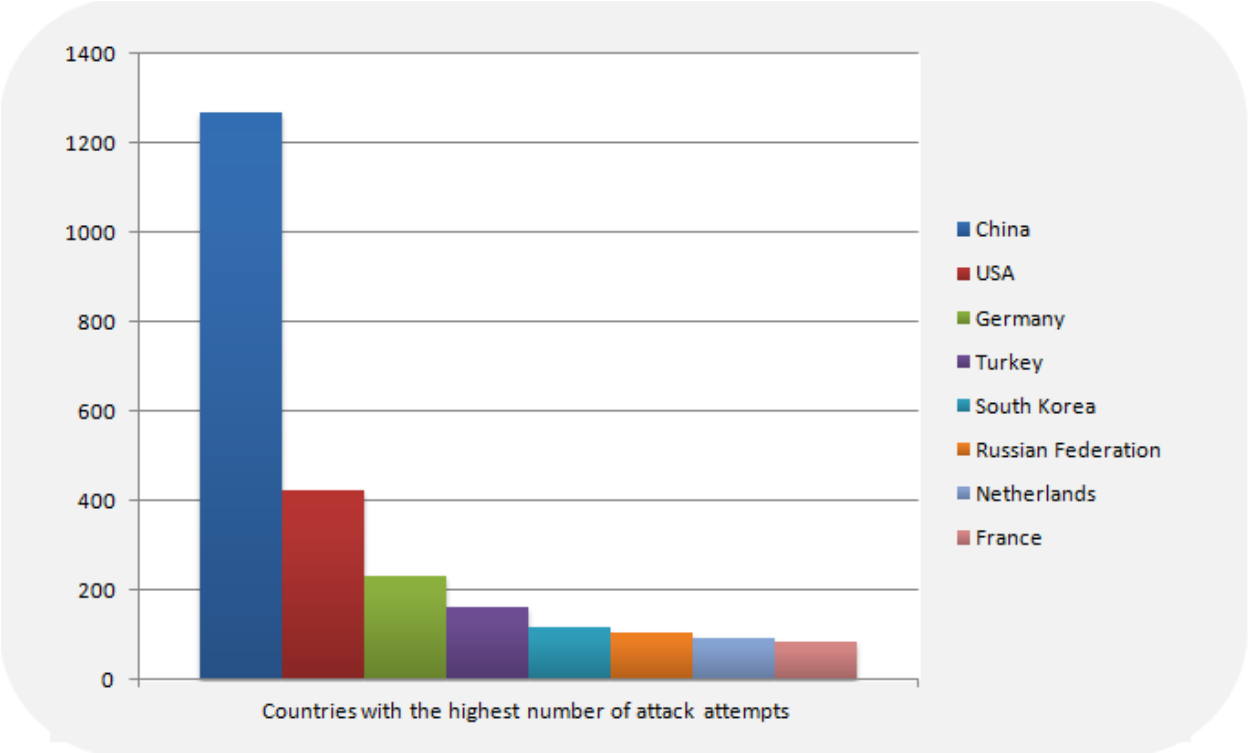

Figure 4.5: Countries with the highest number of attack attempts – HoneyBot log file

Figure 46: Visualization of Attacks per Country

4.2 Comparison Honeyd's' and HoneyBot's Results

The results that were presented in the above two sections, are quite similar. Both honeypots were attacked on similar ports, such as SSH (22), TELNET (23) and web ports 80, 8080. Also, the percentages showing the maximum targeted ports was referring to same types of ports SSH (22), TELNET (23). This was anticipated, since the connection to the majority of ~okeanos hosts is performed through the SSH and TELNET protocol. In addition, the increased number of attacks on web ports indicates a high amount of ~okeanos service users having installed web applications to their virtual machines lacking knowledge on how to protect them. As a result the ~okeanos infrastructure is threatened to be compromised.

The countries emerging as the origins of the highest portion of attack traffic were almost the same. Both tools received the majority of the number of attacks from China and the United States. Other countries targeting both tools as part of the ~okeanos infrastructure were South Korea, Turkey and France. Most of these countries are widely known for their hacking activity, so high attack traffic on an academic service whose users are not familiar with security measures is not surprising.

