

Master Thesis

Analysis and Deployment of Honeypot Solutions on Single Board Computers

Submitted by: Ovais Yousuf

Matriculation Number: 1025013

First examiner: Prof. Dr. Christian Baun

Second examiner: Prof. Dr. Dr. Herbert Nosko

Submission deadline : 11 April 2016

Statutory Declaration

I hereby declare that the thesis has been written by myself without any external unauthorized help, that it has been neither presented to any institution for evaluation nor previously published in its entirety or in parts. Any parts, words or ideas, of the thesis, including figures etc., which are quoted from or based on other sources, have been acknowledged as such without exception.

06-April-2016

OvaisYousuf

Acknowledgement

I thank all those who prayed for my success specially to my parents and continued asking about my achievements, hence boosting up my moral and giving me hope to go ahead.

It is indeed a great honor for me to express my deep indebtedness to my supervisor **Prof. Dr. Christian Baun** professor of Computer science and engineering in Operating Systems, Computer Networks, Distributed Systems (Cloud Computing) who gave the opportunity to write this thesis under his supervision and guided me throughout the Project. Their devotion and commitment enabled me to accomplish the task assigned. My countless appreciation goes to my other committee members as well expressly **Prof. Dr. Dr. Herbert Nosko**. I thank him for kindly agreeing to be on my master committee and for his helpful advice and suggestions. He spent his precious time and effort and provided a practical, international and friendly environment during my entire course.

I would also like to acknowledge my colleagues and friends in particular and all the family members in general. It is because of their devotion that I can see this day today. They imparted me the power of knowledge which is a base for not only this project, but also for the future tasks.

Table of Contents

Abstract	12
1 Chapter 1	13
1.1 Introduction	13
1.2 Goal	14
1.3 Advantage of a Honeypot over Network Intrusion Detection System (NIDS)	15
1.4 High Interaction Honeypots	15
1.5 Low Interaction Honeypots	16
1.6 Physical Honeypots	18
1.7 Virtual Honeypots	18
Chapter 2	19
2 Secure Shell Honeypot Kippo	19
2.1 Background	19
2.2 Features	20
2.3 Kippo Directories	21
2.4 Kippo File System	21
2.5 Requirements	22
Chapter 3	23
3 Glstopf Web Application Honeypot	23
3.1 Background	23
3.2 General Approach	24
3.3 Requirements	26

Chapter 4	27
4 Honeyd the Virtual Honeypot	27
4.1 Background	27
4.2 Design of the HoneyD	27
4.3 Architecture	28
4.4 Personality Engine	30
4.5 Logging	31
Chapter 5	32
5 Raspberry Pi 2	32
5.1 Interfaces and Functions	32
5.2 Nmap	33
5.2.1 Most Commonly Used Commands	34
5.3 Hydra	35
5.3.1 Hydra Working	35
5.4 Medusa	36
5.5 Putty	37
Chapter 6	38
6 Realisation and Implementation	38
6.1 Installation of Raspbian Wheezy Image	39
6.1.1 Taking Remote of the Raspberry Pi through Putty	41
6.2 Installation of Honeyd on Raspberry Pi 2	45
6.2.1 Setting up the fake virtual windows machine using HoneyD	55
6.2.2 Scanning the functionality of HoneyD by performing Nmap Foot printing	58
6.2.3 Testing the functionality of HoneyD via Brute Force Attack using Hydra	60
6.3 Installation of Kippo Honeypot on Raspberry Pi 2	63
6.3.1 Integrating MySQL Database with Kippo	74
6.3.2 Creating the Fake File System using fs.pickle	78
6.3.3 Testing the functionality of Kippo via Brute Force Attack using Medusa from Attacker Machine	80

6.3.4	Logging Feature of Kippo Server	84
6.4	Installation of Glastopf Honeypot on Raspberry Pi 2	87
6.4.1	Install and Configure the PHP sandbox	95
6.4.2	Testing the Functionality of Glastopf Honeypot through a LFI attack from a Linux Machine.....	105
7	Summary and Future Perspectives	110
7.1	Summary	110
7.2	Future perspectives.....	111
8	Abbreviations	113
	Bibliography	116

List of Figures

Figure 1.1.1: General Block diagram of a Honey pot solution.....	12
Figure 1.4.1: Example of a High Interaction Honeypot.....	15
Figure 1.5.1: An example of a Low Interaction Honeypot.....	16
Figure 3.2.1: Glastopf General Architecture.....	25
Figure 3.2.2: Flowchart of Glastopf Engine.....	26
Figure 4.2.1: Pictorial representation of the Honeyd design.....	28
Figure 4.3.1: the basic working architecture of Honeyd.....	30
Figure 5.1.1: Raspberry Pi Model B.....	32
Figure 5.2.1: Sample output of Nmap against a Linux host.....	34
Figure 5.3.1: The protocol list on which hydra has been tested.....	36
Figure 5.5.1: Putty Graphical User Interface GUI.....	37
Figure 6.1.1: Selection of Wheezy Image.....	39
Figure 6.1.2: Burning of the Image	40
Figure 6.1.3: The Image has been mounted successfully.....	40
Figure 6.1.1.1: IP address assigned dynamically from the DSL Router.....	41
Figure 6.1.1.2: Putty manager to take the remote of the Raspberry Pi.....	42
Figure 6.1.1.3: Raspberry Pi login screen.....	43
Figure 6.1.1.4: Raspberry Pi login screen.....	43
Figure 6.1.1.5: Raspberry Pi Setup Window.....	44
Figure 6.2.1: Updating the packages in raspberry pi.....	45
Figure 6.2.2: Installing the libdnet package.....	46
Figure 6.2.3: Installation of pre-requisite for Honeyd.....	47
Figure 6.2.4: Installing git.....	48
Figure 6.2.5: Changing the directory.....	48

Figure 6.2.6: Cloning the Honeyd to the local machine.....	49
Figure 6.2.7: Navigating to the directory Honeyd.....	49
Figure 6.2.8: Running the autogen.sh script.....	50
Figure 6.2.9: Configuring the build parameters.....	51
Figure 6.2.10: Screenshot of the make installation.....	52
Figure 6.2.11: Installation of Honeyd software.....	53
Figure 6.2.12: Honeyd starting as background process.....	54
Figure 6.2.13: Installation of 'farpd'.....	54
Figure 6.2.1.1: Conceptual setup of the HoneyD Implementation.....	55
Figure 6.2.1.2: Configuration file screen shot.....	56
Figure 6.2.1.3: Launching screen shot of HoneyD.....	57
Figure 6.2.1.4: Ping output from the Windows machine.....	58
Figure 6.2.2.1: Screen shot of Honeyd launching with -l flag.....	59
Figure 6.2.2.2: Nmap scanned output.....	60
Figure 6.2.3.1: Screenshot of the Attack.....	61
Figure 6.2.3.2: Logs created by HoneyD.....	62
Figure 6.3.1: Snapshot of the configuration file.....	64
Figure 6.3.2: Restarting the server.....	64
Figure 6.3.3: Snapshot with the new port.....	65
Figure 6.3.4: Screenshot for installing git.....	66
Figure 6.3.5: Screen shot of the authbind.....	67
Figure 6.3.6: Screen shot for adding the new user.....	67
Figure 6.3.7: Screen shot of the user values.....	68
Figure 6.3.8: Screenshot for sudo user.....	69
Figure 6.3.9: Screenshot for git cloning.....	70
Figure 6.3.10: Screen shot of kippo.cfg file.....	71

Figure 6.3.11: Screenshot of start.sh.....	72
Figure 6.3.12: Screenshot of netstat output.....	72
Figure 6.3.13: Kippo running in the background.....	73
Figure 6.3.14: Screenshot of netstat output.....	73
Figure 6.3.1.1: Screenshot for installing MySQL.....	74
Figure 6.3.1.2: Screenshot for MySQL root user.....	75
Figure 6.3.1.3: Screenshot for creating database for Kippo.....	76
Figure 6.3.1.4: Screenshot of selecting kippo as a database.....	77
Figure 6.3.1.5: Screenshot of killing kippo process.....	78
Figure 6.3.1.6: Screenshot of Database configuration.....	78
Figure 6.3.2.1: Screenshot of the fake file system in fs.pickle.....	79
Figure 6.3.2.2: Screenshot for new user and password.....	80
Figure 6.3.2.3: Screenshot for loading mysql and starting kippo.....	80
Figure 6.3.3: Conceptual diagram of brute force attack on kippo.....	81
Figure 6.3.3.1: Screenshot of Nmap foot printing.....	82
Figure 6.3.3.2: Screenshot of the attack from Linux machine.....	83
Figure 6.3.3.3: Screenshot of taking the access of kippo server.....	84
Figure 6.3.3.4: Screenshot of fake file system.....	85
Figure 6.3.4.1: Screenshot of accessing the MySQL database.....	86
Figure 6.3.4.2: Screenshot of the sessions log created.....	86
Figure 6.3.4.3: Screenshot of auth log created.....	87
Figure 6.3.4.4: Screenshot for replay logs of the attacker sessions.....	88
Figure 6.4.1: Screenshot of putty manager.....	89
Figure 6.4.2: Screenshot of glastopf packages installations 1.....	90
Figure 6.4.3: screenshot of glastopf package installation 2.....	91
Figure 6.4.4: Screenshot of glastopf packages installation 3.....	92