The next paragraph outlines the evaluation of Honeyd and HoneyBot, based on their performance and the quality of their results. Both tools gather similar types of characteristics. They keep the source and destination IPv4 and also the source and the destination port of each packet. This information is very useful and allows the security analyst to have a quick and valid insight on the infrastructure security status infrastructure constantly.

However, it seems that the amount of data gathered from each tool is different. Honeyd in a timeframe of half month managed to collect 6080 attacks. On the other hand, HoneyBot in a time frame of two months gathered 3575. This observation does not lead to the conclusion that Honeyd is a better tool than HoneyBot, taking into account the configuration approach followed in the latter, i.e. analyzing the behaviour of the existing ~okeanos VMs to allow for a more convincing imitation of their respective services. The conclusions that can be safely drawn from the above results are the following:

- The honeypot configuration approach is critical with respect to the amount of attacks it will be able to capture and consequently to the quality of data that it will gather
- Honeyd is capable of capturing high quality data with minimal effort for the actual configuration of the tool.

4.2 Explanation of data analysis tool

The tool that was created for the analysis of the results produced by the two honeypots was developed in ***Python***. Both tools' log files store the information of each packet in a row. The *Python* script reads every line and assigns each attribute to a variable. By processing the elements of each variable the amount and type of ports that were targeted and the number of IPv4 addresses where the attacks originated from were extracted. Finally, based on the information about each IPv4 address the source country of each attack was established.

For the statistical analysis of the results three functions were created. The first one is called ***find_unique*** and its purpose is to identify the unique elements in a *list* and return a dictionary containing the elements as *keys* and the number of appearances as *values*. The second one is called ***find_max*** and is used to sort the *keys* of a *dictionary* in descending order based on its values and return the pairs as a *list*. The last one is called ***draw_map***, and as its name indicates, it was used for the creation of a map which shows the countries with the maximum attacking rate.

Chapter 5

Conclusions

This chapter outlines the conclusions derived through the evaluation of statistical interpretation of the obtained data and refers to future work which could lead to effective addressing of specific security incidents.

5.1 Conclusions

The present diploma thesis describes the implementation of a Honeypot in the production environment of ~okeanos platform. Cloud computing infrastructures, such as ~okeanos, receive on a daily basis a great volume of attacks. The main objective of the diploma thesis was the exploration of the types of attacks that target ~okeanos platform. The tools that were used in this direction were Honeyd and HoneyBot. Both tools gave results of excellent quality and importance. Another objective was the creation of a tool that could help the security analyst to extract valuable conclusions about the status of the infrastructure in terms of security.

The tool created within the scope of this diploma thesis gives the capability of knowing the attacking trends of each period, the services that attract the majority of the attacks and the origin of each attack. All the above information is useful with respect configuring ~okeanos network in order to avoid the attacks which constitute the greatest threat for the infrastructure.

The information obtained by the two Honeypots and the analysis of their results could become an integral part of the incident handling process for two main reasons. The first reason is that they can be used for adding new or tuning existing configuration parameters e.g. adding some firewall rules or excluding some services from the infrastructure. Secondly, the results are very useful for educational services. In the information security field the weakest link in the chain is the human factor, so it is very important that the users of the service are well informed about the risks that they are facing. In this way they could learn to use stronger credentials and take all the necessary security measures.

The main purpose of this diploma thesis, which was the implementation of an integrated solution concerning an additional method of incident handling process of ~okeanos platform, was achieved. This document presents the deployment of Honeyd, based on the needs of ~okeanos service. Furthermore, it outlines a method for transferring the log data securely with FTPS protocol and finally, it describes the methods that were used for the statistical process of the results. The implementation was successful and the collected data provide an insight into the potential threats and the vulnerabilities that should be taken into account in effectively securing ~okeanos service.

In conclusion, the prevailing idea in the field of information security is that “one can never assume that one is safe”. On the contrary, one should always be vigilant and take the appropriate measures to ensure a high level of protection. This diploma thesis is considered to have fulfilled its purpose in this respect as it gives the security analyst a valuable tool to use in order to more effectively the production environment he works for.

5.2 Future Work

The purpose of this section is to present the ideas and suggest potential avenues that should be explored in extending the implementation of the solution that has been created within the scope of this diploma thesis.

Firstly, the results that have been produced from Honeyd could be further processed from a tool that is currently being developed within Grnet, named ASPIDA. The ASPIDA tool provides information about the services that are hosted on ~okeanos virtual machines, about the IPs that have been found in black lists and plenty of other services that extend the scope of the present diploma thesis. The information provided by Honeyd could be part of the services that are included in ASPIDA tool. At this point we should mention that we have created a suitable *dictionary*-based data structure in Python for the interconnection of the data produced by the Honeyd with the REST API of ASPIDA tool. Consequently, all the experiments would be executed from one integrated tool and the results would be gathered in the same way from a centralized platform. Moreover, the present implementation could be a part of other tools that have been developed from Grnet for security purposes.

An additional direction for future work concerns the deployment of supplementary services of Honeyd. Honeyd is capable of emulating many different services and network topologies. In addition, the present implementation could be used for the creation of a cluster of virtual machines that could host not only different types of Honeypots but also other security tools. This "security" cluster could be used as a tremendously powerful tool, which would allow the CERT team to manage each security issue in a more complete and comprehensive way.

References

[1]Kissel, Richard. "Glossary of key information security terms." NIST Interagency Reports NIST IR 7298 (2013): 3.

[2]http://www.forbes.com/sites/danmunro/2015/02/05/health-data-breach-at-anthem-is-a-blockbuster-could-affect-80-million/

[3]https://25zbkz3k00wn2tp5092n6di7b5k-wpengine.netdna ssl.com/files/2015/02/Carbanak_APT_eng.pdf

[4]Weber, Rolf H. "Internet of Things–New security and privacy challenges." Computer Law & Security Review 26.1 (2010): 23-30.

[5]ENISA: Cloud Security Incident Reporting – Framework for reporting about major cloud security incidents-December 2013.

[6]https://okeanos.grnet.gr

[7]ALERT LOGIC, 'CLOUD SECURITY REPORT, Research on the Evolving State of Cloud Security' (Spring 2014) [Available at: https://www.rackspace.com/knowledge_center/sites/default/files/whitepaper_pdf/ALERT-LOGIC-CLOUD-SECURITY-REPORT-Spring-2014.pdf]

[8]V. Koukis, P. Louridas, '~okeanos IaaS', (26-30 March, 2012) [Available at: https://okeanos.grnet.gr/static/medialibrary/2012/04/vkoukis-egicf2012.pdf]

[9]A. Zaharis, 'An Analysis of Network Security Incidents',2013 [Available at: https://cert.grnet.gr/sites/cert.grnet.gr/files/GRNET_CERT_2013v.1.1.pdf]

[10]Lance Spitzner, 'Honeypots: Tracking Hackers', Addison-Wesley Professional (September 20, 2002)

[11]ENISA: Proactive Detection of Security Incidents-Honeypots - 2012/11/20

[12]http://www.honeyd.org/

[13]https://www.honeynet.org/

[14]Peter, Eric, and Todd Schiller. "A practical guide to honeypots." *Washington Univerity* (2011).

[15]Provos, Niels. "A Virtual Honeypot Framework." USENIX Security Symposium. Vol. 173. 2004.