Figure 6.4.5: Screenshot of glastopf packages installation 4.....	93
Figure 6.4.6: Screenshot of glastopf packages installation 5.....	94
Figure 6.4.7: Screenshot of glastopf packages installation 6.....	95
Figure 6.4.8: Screenshot of glastopf packages installation 7.....	96
Figure 6.4.1.1: Screenshot of php-sandbox cloning from git.....	96
Figure 6.4.1.2: Screenshot of php sandbox installation.....	97
Figure 6.4.1.3: Screenshot of configuration and enabling BFR.....	97
Figure 6.4.1.4: Screenshot of running make and installing make.....	98
Figure 6.4.1.5: Adding the path to php.ini file.....	99
Figure 6.4.1.6: Screenshot of php zend engine installation.....	99
Figure 6.4.1.7: Screenshot of libinjection cloning from git.....	100
Figure 6.4.1.8: Screenshot of pylibinjection cloning from git.....	100
Figure 6.4.1.9: Screenshot of building python setup.py.....	101
Figure 6.4.1.10: Screenshot of cloning glastopf from git.....	101
Figure 6.4.1.11: Screenshot of python setup.py install.....	102
Figure 6.4.1.12: Screenshot of glastopf-runner.....	103
Figure 6.4.1.13: Screenshot of glastopf.cfg file.....	103
Figure 6.4.1.14: Screenshot of glastopf web server running.....	104
Figure 6.4.1.15: Screenshot of the webpage running on glastopf.....	104
Figure 6.4.1.16: Screenshot of glastopf.....	105
Figure 6.4.1.17: Screenshot for installation of sqlite3.....	105
Figure 6.4.1.18: Screenshot of the database fields and events.....	106
Figure 6.4.2.1: Conceptual Figure of the Attack scenario.....	107
Figure 6.4.2.2: Screenshot of LFI attack from attacker machine.....	107
Figure 6.4.2.3: Screenshot of the glastopf logs.....	108
Figure 6.4.2.4: Screenshot of glastopf window.....	108

Figure 6.4.2.5: Screenshot of /etc/passwd file accessed by the attacker.....	109
Figure 6.4.2.6: Screenshot for the analysis of logs in sqlite3 database.....	109
Figure 6.4.2.7: Screenshot of further analysis of the logs.....	110

Abstract

In the past several years there has been extensive research into honeypot technologies, primarily for detection and information gathering against external threats. [1]

This thesis brings the concept of analysing and deployment of the open source honeypot solutions on the raspberry pi 2. The selected honeypot solutions are mainly low-level interaction but one used in the thesis work is a medium-interaction. The honeypots are installed on single board computers to test the functionalities. The test scenarios are created in the LAN without the internet access. For this kind of situation when there is no availability of internet self made tests and attacks are created to prove the functionality of selected honeypots. Three kinds of honeypots are used in this thesis work. One is a SSH server honeypot, second is the web server honeypot and the last is the virtual honeypot.

To test the functionality of all the honeypots some brute force attack techniques are used. These attacks are performed with the help of hacking tools to crack the password credentials and by using some available python scripts.

The proof of the functioning part of these selected honeypots is their attack detection with the nature of attack and the logs that are generated after and during the attacks. With this the network security can be made more protected and secure.

1 Chapter 1

This chapter provides a deep background about the technologies involved in this Master Thesis. In order to understand the realization of the research work it is essential to have a look at the different possible technologies and techniques. Therefore, it would be useful and necessary to understand background knowledge of these technologies, protocols and software used in this thesis work.

1.1 Introduction

Honeypots are an influential, new equipment with unbelievable prospective. Honeypots are able to do everything from detecting fresh attacks never seen in their natural habitat before, from tracking programmed credit card scam and individuality stealing. By the passage of time and robust growth of the technology many conceptual things related to commercial honeypots and open source solutions with fully recognized sources are available.

Nevertheless, a vast arrangement of research pointed on capturing discovering, and researching outdoor threats. Being nasty and treacherous, attacks made by the attackers are frequently casual as the attackers further interested in the quantity of the systems they can break into other then which systems they shatter into. For hazardous and shocking type of threats the amount of research is not as much as it should be at this point, the sophisticated user of the organization. The network of the company is well recognized by this reliable user who is actually not mainly interested in the computers of the company but in the precise information of the company, and this is extremely dangerous issue to handle and to diminish.

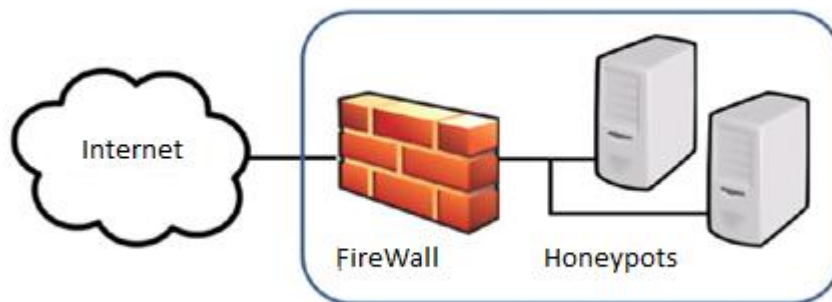


Figure 1.1.1: General Blockdiagram of a Honey pot solution

A honeypot is considered as an inimitable security reserve. A resource for that one wanted to be interconnected by the black chat community.

“A honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource.” [1]

The above lines of quotations explain that a honeypot is a resource that gains their value if they interact with the attacks. And it is totally diverse from usual security mechanisms. [1]

1.2 Goal

The goal of this thesis is as follows:

- This thesis work presents the analysis of three different honey pots solutions on a single board computer that is Raspberry Pi 2.
- Check the functionalities of some honeypot solutions
- Implement a honeypot solution on a single board computer
- Investigate how to test the functionality.
- Perform some simulated attacks and real attacks.
- Summary of the final result based on functionalities of all three honeypots.

1.3 **Advantage of a Honeypot over Network Intrusion Detection System (NIDS)**

The purpose of this section is to present the basic comparison in NIDS and honeypots. The main motivation in their use by contrasting honeypots with network intrusion detection systems (NIDS) is that NIDS are losing their importance of tricky techniques to make the network secure as the new secure protocols are available which has the ability of encrypting the data plus to secure the network traffic very privately.

Also NIDS undergoes from high false positive rates that decreases their worth even more. Honeypots are capable to figure these types of problems.

Honeypot is somewhat a resource that is placed so that it should get compromised and probed as it is strongly observed. A Honeypot becomes indomitable from the received information when probed. The logging feature of the honeypot is the main asset that provides the information about entering and leaving of the data or information which is not available in NIDS. [2]

The logging ability of the honeypot is quite capable of logging the key strokes of the associated session yet the network traffic is well encrypted.

On the other hand NIDS entail the signature of recognized attacks and normally not succeeded when goes in contact with the coming signatures when implemented.

But honeypots are able to notice vulnerabilities that are even not yet understood. To make sense it means that honeypots keeps an eye on the network traffic passing through the honeypot even if the signature of the exploit is all new because directing any traffic to honeypot is suspicious because honeypot is nothing but a decoy system.[2]

1.4 **High Interaction Honeypots**

A high-interaction honeypot is a straight computer system which means a commercial off-the-shelf (COTS) computer. It may be a router or may be just a normal switch. These sorts of systems have no usual activities in the network as no active users are connected to them. So no normal or daily routine traffic is generated as there is no processes are running other than some services that are made to run to attract the intruder. This phenomenal

specialty of these kinds of honeypots helps in detecting the attack alarms. So who ever comes in contact with the high-interaction honeypot will be considered as apprehensive, and so every system activity will be logged and stored for all the traffic towards high-interaction honeypots. For instance the pictorial representation of the High interaction honeypot can be seen in the figure 1.4.1 below.

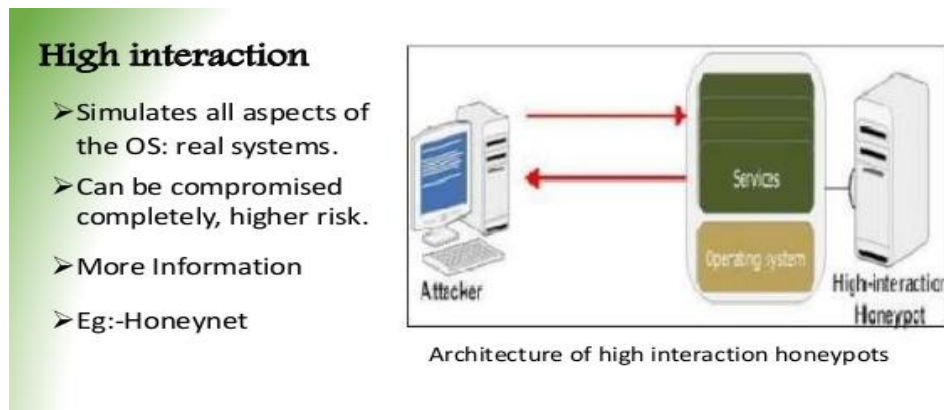


Figure 1.4.1: Example of a High Interaction Honeypot [3]

High-interaction honeypots use original operating systems with every possible defect so that they can be completely compromised. No fake services are used to emulate so that the original flavor of the system and services can be enjoyed by the intruder. On the other hand, the network personals can observe the wide threat information by monitoring the tools used by the intruder and gaining the knowledge about the bad ambitions of the intruder once they get into the system without permission. The only dark side of high-interaction honeypots is that they play their role on high risk as the intruder has a full right to use the operating system. [2]

1.5 Low Interaction Honeypots

The factors which distinguish low-interaction honeypots are the emulated services and the network stacks, which are the replica from an original operating system. These types of honeypots are designed in a manner that they provide only significant information to the network personals about the activities performed by the attacker. The pictorial representation of the Low interaction honeypot can be seen in the figure 1.5.1 below.