[16]A Honeypot System for Efficient Capture and Analysis of Network Attack Traffic, Abhay Nath Singh, R.C.Joshi 2011

[17]http://www.atomicsoftwaresolutions.com/

[18]HoneyBot Services: 'Client Data Collection',Team CIRCL [Available at: http://www.circl.lu/assets/files/honeybotclient.pdf]

[19]http://nmap.org/

[20]http://tools.ietf.org/html/rfc959

[21]http://tools.ietf.org/html/rfc4217

[22]https://filezilla-project.org/index.php

APPENDIX

```
### Default Template
create default
# Set default behavior

set default default tcp action block
set default default udp action block
set default default icmp action block

### Standard Windows 2000 computer
create win2k
set win2k personality "Microsoft Windows 2000 Server SP2"
set win2k default tcp action reset
set win2k default udp action reset
set win2k default icmp action allow
set win2k uptime 3567
set win2k droprate in 13
add win2k tcp port 21 "sh scripts/win32/win2k/msftp.sh $ipsrc $sport $ipdst $dport"
add win2k tcp port 80 "sh scripts/win32/win2k/iis.sh $ipsrc $sport $ipdst $dport"
add win2k tcp port 110 "sh scripts/win32/win2k/exchange-pop3.sh $ipsrc $sport $ipdst $dport"
add win2k tcp port 143 "sh scripts/win32/win2k/exchange-imap.sh $ipsrc $sport $ipdst $dport"
add win2k tcp port 389 "sh scripts/win32/win2k/ldap.sh $ipsrc $sport $ipdst $dport"
add win2k tcp port 5901 "sh scripts/win32/win2k/vnc.sh $ipsrc $sport $ipdst $dport"

# This will redirect incomming windows-filesharing back to the source
add win2k udp port 445 proxy $ipsrc:445
add win2k tcp port 445 proxy $ipsrc:445
set win2k ethernet "aa:36:29:2e:2e:2a"
bind 83.212.86.180 win2k

#### Linux Suse 8.0 template
create suse80
set suse80 personality "Linux 2.4.7 (X86)"
set suse80 default tcp action reset
set suse80 default udp action block
set suse80 default icmp action open
set suse80 uptime 79239
set suse80 droprate in 4
add suse80 tcp port 21 "sh scripts/unix/linux/suse8.0/proftpd.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp port 22 "sh scripts/unix/linux/suse8.0/ssh.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp port 23 "sh scripts/unix/linux/suse8.0/telnetd.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp port 79 "sh scripts/unix/linux/suse8.0/fingerd.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp port 80 "sh scripts/unix/linux/suse8.0/apache.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp port 110 "sh scripts/unix/linux/suse8.0/qpop.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp port 143 "sh scripts/unix/linux/suse8.0/cyrus-imapd.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp port 515 "sh scripts/unix/linux/suse8.0/lpd.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp port 3128 "sh scripts/unix/linux/suse8.0/squid.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp port 8080 "sh scripts/unix/linux/suse8.0/squid.sh $ipsrc $sport $ipdst $dport"
add suse80 tcp port 8081 "sh scripts/unix/linux/suse8.0/squid.sh $ipsrc $sport $ipdst $dport"
add suse80 udp port 514 "sh scripts/unix/linux/suse8.0/syslogd.sh $ipsrc $sport $ipdst $dport"
set suse80 ethernet "aa:36:29:2e:2e:2a"
bind 83.212.86.185 suse80
```

```
#### Suse7.0 computer
create suse70
set suse70 personality "Linux 2.2.12 - 2.2.19"
set suse70 default tcp action reset
set suse70 default udp action block
set suse70 default icmp action open
set suse70 uptime 97239
set suse70 droprate in 2
add suse70 tcp port 21 "sh scripts/unix/linux/suse7.0/proftpd.sh $ipsrc $sport $ipdst $dport"
add suse70 tcp port 22 "sh scripts/unix/linux/suse7.0/ssh.sh $ipsrc $sport $ipdst $dport"
add suse70 tcp port 23 "sh scripts/unix/linux/suse7.0/telnetd.sh $ipsrc $sport $ipdst $dport"
add suse70 tcp port 79 "sh scripts/unix/linux/suse7.0/fingerd.sh $ipsrc $sport $ipdst $dport"
add suse70 tcp port 80 "sh scripts/unix/linux/suse7.0/apache.sh $ipsrc $sport $ipdst $dport"
add suse70 tcp port 110 "sh scripts/unix/linux/suse7.0/qpop.sh $ipsrc $sport $ipdst $dport"
add suse70 tcp port 143 "sh scripts/unix/linux/suse7.0/cyrus-imapd.sh $ipsrc $sport $ipdst $dport"
add suse70 tcp port 515 "sh scripts/unix/linux/suse7.0/lpd.sh $ipsrc $sport $ipdst $dport"
add suse70 tcp port 3128 "sh scripts/unix/linux/suse7.0/squid.sh $ipsrc $sport $ipdst $dport"
add suse70 tcp port 8080 "sh scripts/unix/linux/suse7.0/squid.sh $ipsrc $sport $ipdst $dport"
add suse70 tcp port 8081 "sh scripts/unix/linux/suse7.0/squid.sh $ipsrc $sport $ipdst $dport"
add suse70 udp port 514 "sh scripts/unix/linux/suse7.0/syslogd.sh $ipsrc $sport $ipdst $dport"
set suse70 ethernet "aa:36:29:2e:2e:2a"
bind 83.212.86.184 suse70
```

Figure 6.1: configuration file of Honeyd

```
#Config file /etc/vsftpd.conf
#
# The default compiled in settings are fairly paranoid. This sample file
# loosens things up a bit, to make the ftp daemon more usable.
# Please see vsftpd.conf.5 for all compiled in defaults.
#
# READ THIS: This example file is NOT an exhaustive list of vsftpd options.
# Please read the vsftpd.conf.5 manual page to get a full idea of vsftpd's
# capabilities.
#
# Run standalone?  vsftpd can run either from an inetd or as a standalone
# daemon started from an initscript.
listen=YES
#
# Run standalone with IPv6?
# Like the listen parameter, except vsftpd will listen on an IPv6 socket
# instead of an IPv4 one. This parameter and the listen parameter are mutually
# exclusive.
#listen_ipv6=YES
#
# Allow anonymous FTP? (Beware - allowed by default if you comment this out).
anonymous_enable=NO
#
# Uncomment this to allow local users to log in.
local_enable=YES
#
```

```
# Default umask for local users is 077. You may wish to change this to 022,
# if your users expect that (022 is used by most other ftpd's)
#local_umask=022
#
# Uncomment this to allow the anonymous FTP user to upload files. This only
# has an effect if the above global write enable is activated. Also, you will
# obviously need to create a directory writable by the FTP user.
#anon_upload_enable=YES
#
# Uncomment this if you want the anonymous FTP user to be able to create
# new directories.
#anon_mkdir_write_enable=YES
#
# Activate directory messages - messages given to remote users when they
# go into a certain directory.
dirmessage_enable=YES
#
# If enabled, vsftpd will display directory listings with the time
# in  your  local  time  zone.  The default is to display GMT. The
# times returned by the MDTM FTP command are also affected by this
# option.
use_localtime=YES
#
# Activate logging of uploads/downloads.
xferlog_enable=YES
#
# Make sure PORT transfer connections originate from port 20 (ftp-data).
connect_from_port_20=YES
#
# If you want, you can arrange for uploaded anonymous files to be owned by
# a different user. Note! Using "root" for uploaded files is not
# recommended!
#chown_uploads=YES
#chown_username=whoever
#
# You may override where the log file goes if you like. The default is shown
# below.
#xferlog_file=/var/log/vsftpd.log
#
# If you want, you can have your log file in standard ftpd xferlog format.
# Note that the default log file location is /var/log/xferlog in this case.
#xferlog_std_format=YES
#
# You may change the default value for timing out an idle session.
#idle_session_timeout=600
#
# You may change the default value for timing out a data connection.
#data_connection_timeout=120
#
# It is recommended that you define on your system a unique user which the
# ftp server can use as a totally isolated and unprivileged user.
#nopriv_user=ftpsecure
#
```

```
# Enable this and the server will recognise asynchronous ABOR requests. Not
# recommended for security (the code is non-trivial). Not enabling it,
# however, may confuse older FTP clients.
#async_abor_enable=YES
#
# By default the server will pretend to allow ASCII mode but in fact ignore
# the request. Turn on the below options to have the server actually do ASCII
# mangling on files when in ASCII mode.
# Beware that on some FTP servers, ASCII support allows a denial of service
# attack (DoS) via the command "SIZE /big/file" in ASCII mode. vsftpd
# predicted this attack and has always been safe, reporting the size of the
# raw file.
# ASCII mangling is a horrible feature of the protocol.
#ascii_upload_enable=YES
#ascii_download_enable=YES
#
# You may fully customise the login banner string:
#ftpd_banner=Welcome to blah FTP service.
#
# You may specify a file of disallowed anonymous e-mail addresses. Apparently
# useful for combatting certain DoS attacks.
#deny_email_enable=YES
# (default follows)
#banned_email_file=/etc/vsftpd.banned_emails
#
# You may restrict local users to their home directories.  See the FAQ for
# the possible risks in this before using chroot_local_user or
# chroot_list_enable below.
#chroot_local_user=YES
#
# You may specify an explicit list of local users to chroot() to their home
# directory. If chroot_local_user is YES, then this list becomes a list of
# users to NOT chroot().
# (Warning! chroot'ing can be very dangerous. If using chroot, make sure that
# the user does not have write access to the top level directory within the
# chroot)
chroot_local_user=YES
chroot_list_enable=NO
# (default follows)
#chroot_list_file=/etc/vsftpd.chroot_list
#
# You may activate the "-R" option to the builtin ls. This is disabled by
# default to avoid remote users being able to cause excessive I/O on large
# sites. However, some broken FTP clients such as "ncftp" and "mirror" assume
# the presence of the "-R" option, so there is a strong case for enabling it.
#ls_recurse_enable=YES
# Customization
#Some of vsftpd's settings don't fit the filesystem layout by default.
#
# This option should be the name of a directory which is empty.  Also, the
# directory should not be writable by the ftp user. This directory is used
# as a secure chroot() jail at times vsftpd does not require filesystem
# access.
```

```
secure_chroot_dir=/var/run/vsftpd/empty
#
# This string is the name of the PAM service vsftpd will use.
pam_service_name=vsftpd
#
# This option specifies the location of the RSA certificate to use for SSL
# encrypted connections.
rsa_cert_file=/etc/ssl/private/vsftpd.pem
ssl_enable=YES
allow_anon_ssl=NO
force_local_data_ssl=YES
force_local_logins_ssl=YES
ssl_tlsv1=YES
ssl_sslv2=YES
ssl_sslv3=YES
# Filezilla uses port 21 if you don't set any port
# in Servertype "FTPES - FTP over explicit TLS/SSL"
# Port 990 is the default used for FTPS protocol.
# Uncomment it if you want/have to use port 990.
#listen_port=990
```