To, understand this lets take if an attacker asks for HTTP pattern from the HTTP server the low-interaction honeypots will answers only then a specific file from the complete HTTP pattern asked for. Here it means that the level of control moves should be very precise to fool the attacker or the dangerous tools used against the system, which can be a virus or a worm to bait the system. The benefit of low-interaction honeypots is their straightforwardness and simple upholding. It is really simple just go and deploy the system and start collecting the information of the intruder.

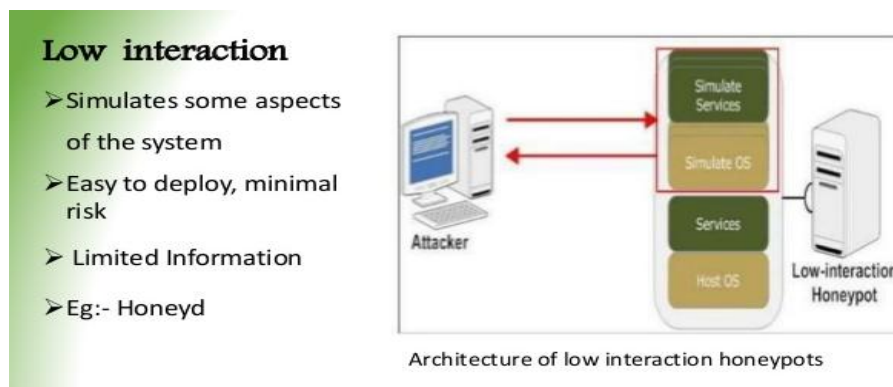


Figure 1.5.1: An example of a Low Interaction Honeypot [4]

The intruder is helpless to completely compromise the honeypot since they come in contact just with a simulation. Low-interaction honeypots build controlled surroundings so the risk concerned is restricted. This gives a peace to one that one does not have to fear that he abuses their honeypots. [2]

High-Interaction	Low-Interaction
Real services, OS's, or applications	Emulation of TCP/IP stack, vulnerabilities, and so on
Higher risk	Lower risk
Hard to deploy and maintain	Easy to deploy and maintain
Capture extensive amount of information	Capture quantitative information about attacks

Table 1: Advantages and Disadvantages of High and Low Interaction Honeypots [2]

The above Table 1 provides the summarized overview of high and low interaction honeypots explaining the advantages and disadvantages of each approach.

1.6 Physical Honeypots

The next possibilities are physical and virtual honeypots in the field of honeypots. The physical honeypots are the one that are deployed on the original machines which normally means a high-interaction honeypots but it can be a low-interaction honeypot also. This means that a single honeypot is deployed with one assigned IP address from the pool of IP addresses per machine.

This type of system are very difficult to implement and to take care of as assigning the IP address per machine is very unfeasible and sounds unpractical. For such type of cases virtual honeypots are the best solutions available in the market. [2]

1.7 Virtual Honeypots

The last type, which is the virtual honeypots they are very appealing. The basic sense in their deployment is the scalability and easy look after. One can deploy more than hundreds of honeypots just on a single physical machine. The attractiveness of the virtual honeypots is they are very reasonable to deploy and easily available on the internet for anybody who wants to have it.

In comparison with physical honeypots virtual honeypots are more trivial. This means here that one can place thousands of virtual host on a network with only one computer which acts as a honeypots other than deploying a single honeypot on a single computer. This will help in maintenance of the honeypots and the requirements for implementation will also be less. [2]

Chapter 2

In the following chapter the technology which is discussed is related to SSH Honeypot named as Kippo. This chapter is a description regarding the background and will discuss some important features, directories and the system requirements for the SSH kippo honeypot.

2 Secure Shell Honeypot Kippo

2.1 Background

A very frequent method to take shell access remotely of any operating system is by means of a Secure Shell network protocol SSH. SSH protocol is quite able in providing a remote access to an insecure network area. For this the SSH server is available to provide an authentic access to the user who uses SSH protocol to get the remote access. The methods used by the attackers to get the remote login of the SSH server might be any available technique or tool available in the market it can be a brute force tool or a dictionary tool for attack.

In order to study the activities executed by attackers after they get into a system with an SSH server, one can use a Kippo honeypot. Kippo SSH honeypot is somewhat in between the high-interaction and low-interaction honeypot called medium-interaction honeypot. Kippo honeypot is intended to log many of the brute force attacks and the most important feature is it logs the complete shell interaction executed by the intruder

Kippo honeypot provides a feature of a fake file system by simulating a Debian linux system which is seen by the intruder at the time of the login and execution. The intruder can move around into the system but unable to destroy anything.

Kippo honeypots provides the attacker the complete login facility of the system and the feel to the attacker that he is into a genuine SSH session with in the server. Once getting into the system after a successful breakup of the password the intruder can go around into the fake system. Every single step executed by the attacker is saved and observable. Kippo honeypot provides

many Linux commands that are used in daily routine. The attacker will be provided a facility to download any file from the internet also, which will be stored in the “dl” directory of the kippo honeypot.

Kippo honeypot is autonomous to any operating system as it is developed in python programming language. In real meaning, through valuable imitation it is able to let an attacker to log in and come in contact with what they think is original, compromised system.

The main idea of the operation is to bring to the attacker the notion of navigating the actual system. [5]

2.2 Features

The most attracted features of the Kippo honeypot that makes it attracted for the users to use are as follows

- Fake file system with the capacity of building and deleting the files. A completely artificial file system similar to a Debian 5.0 mechanism is integrated.
- Opportunity provided for adding contents to a file which are not real contents of any production system. This will trap the attacker who will try to use ‘cat’ command for files such as /etc/passwd. Only nominal file contents are integrated.
- Kippo also support the UML file format for the session logs saved in. which can later analyze with original timings.
- Accumulates all the files which were downloaded throughout the SSH session (simulating the commands for downloading which is wget) for later scrutiny.
- Accumulates the material about the intruder’s activities in the operating system (the used commands) in a format granting the replay in a screen-cast format.
- Simulates the conclusion of the SSH session. Which means when the attacker types the “exit” command to finish the session it actually don’t finish the session it will provide another shell like terminal for collecting extra information about the attacker’s activity.

2.3 Kippo Directories

Most vital directories and files of the kippo system are as follows

- **data:** It is a directory for the data files that are mixed in nature, such as the database for passwords.
- **fs.pickl:** A virtual file system that is developed in the format of Python pickle.
- **dl:** When an attacker downloads any worm or virus or any kind of exploit from internet they are saved in this directory which is designed only for this purpose.
- **kippo.cfg:** It is the heart of the kippo server which plays an instructive role.
- **honeys:** In this directory one can design their own fake directories which includes any file that can be accessed by the attacker.
- **txtcmds:** One can create some extra Linux commands that are not available by default in the kippo honeypot and this is possible in this directory.
- **log:** This directory is the main directory which provides every possible information about the attack and the session created.
- **start.sh:** It's a directory in which the start script of the kippo is available.
- **utils:** The playlog.py utility is available in this directory which is used for the complete session replay.

2.4 Kippo File System

Kippo is capable of storing the files in a system known as pickle file.

Pickle is the benchmark mechanism for object soap. The object rebuilding is provided by pickle as it uses a stack based virtualization process. Pickle can be defined as a partial memory space that provides a binary format for the objects to store on a hard disk. This will also help the user in retrieving the stuff from the memory afterwards.

Whenever the new user connects to the kippo honeypot, it loads this pickle file system into the memory. It does not affect the pickle file system in real whatever the attacker did there it will be reloaded as fresh and new system on the next access of the file system.

2.5 Requirements

The software that is compulsory to utilize Kippo honeypot are as follows

- An operating system (may be a Debian, CentOS and Windows platforms)
- Python 2.5+
- Twisted 8.0+
- PyCrypto
- Zope Interface

Chapter 3

Chapter 3 discusses the background knowledge required and will discuss the general approach needed to implement the Glastopf web application server side honeypot including the system requirements for the glastopf technology.

3 Glastopf Web Application Honeypot

3.1 Background

Presently, 60% of total number of attempted attacks over the internet is against web applications. Organizations cannot bear the cost of their websites to be compromised, as this can effect in serving mean content to customers, or leaking customer's data. Certain features are commonly available to all web applications, whether the particular web application is an element of a personal web page, or a company's website. Mainly people belief in the consistency of web applications and they are often hosted on powerful servers with higher bandwidth connections with the Internet. Taking into account the large number of attacks and knowing the possible consequences of successful break-ins, we are determined to put more effort into the development of honeypots to better recognize these attacks. Glastopf is competent of emulating hundreds of vulnerabilities to assemble data from attacks that aim web applications. The principle behind it is quite simple. Reply to the attack using the reaction the attacker is expecting from his endeavour to exploit the web application. There are presently other web application honeypots available, but ours uses a diverse approach. For example, In spite of the adapted web app templates used by search entities to catch the fanciness more attacks over time, Glastopf supports multistage attacks, list of vulnerable requests and a vulnerability emulator. The key principle of a low interaction honeypot is simple. With most of the currently available automated honeypots, one just have to start the program, watch the bad guys attacking, send the collected files to a sandbox, exhibit the attack

events in a web interface. This will help one to assess the collected data and to understand how to collect and process incoming attacks. [6]

Glastopf compiles data by emulating hundreds of network holes. Contrasting many other honeypots, Glastopf focuses on responding with the correct reply to the attacker exploiting the targeted Web application, and not the particular vulnerability.

3.2 General Approach

Lukas Rist has founded Glastopf, which is basically a python based web application server side honeypot. [7]

- Vulnerability type emulation which means not mainly vulnerability emulation. Glastopf can handle mysterious attacks of the same type after the vulnerability type is once emulated. Might be bit slower and more complicated during the implementation phase, but it remains ahead of the intruders until they appears with a new technique or determine a new fault in this realization.
- Modular design to include new logging capabilities or attack type handlers. A variety of database capabilities are already situated. HPFeeds logging is facilitated for centralized data collection.
- Known attack kind emulation is previously located. Remote File Inclusion via an integrated PHP sandbox, Local File Inclusion facilitating files from a virtual file system and HTML injection by using POST requests.
- Adversaries, typically make use of search engines and special crafted search requests to locate their victims. In order to magnetize them, Glastopf offers those keywords (as known as "dork") and in addition extracts them from requests, extending its attack plane involuntarily. Resulting, the honeypot gets more and more striking with each fresh attack made over it.
- SQL injection emulator will be made public, which will provide IP assigning for crawler detection and intelligent dork selection.

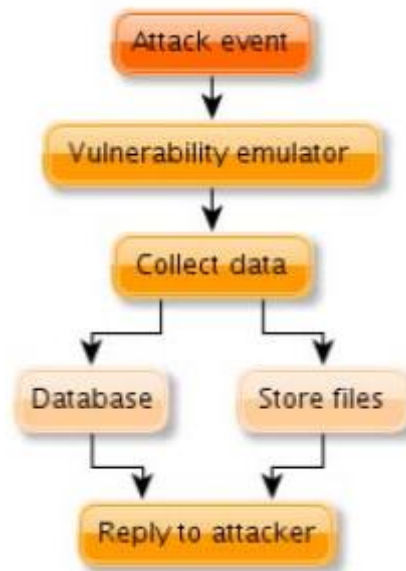


Figure 3.2.1: Glastopf General Architecture [6]

It has limited capability of responding to particular types of attacks, so as to aggravate the attacker to send further information, while trying to compromise the emulated web server. The general architecture of the glastopf can be seen in the figure 3.2.1 above.

There is a central database for attack information storing. The central database helps to centralize the hits on every single Glastopf honeypot. These way researchers may have access to large information about web server attacks.

It supports statistics fun-out through irc channels or tweeter. It connects to a particular channel and in real time, broadcasts information about incoming attacks. Figure 3.2.2 is the pictorial representation of the Glastopf flowchart below.

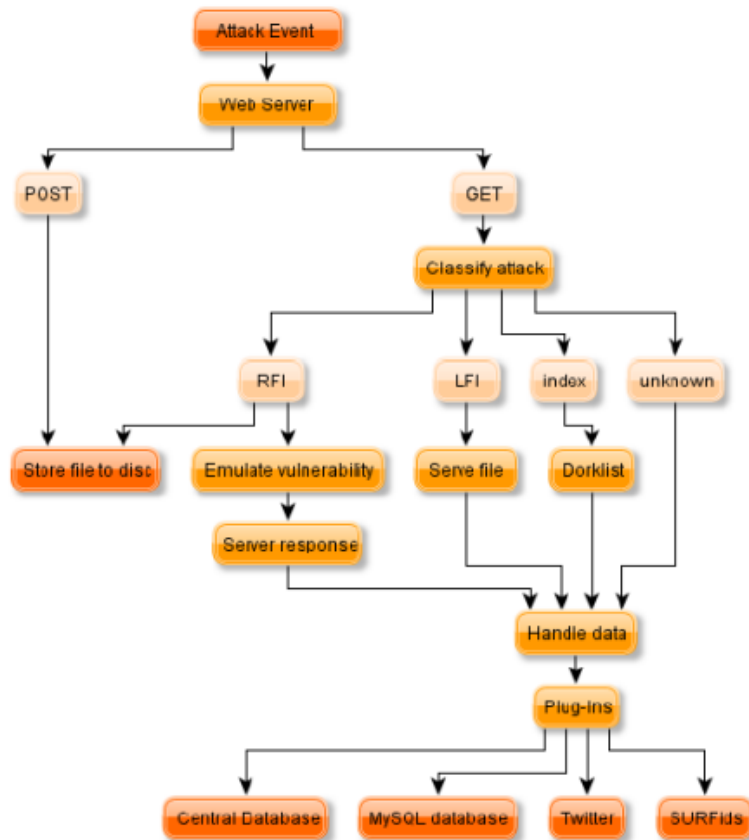


Figure 3.2.2: Flowchart of Glastopf Engine [6]

3.3 Requirements

- Install the required dependencies
- Install and configure the PHP sandbox
- Install pylibinjection
- Install Glastopf
- Configuration
- Testing the Honeypot

Chapter 4

This chapter provides an overview of the virtual honeypot technology used in this master thesis. This chapter discusses the background knowledge with the basic design and architecture of the virtual honeypot Honeyd.

4 Honeyd the Virtual Honeypot

4.1 Background

With the passage of time the IT business is expanding on daily basis, with this expansion of IT infrastructure the importance of the internet security is also increasing rapidly. There are many available internet security solutions in the market.

Honeyd is software for virtual honeypots. It has an ability to simulate the systems at the third layer of the OSI model. Honeyd provides a facility to configure the services for every virtual honeypot by responding the network requests for each virtual host as honeyd facilitates the IP protocol stack of the OSI model. This means that when a request comes it goes to the personality engine where it compares it with the network stack of the personality assigned to the operating system.

Honeyd is a framework with too much power it depends on the user how one is using it on the network. Honeyd is quite capable of identifying and stopping the viruses and network worms. [8]

4.2 Design of the HoneyD

Honeyd is trivial software used to create thousands of virtual hosts. One can take simply a single machine and can deploy thousands of IP addresses with different network services running individually. In this part of the thesis lets briefly describes the design of the honeyd.

In the design of the honeyd the designer of this tool keeps in mind that honeyd should operate at the network layer which means the attacker who tries to interact with this system is limited to the third layer of the network. In easy words the simulation of the network stack take into account not the whole aspect of the operating system. The dark side is the attacker is not able to compromise the complete emulated service. But nevertheless it can monitor the active connections and the attempts made. With this ability TCP and UDP including the ICMP responses are also well served. Figure 4.2.1 is the pictorial representation. [8]

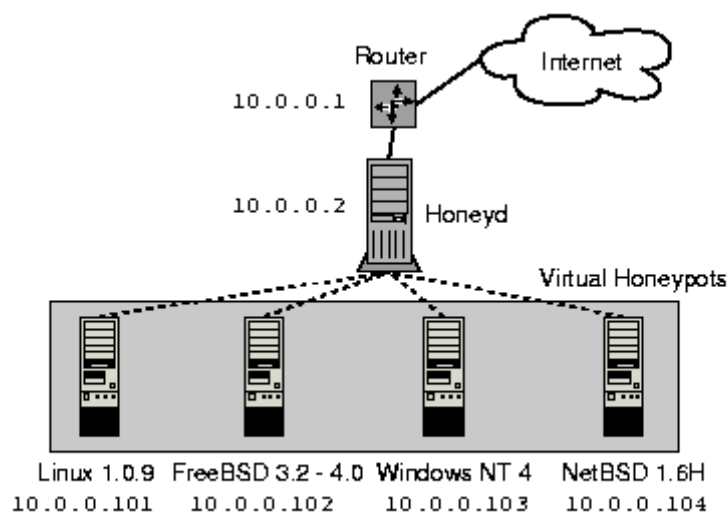


Figure 4.2.1: Pictorial representation of the Honeyd design [8]

4.3 Architecture

Honeyd's architecture contains several components, a central packet dispatcher, an optional routing component, protocol handlers, a personality engine, and a configuration database that is shown in the figure below 4.3.1.

Central packet dispatcher processes the incoming packets. It first takes the length of an IP packet into account and packet's checksum is verified. Framework is conscious about the three major Internet protocols TCP, ICMP and UDP. Packets for other protocols are saved and simply discarded.

The dispatcher must inquire the configuration database before it processes any packet, to find a honeypot configuration that corresponds to the destination IP address. A default template is used, if no specific configuration exists. Given a configuration, the packet and subsequent configuration is forwarded to the protocol specific handler.

Most ICMP requests are backed up by the ICMP protocol handler. By default, all honeypot configurations react to echo requests and produce destination unreachable messages. Handling of other requests depends on the configured characteristics.

For TCP and UDP, the framework is able to launch connections to arbitrary services. Services are exterior applications, which receive data at stdin and their output are sent to stdout. The performance of a service depends completely on the external application. The tool examines if the packet is part of an established connection when there is a connection request is received. In that case, every new entry is sent to the ongoing service application. If the packet contains a connection request, a fresh process is shaped to run the proper service. The framework backs up subsystems and internal services instead of establishing a new process for each connection made. The function that executes in the name space of the virtual honeypot is called subsystem. The subsystem specific application is started when the subsequent virtual honeypot is incorporated. A subsystem can connect to ports, acknowledge connections, and commence network traffic. An internal service is executed within Honeyd, which is a Python script while a subsystem runs as an external process. Internal services require even fewer possessions than subsystems but are only able to accept connections and not begin them.

A simple TCP state machine is contaminated in a Honeyd. The three-way handshake for connection initialization and teardown using FIN or RST is totally supported, but receiver and congestion window management is not completely implemented.