Figure 6.2: configuration file of Vsftpd

#The following piece of code processes the data taken from Nmap tool report. It finds the total number of open ports found in each host and then creates a dictionary containing the name of the port and its appearance frequency. Finally it calculates the most frequently open ports and plots them in a pie chart.

```
from __future__ import division
from libnmap.parser import NmapParser
from libnmap.process import NmapProcess
import operator
from matplotlib import pyplot as plt
import numpy as np
from utilities import *

#Passing the data of the Nmap report to variable nmap_report
nmap_report = NmapParser.parse_fromfile('/home/user/Desktop/Okeanos_Nmap/11-11-2014.xml')
hosts = nmap_report.hosts

open_ports_number = 0
open_ports_final = []
for host in hosts:
        open_ports = host.get_open_ports()
        open_ports_number += len(open_ports) #finally has the total number of open ports
        open_ports_final += open_ports #finally has the total list of open ports

#Creating dictionary containing the name of the port and its appearance frequency
unique_open_ports = find_unique(open_ports_final)
port_results = find_max(unique_open_ports)
max_ports = port_results[0]
max_value = port_results[1]
```

```
#Transforming the max_value into percentage for statistical use
N = 5
top_N_ports = []
top_N_values = []
total_max_value_percent = []
for s in range(N):
        top_N_ports.append(max_ports[s])
        top_N_values.append(max_value[s])
        max_value_percent = (top_N_values[s]/open_ports_number)*100
        total_max_value_percent += [max_value_percent]

rest_open_ports = ((open_ports_number-sum(top_N_values))/open_ports_number)*100
total_max_value_percent = total_max_value_percent+[rest_open_ports]

#Labeling and coloring of the pie chart
labels = top_N_ports+['Other']
colors = ['yellowgreen', 'lightcoral', 'gold', 'lightskyblue','violet', 'lightgrey']
explode = (0, 0, 0, 0, 0, 0) #no offsetting of slice

#Ploting the pie chart
plt.pie(total_max_value_percent, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%',
shadow=False)
plt.axis('equal') # Set aspect ratio to be equal so that pie is drawn as a circle
plt.show()
```