UDP datagram are passed directly to the application. When the software receives a UDP packet for a closed port, it sends an ICMP port unreachable message until this is not allowed by the configured personality. In sending ICMP port unreachable messages, the framework permits network mapping tools like traceroute to determine the simulated network topology. [8]

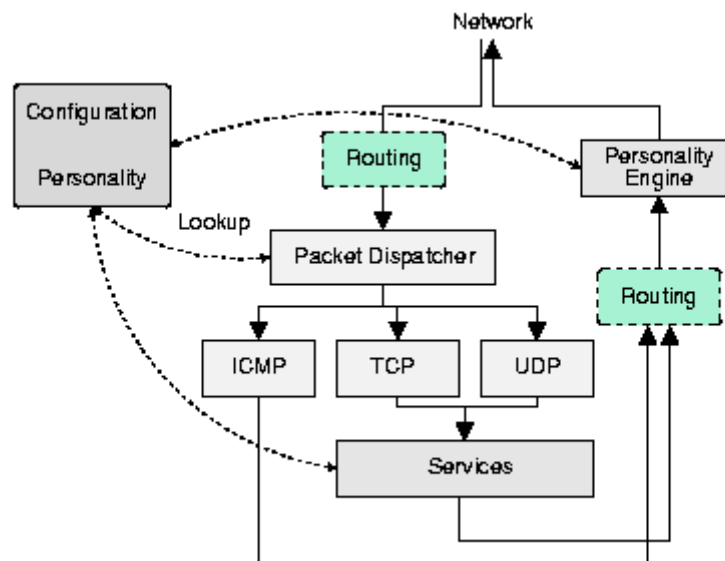


Figure 4.3.1: the basic working architecture of Honeyd [8]

4.4 Personality Engine

Fingerprinting tools like Xprobe or Nmap are commonly run by adversaries to compile credentials about a objective system. It is essential that honeypots do not move out of the way, when fingerprinted. To make them emerge real to a probe, Honeyd simulates the network behaviour of a given OS. This is known as the characteristic of a virtual honeypot. Various characteristics can be assigned to various virtual honeypots. The personality engine makes a honeypot's network stack perform as specified by the characteristic by introducing changes into the headers of the protocol of each outgoing packet in order to match the personalities of the configured OS.

The framework uses Nmap's fingerprint database as its reference for a personality's TCP and UDP behaviour. Xprobe's fingerprint database is used as reference for a personality's ICMP behaviour.

4.5 **Logging**

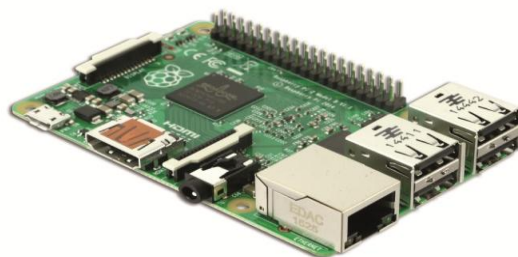
The Honeyd framework backs up numerous ways of logging network activities. It can construct connection logs, which reports attempted and concluded connections for every single protocol. More beneficially, credentials can be compiled from the services. Service applications are able to report information to be logged to Honeyd by using stderr. The framework utilizes syslog to accumulate the information on the machine. [8]

- -l is a flag that is used to store the packets and connections to the logfile

5 Raspberry Pi 2

The enormous popularity of the Raspberry Pi 2 minicomputers with the majority stems from a variety of DIY projects powered by this inexpensive hardware, as well as many freely available open-source software suites. Figure 5.1 shows the hardware components of the Raspberry Pi 2

This is not a consumer device, and depending on what we intend to do with Raspberry Pi 2. The figure 5.1.1 below shows the picture of the Raspberry Pi 2.



32

- **The Secure Digital (SD) Card slot:** Pi doesn't have a local storage like HDD or a SDD; everything is stored on an SD Card.
- **USB port:** There are two USB 2.0 ports. A powered external hub can be used if a desired peripheral needs more power.
- **Ethernet port:** The model one RJ45-Ethernet port.
- **HDMI connector:** The HDMI port offers digital audio/video output. 14 unlike video resolutions are supported, and the HDMI signal can be changed to DVI, composite, or SCART with external adapters.
- **Status LEDs:** Pi has five display LEDs that provide visual feedback. ACT Green Lights when the SD card is accessed, PWR Red Curved up to 3.3V power, FDX gets green if the network adapter is fully duplex, LNK Green Network motion light 100, Yellow On if the network connection is 100Mbps.

Raspberry Pi 2 is a single board computer (SBC) and performance level is also adequate. Presently, Raspberry Pi 2 is not open source hardware. It uses the ARM based multimedia system-on-chip (SoC) with exclusively-functional, entirely open-source drivers, vendor-provided.

5.2 Nmap

Nmap, a network exploration tool and security scanner, is defined by its author, Fyodor, which is really a substitute for operating system fingerprint. Various numbers of hosts on the Internet can be promptly scanned with Nmap to find out which operating system is being used and which services are being offered. To resolve the operating systems and services running on 192.168.1.1, Nmap can be invoked as a root user with the below mentioned command:

```
nmap -sS -O -F 192.168.1.1
```

The output in Figure 5.2.1 informs us that 192.168.1.6 is running a current version of the Linux kernel. Nmap cannot quite inform which version, so it gives a choice between Linux version 2.4.18 and Linux version 2.6.7. These versions keep up a correspondence to the kernel versions that are run by the hosts. Each kernel might select to execute TCP vaguely differently, and

Nmap uses these differences to identify the operating system being run by the host. In this case, the scanned Linux host was certainly running 2.6.7. It is interesting to see that TCP timestamps, allow Nmap to conclude, how long the machine has been running. In this particular case, the host has been up for about one hour. The general foot printing signature of the nmap can be seen in the figure 5.2.1 below. [10]

```
Code View:
Starting nmap 4.11 ( www.insecure.org/nmap/ ) at 2007-01-16 17:45 PDT
Interesting ports on 192.0.2.1:
(The 1208 ports scanned but not shown below are in state: closed)
PORT STATE SERVICE
9/tcp open  discard
13/tcp open  daytime
21/tcp open  ftp
22/tcp open  ssh
23/tcp open  telnet
25/tcp open  smtp
37/tcp open  time
79/tcp open  finger
80/tcp open  http
111/tcp open rpcbind
113/tcp open  auth
139/tcp open netbios-ssn
445/tcp open microsoft-ds
MAC Address: 00:09:5B:AF:34:11 (Netgear)
Device type: general purpose
Running: Linux 2.4.X|2.5.X|2.6.X
OS details: Linux 2.4.18 - 2.6.7
Uptime 0.033 days (since Tue Jan 16 16:57:58 2007)

Nmap finished: 1 IP address (1 host up) scanned in 3.883 seconds
```

Figure 5.2.1: Sample output of Nmap against a Linux host running several services [10]

5.2.1 Most Commonly Used Commands

Nmap is a beyond doubt a composite tool, though a concise overview of the most frequently used command lines are:

- **-p port-ranges:** Specifies ports to be scanned
- **-sV:** Enables version detection, i.e.; Nmap tries to recognize which service is running along with the version on the specified port.
- **-O:** Enables remote operating system's detection
- **-A:** Enables OS detection and version detection

- **-T[0-5]:** Sets the timing option that is a bigger number, which means smaller amount of time between these probes.
- **-oN/-oX/-oG file:** These options specify the output format as normal and XML, respectively. The scan report saved for further analysis.

These six command line flags are sufficient for common use of Nmap. The tool is principally supportive for verification of the targeted honeypot is up/running. It can also find out whether all services are running or not.

5.3 Hydra

A very well-known network log on cracker which can support various different services is Hydra, which is now in version 8.1 and last updated December 12th 2014 (Similar projects and tools include [13] and [12]).

5.3.1 Hydra Working

Hydra is a password breaking tool. In network security (IT security), password breaking is the method of guessing passwords from databases that are stored in or are in transit contained by a computer system or network. A general approach, and the approach used by Hydra and many other similar testing tools is referred to as Brute Force. A concise bytes on 'Brute Force Hacking' could be simply done but since this part of the thesis is all relating Hydra let's leave the brute-force attack concept surrounded by this password-guessing tool.

Brute force simply means that the program launches a persistent stream of passwords at a log in for password guessing. As it is known, majority of the users have relatively weak passwords and all too often they can be easily guessed. A bit of social engineering and the probability of finding the exact password for a user are enhanced. Most people (especially those non-IT professionals, base their 'secret' passwords on nouns and words that they might not easily forget. Those words are usually: loved ones, children's names, street addresses, place of birth etc. All of these are easily obtained through social media so as soon as the hacker has collected this data, it can be without much effort be compiled within a 'password list'.

Brute force will include the list that the hacker has built up and will likely merge it with other known tools and begin the attack. Depending on the processing speed of the hacker's machine, Internet connection the brute force method will analytically poll through each credential until the exact one is determined. The protocol access list for hydra can be seen in the figure 5.3.1 below. [11]

afp	cisco	cisco-enable	cvs
firebird	ftp	http-get	http-head
http-proxy	https-get	https-head	https-form-get
https-form-post	icq	imap	imap-ntlm
ldap2	ldap3	mssql	mysql
ncp	nntp	oracle-listener	pcanywhere
pcnfs	pop3	pop3-ntlm	postgres
rexec	rlogin	rsh	sapr3
sip	smb	smbnt	smtp-auth
smtp-auth-ntlm	snmp	socks5	ssh2
teamspeak	telnet	vmauthd	vnc

Figure 5.3.1: The protocol list on which hydra has been tested [11]

5.4 Medusa

Medusa is projected to be a prompt, extremely parallel, modular, login brute-forcer. The objective is to sustain as many services which permit remote validation as possible. Brute-force testing can be performed against numerous hosts, users or passwords in parallel.