Figure 6.3: Visualization of data from Nmap tool

```
#The following piece of code processes the data taken from HoneyBot log file. It finds the number
of attacked ports (source_ports) of each host, the number of attacker ips (dest_ips) and the number
of attacker countries.

import requests
from utilities import *
from geoip import geolite2
from mpl_toolkits.basemap import Basemap
import operator
from matplotlib import pyplot as plt
import numpy as np

logs = open('Honeyd_Log.out')

c = 0 #key of dictionary that contains the id of each variable (e.g. date, source_ip, line)
tool = dict()  #name of the dixtionary

date = []
prot = []
t = []
source_ip = []
source_port = []
dest_ip = []
```

Figure 6.4: Parser of the Honeyd log file

```
for line in logs:
	if line.find('------')==-1:  #exclude first and last line that contain no useful info
		fields = line.split(' ') #Parcing the fields of every line
		date.append('-'.join(fields[0].split('-')[0:3]))
		prot.append(fields[1])
		t.append(fields[2])
		source_ip.append(fields[3])
		if prot[-1]!="icmp(1)": # exclude icmp packets
			source_port.append(fields[4])
			dest_ip.append(fields[5])
			pointer = str(fields[6]).find(":") # editing the format of fields[6] elements
			fields[6] = str.replace(str(fields[6]), "\n", "")
			if pointer!=-1:
				dest_port.append(str(fields[6])[0:pointer])
			else:
				dest_port.append(str(fields[6]))

#Finding unique destination ports and their appearance frequency
dictionary_of_dest_ports = find_unique(dest_port)
port_results = find_max(dictionary_of_dest_ports)
max_unique_ports = port_results[0]
max_unique_port_value = port_results[1]

#Finding unique source ips and their appearance frequency
dictionary_of_source_ip = find_unique(source_ip)
ip_results = find_max(dictionary_of_source_ip)
max_unique_ips = ip_results[0]
max_unique_ip_value = ip_results[1]

#Finding geolocation of each source ip and each country's appearance frequency
max_unique_ips = map(str, max_unique_ips)
country_name = []
for country in max_unique_ips:
	if country!='0':
		ip_info = geolite2.lookup(country)
		if ip_info!=None: #Avoiding zero entries
			country_name.append(ip_info.country)

unique_countries = find_unique(country_name)
countries_results = find_max(unique_countries)
max_unique_countries = countries_results[0]
max_unique_country_value = countries_results[1]

#Draw the most attacking country on map
draw_map(countries_results)

"""#Dictionary format for use in ASPIDA Tool
		tool.update({ (c,'date'): date,
		(c,'source_ip'): source_ip,
		(c,'destination_ip'): dest_ip,
		(c,'raw'): line
		})
		c+=1
#Connection with the REST API of ASPIDA Tool
r = requests.post("url", params=tool) #Response object"""
```

```
import csv   #The following piece of code processes the data taken from HoneyBot log file. It finds the number of
import sys  # attacked ports (source_ports) of each host, the number of attacker ips (dest_ips) and the number
import operator   # of attacker countries.
from mpl_toolkits.basemap import Basemap
from matplotlib import pyplot as plt
import numpy as np
from geoip import geolite2
from utilities import *
import copy

f = open('Log_20141124.csv', 'rt')
reader = csv.reader(f)

date = [0]
dest_ip = [0]
dest_port = [0]
source_ip = [0]
source_port = [0]
for row in reader:
        date.append(row[2])
        dest_ip.append(row[6])
        dest_port.append(row[7])
        source_ip.append(row[8])
        source_port.append(row[9])

#Finding unique source ports and their appearance frequency
dictionary_of_source_ports = find_unique(source_port)
port_results = find_max(dictionary_of_source_ports)
max_unique_ports = port_results[0]
max_unique_port_value = port_results[1]

#Finding unique destination ips and their appearance frequency
dictionary_of_dest_ip = find_unique(dest_ip)
ip_results = find_max(dictionary_of_dest_ip)
max_unique_ips = ip_results[0]
max_unique_ip_value = ip_results[1]

#Finding geolocation of each destination ip and each country's appearance frequency
max_unique_ips = map(str, max_unique_ips)
country_name = []
for c in max_unique_ips:
        if c!='0':
                ip_info = geolite2.lookup(c)
                if ip_info!=None: #Avoiding zero entries
                        country_name.append(ip_info.country)
unique_countries = find_unique(country_name)
countries_results = find_max(unique_countries)
max_unique_countries = countries_results[0]
max_unique_country_value = countries_results[1]

#Draw the most attacking country on map
print draw_map(countries_results)
```

Figure 6.4: Parser of the HoneyBOT log file

```
import operator

#This function takes as input a list of elements and returns a dictionary with the rate of appearance of each unique element

def find_unique(list_of_elements):

	unique_elements = set(list_of_elements) #find unique elements
	value_per_element = dict()
#Dictionary initialization
	for k in unique_elements:
		value_per_element[k] = 0

#The number of times each unique element has been found in the list is stored in the dictionary
	for unique_element in unique_elements:
		for element in list_of_elements:
			if element == unique_element:
				value_per_element[element]+=1
	return value_per_element
```

Figure 6.5: find_unique function

```
#This function takes as input the return of the "find_unique" function and returns the unique elements with their max value

def find_max(dictionary_of_elements):

#Finding most frequently occurring elements and their appearance value
	max_unique_elements = []
	max_unique_value = []

	results = sorted(dictionary_of_elements.items(), key=operator.itemgetter(1), reverse=True)

	for i in range(len(dictionary_of_elements)):
		max_unique_elements.append(results[i][0])
		max_unique_value.append(results[i][1])

	return [max_unique_elements, max_unique_value]
```

Figure 6.6: find_max function

```
#This function takes an input a list of countries with the major frequency appearance and prints a map with them

from __future__ import division
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
import numpy as np

def draw_map(countries_results):
	# Open the data file containing countries' latitudes and  longitudes
	data_file = open('country_latlon.csv')

	#Creation of a dictionary with these data
	world = dict()

	for line in data_file.readlines():
		fields = line.split(',')
		world[fields[0]] = (fields[1], fields[2].strip())

	# Store the latitudes, longitudes and the frequency appearance of each country in the appropriate lists
	N = 8 #The number of countries that are going to printed on the map
	lons = []
	lats = []
	attacks_size = []
	for i in range(N):
		lats.append(float(world[countries_results[0][i]][0]))
		lons.append(float(world[countries_results[0][i]][1]))
		attacks_size.append(countries_results[1][i]/sum_countries)
	sum_countries = sum(countries_results[1])

	# --- Build Map ---
	worldmap = Basemap(projection='mill', resolution = 'i', area_thresh = 100000.0, lat_0=0, lon_0=0)
	worldmap.drawcoastlines(color = 'ForestGreen')
	worldmap.drawcountries(linewidth =1, color = 'ForestGreen')
	worldmap.drawmapboundary(fill_color = 'Snow')
	worldmap.drawrivers(color = 'Snow')
	worldmap.fillcontinents(color = 'Snow', lake_color='Snow')

	n = 0
	for lon, lat, attack in zip(lons, lats, attacks_size):
		if attack>0.1:
			n=20
		else:
			n=10

		x,y = worldmap(lon, lat)
		worldmap.plot(x, y, 'ro', markersize = n)
	plt.show()
	return
```

Figure 6.7: draw_map function