Flexible user input. Target information (host/user/password) can be specified in a range of ways. For example, each item can either be a single entry or a

file containing several entries. Moreover, a combinational file format allows the user to filter their target record.

It has a modular architecture which means every single service module exists as a self-governing **.mod file**. This means that no modifications are essential to the core application in order to enlarge the supported list of services for brute-forcing. [13]

5.5 Putty

Putty is an SSH and telnet client, designed initially by Simon Tatham for the Windows platform. Putty is available with source code and is open source software, developed and supported by a group of volunteers. The GUI of putty can be seen in the figure 5.5.1 below.

To download putty:

<http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>

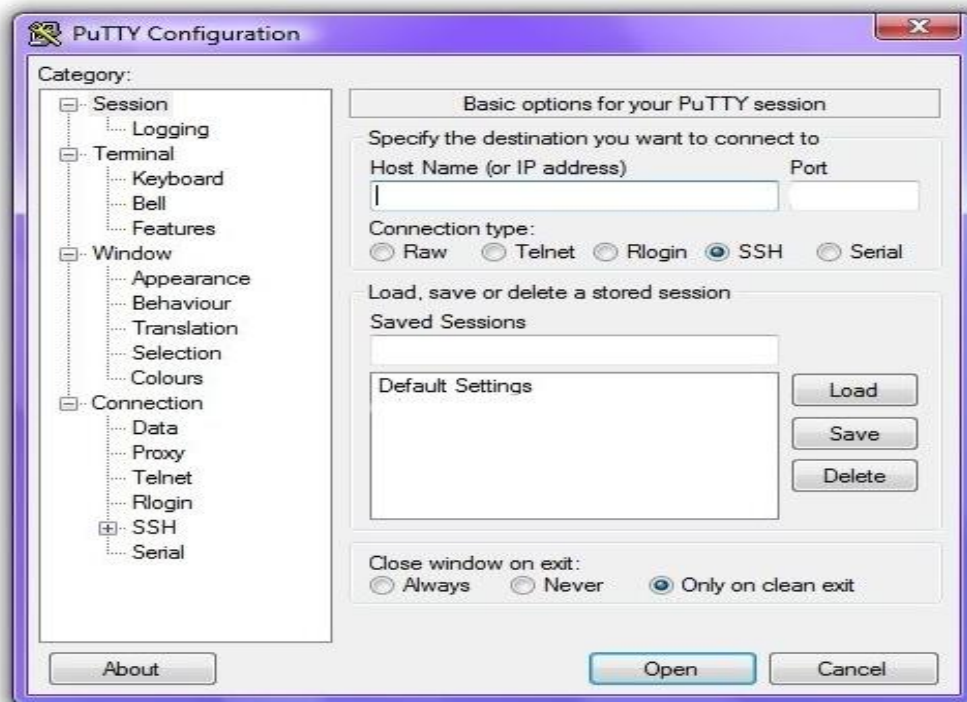


Figure 5.5.1: Putty Graphical User Interface GUI [14]

Chapter 6

This section of the report gives a detailed implementation of the whole testing environment which includes setting up raspberry pi 2, Honeypot packages installations, configurations and screenshots containing various outputs.

6 Realisation and Implementation

To simplify implementation procedures and understand the realization concept, it can be divided into following parts:

- The raspberry pi 2 setup which includes installation of Raspbian wheezy image on the 16GB SD card using WinDiskImager32 to get run on raspberry pi 2.
- Installation of Kippo Honeypot packages along with the supporting tools to be deployed on raspberry pi and can make in a running condition and can then later tested internally or locally by another linux machine on the same LAN.
- Installation of Glastopf Honeypot packages along with the supporting tools to be deployed on raspberry pi 2 and can make in a running condition and can then later tested internally or locally on the same LAN.
- Installation of HoneyD Honeypot packages along with the supporting tools to be deployed on raspberry pi 2 to get it in a running condition and can then later tested by brute forcing it through the Linux machine internally or locally on the same LAN.

6.1 Installation of Raspbian Wheezy Image

The first step in installation and deployment of different server side honeypots it is needed to burn the Raspbian Wheezy image on the Micro SD card using Win32 Disk Imager.

- Download the Win32 Disk Imager from the following web link
http://www.chip.de/downloads/Win32-Disk-Imager_46121030.html
- Then Download the Raspbian Wheezy Image from the following link
<https://www.raspberrypi.org/downloads>

After downloading of both the software lets first open the Win32 Disk Imager and select the path under the name (Device) from the download directory of the computer where Raspbian Wheezy is downloaded to burn on the Micro SD card.

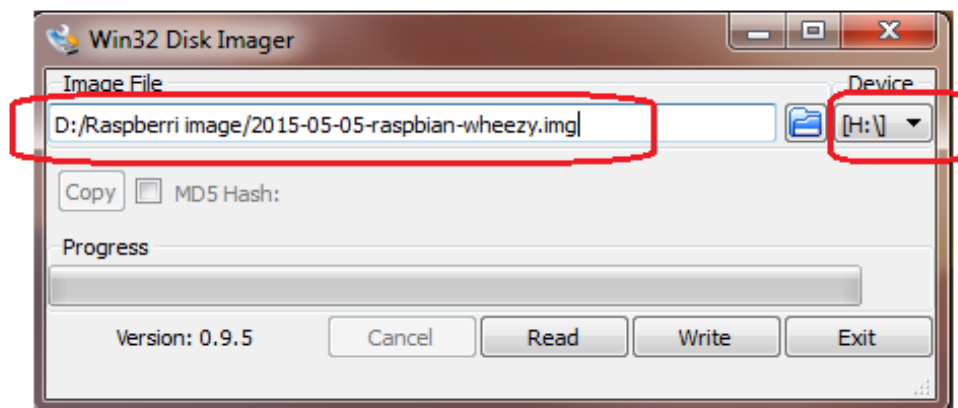


Figure 6.1.1: Selection of Wheezy Image to burn with Win32 Disk Imager

After the selection of the right image path from the download section the next step is to press the (Write) button to burn the Image on the SD card which can be seen in the figure 6.1.2 below.

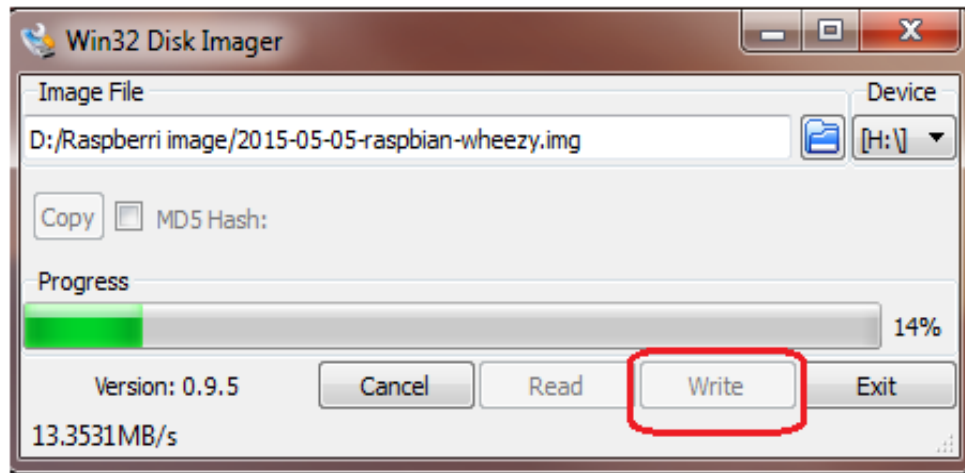


Figure 6.1.2: Burning of the Image in progress by pressing Write option

After the progress has been completed the Image will be effectively mounted on the SD card which can be seen in the figure 6.1.3 below.

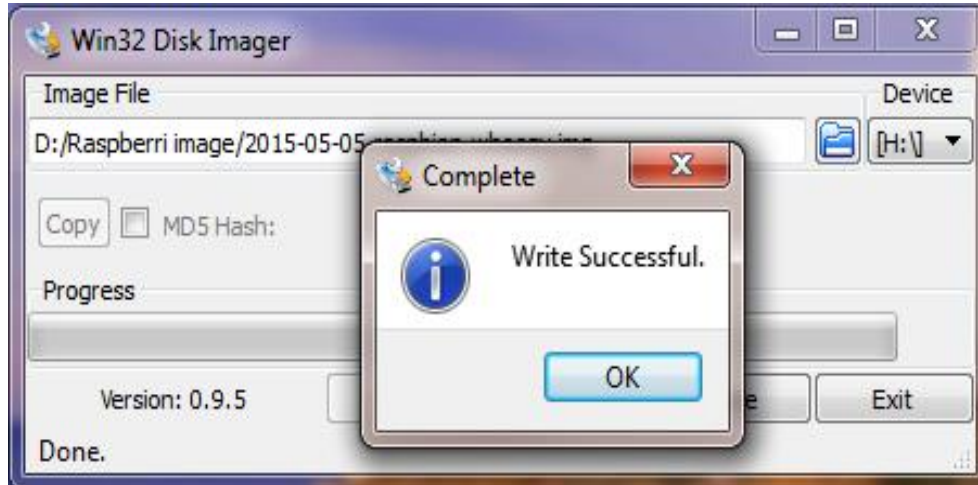


Figure 6.1.3: The Image has been mounted successfully

After the successful mounting of the Raspbian ISO image on the SD card one should insert the SD card into the SD slot space of the Raspberry Pi. The next one thing needed to make raspberry pi 2 up is to give the electricity which can be provided by the separate power adapter or in our case through the PC with the normal charging cable

6.1.1 Taking Remote of the Raspberry Pi through Putty

There are many ways to configure the Raspberry Pi.

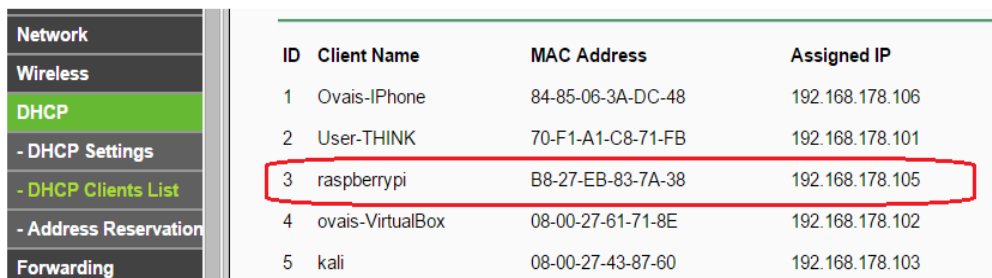
- One can configure the Raspberry Pi through connecting the Raspberry Pi with the HDMI cable into the Screen with a mouse and keyboard and can take the Desktop of the Raspberry Pi by installing the VNC server to take GUI of raspberry Pi.
- The other option is taking the Remote session by using Putty manager to work in the CLI mode of the Raspberry Pi

Let's download the Putty manager from the following link:

<http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>

To connect or to take the remote session of the Raspberry Pi 2 an IP address must be known which can be provided by the DSL TP-Link home router as the Raspberry Pi 2 is connected with a 5e Ethernet cable with the TPLink DSL home router.

The IP is being dynamically provided by the DSL router and can be seen in the figure below 6.1.1.1.



Network	ID	Client Name	MAC Address	Assigned IP
Wireless	1	Ovais-IPhone	84-85-06-3A-DC-48	192.168.178.106
DHCP	2	User-THINK	70-F1-A1-C8-71-FB	192.168.178.101
- DHCP Settings	3	raspberrypi	B8-27-EB-83-7A-38	192.168.178.105
- DHCP Clients List	4	ovais-VirtualBox	08-00-27-61-71-8E	192.168.178.102
- Address Reservation	5	kali	08-00-27-43-87-60	192.168.178.103
Forwarding				

Figure 6.1.1.1: IP address assigned dynamically from the DSL Router

After obtaining the IP address for the Raspberry Pi which is **192.168.178.105**. The port should be set to 22 always as this method is used to take the SSH session of the raspberry Pi. Let's open the Putty manager and provide the IP address obtained from the router and press open button to get the remote SSH access of the raspberry Pi CLI terminal. Which can be seen in the figure 6.1.1.2 below.

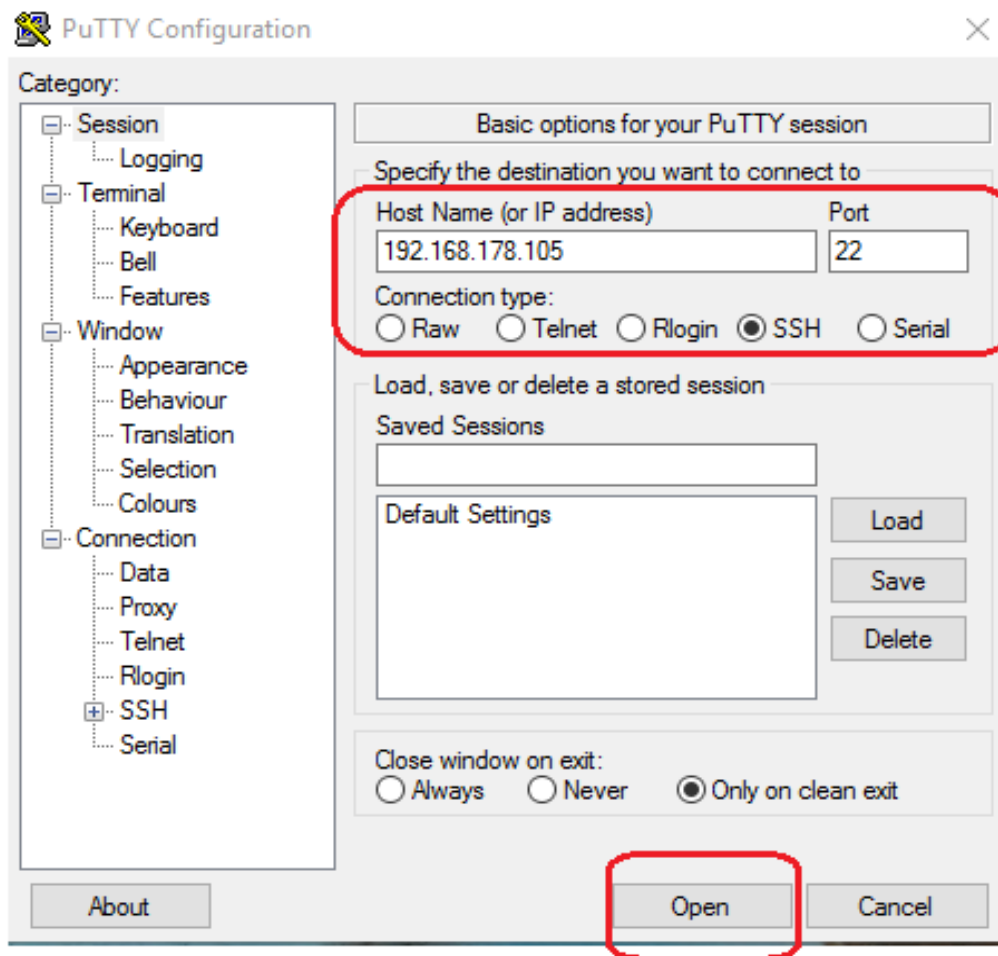


Figure 6.1.1.2: Putty manager to take the remote of the Raspberry Pi CLI

After the successful connection from putty manager to the raspberry Pi a black window will be open to provide the credentials of the Raspberry Pi shown in figure 6.1.1.3 below.



Figure 6.1.1.3: Raspberry Pi login screen

After this step the next figure 6.1.1.4 below shows the credentials needed to get the terminal access of the Raspberry pi which is by default:

Login: pi

Password: raspberry

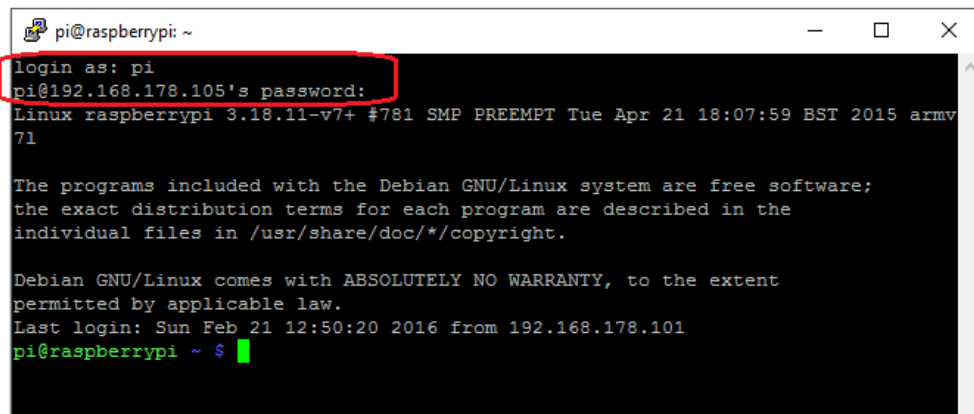


Figure 6.1.1.4: Raspberry Pi login screen

Once the Pi has been started for the first time, a configuration window called the "Setup Options" will emerge after entering the command

```
$ sudo raspi-config
```

After entering the above command the next a window will be opened which can be seen in the figure 6.1.1.5 below

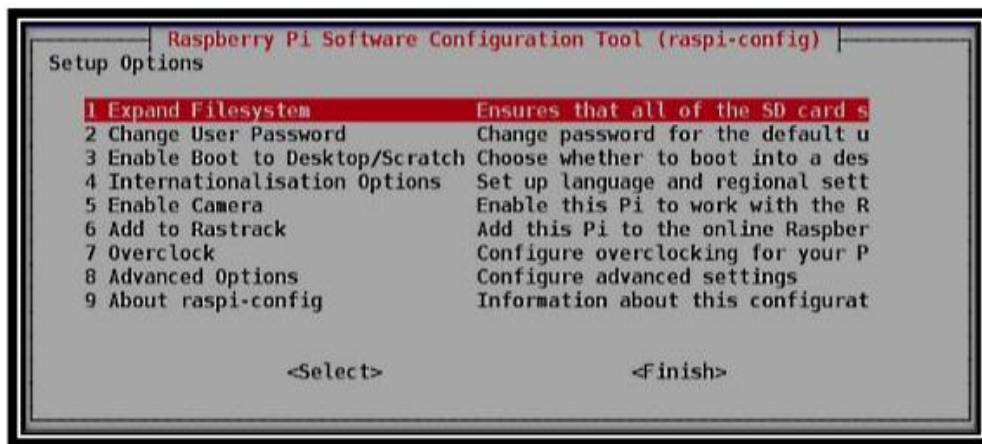


Figure 6.1.1.5: Raspberry Pi Setup Window

Now that the Setup Options window is active, next have to fix one thing. After finishing each of the steps above, if it asks to restart the raspberry pi 2, please follow the instructions. Selecting the first option in the list of the setup window, than select the "Expand File system" option and press enter button. To make this sure just make use of all the space available on the SD card as a complete partition. All this solution is, expanding the operating system to fit the whole space on the memory card which can be used as the storage memory for the raspberry pi 2 and then start the system again.

6.2 Installation of Honeyd on Raspberry Pi 2

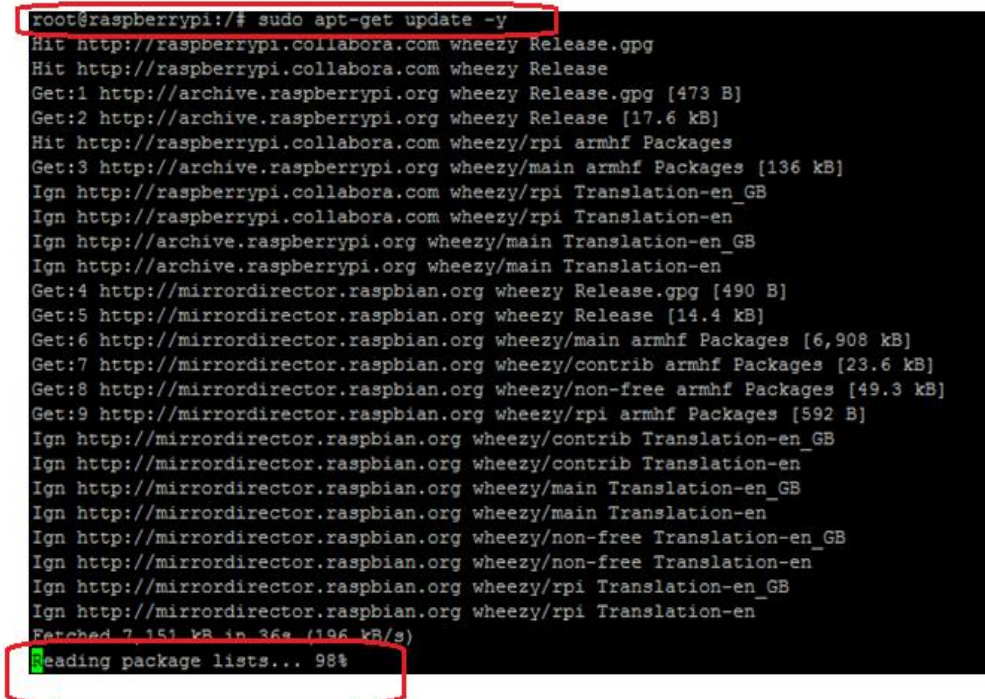
By default, the initial user is not a **superuser**, unlike other linux distributions, where the user is a **superuser** by default. On the other hand, the user is normally in the sudo-ers group. Thus, one has to be typing a lot of "**sudo**" in front of the following commands. If one receives a message that the user doesn't have the permission, just slap a sudo onto the front of the command and it should work.

After following the instructions mentioned in the above section 6.1.1

The next thing is to start deploying the Honeyd software on the raspberry pi. First the best thing is to do is to run the update on the raspberry pi by typing the following command in the terminal

```
root@raspberrypi:/# sudo apt-get update -y
```

The output of the above command can be seen into the following figure 6.2.1

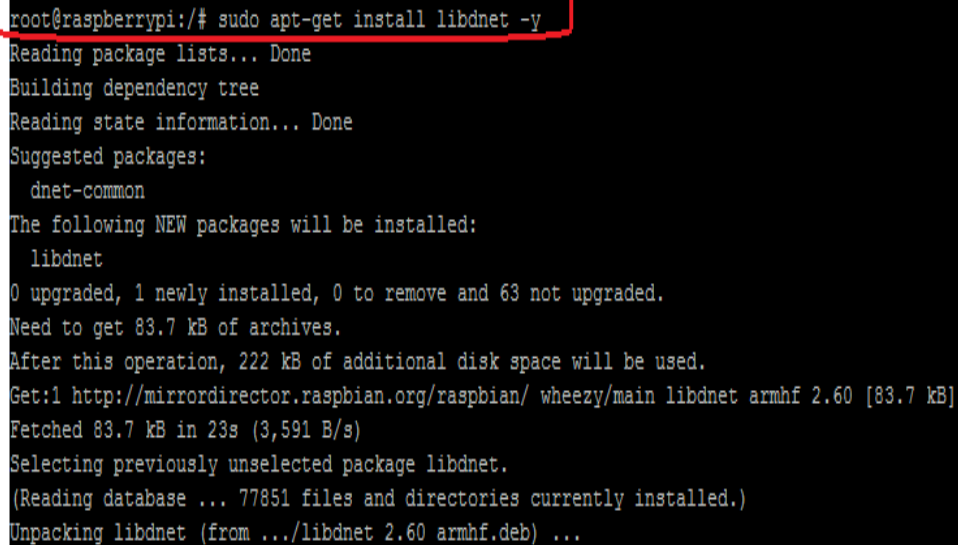


```
root@raspberrypi:/# sudo apt-get update -y
Hit http://raspberrypi.collabora.com wheezy Release.gpg
Hit http://raspberrypi.collabora.com wheezy Release
Get:1 http://archive.raspberrypi.org wheezy Release.gpg [473 B]
Get:2 http://archive.raspberrypi.org wheezy Release [17.6 kB]
Hit http://raspberrypi.collabora.com wheezy/rpi armhf Packages
Get:3 http://archive.raspberrypi.org wheezy/main armhf Packages [136 kB]
Ign http://raspberrypi.collabora.com wheezy/rpi Translation-en_GB
Ign http://archive.raspberrypi.org wheezy/rpi Translation-en
Ign http://archive.raspberrypi.org wheezy/main Translation-en_GB
Ign http://archive.raspberrypi.org wheezy/main Translation-en
Get:4 http://mirrordirector.raspbian.org wheezy Release.gpg [490 B]
Get:5 http://mirrordirector.raspbian.org wheezy Release [14.4 kB]
Get:6 http://mirrordirector.raspbian.org wheezy/main armhf Packages [6,908 kB]
Get:7 http://mirrordirector.raspbian.org wheezy/contrib armhf Packages [23.6 kB]
Get:8 http://mirrordirector.raspbian.org wheezy/non-free armhf Packages [49.3 kB]
Get:9 http://mirrordirector.raspbian.org wheezy/rpi armhf Packages [592 B]
Ign http://mirrordirector.raspbian.org wheezy/contrib Translation-en_GB
Ign http://mirrordirector.raspbian.org wheezy/contrib Translation-en
Ign http://mirrordirector.raspbian.org wheezy/main Translation-en_GB
Ign http://mirrordirector.raspbian.org wheezy/main Translation-en
Ign http://mirrordirector.raspbian.org wheezy/non-free Translation-en_GB
Ign http://mirrordirector.raspbian.org wheezy/non-free Translation-en
Ign http://mirrordirector.raspbian.org wheezy/rpi Translation-en_GB
Ign http://mirrordirector.raspbian.org wheezy/rpi Translation-en
Fetched 7,151 kB in 36s (196 kB/s)
Reading package lists... 98%
```

Figure 6.2.1: Updating the packages in raspberry pi

The next step in installation of the Honeyd is the installation of a pre-requisite for some network activities that Honeyd carries out. This can be seen in the figure 6.2.2 below

```
root@raspberrypi:/# sudo apt-get install libdnf -y
```



```
root@raspberrypi:/# sudo apt-get install libdnf -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  dnf-common
The following NEW packages will be installed:
  libdnf
0 upgraded, 1 newly installed, 0 to remove and 63 not upgraded.
Need to get 83.7 kB of archives.
After this operation, 222 kB of additional disk space will be used.
Get:1 http://mirrordirector.raspbian.org/raspbian/ wheezy/main libdnf armhf 2.60 [83.7 kB]
Fetched 83.7 kB in 23s (3,591 B/s)
Selecting previously unselected package libdnf.
(Reading database ... 77851 files and directories currently installed.)
Unpacking libdnf (from .../libdnf_2.60_armhf.deb) ...
```

Figure 6.2.2: Installing the libdnf package

Libdnf provides a basic and moveable interface to numerous low-level networking routines which includes [15].

- The manipulation of the network addresses
- network filtering
- It provides network manipulation and interface lookup
- The tunneling of IP protocol
- The transmission of IP packets with Ethernet frames

The next step is the installation of the pre-requisite for HoneD itself which can be seen in the following figure 6.2.3

The command used to install the pre-requisite is

```
root@raspberrypi:/# sudo apt-get install libevent-dev libdumbnet-dev
libpcap-dev libpcr3-dev libedit-dev bison flex libtool automake
```

```
root@raspberrypi:/# sudo apt-get install libevent-dev libdumbnet-dev libpcap-dev libpcr3-dev libedit-dev bison flex libtool automake
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  autoconf autotools-dev libbison-dev libbsd-dev libdumbnet1 libevent-core-2.0-5
  libevent-extra-2.0-5 libevent-openssl-2.0-5 libevent-pthreads-2.0-5 libltdl-dev
  libpcap0.8 libpcap0.8-dev libpcr3-dev libtinfo-dev m4
Suggested packages:
  autoconf2.13 autoconf-archive gnu-standards autoconf-doc gettext bison-doc
  libtool-doc automaken gfortran fortran95-compiler gcj
The following NEW packages will be installed:
  autoconf automake autotools-dev bison flex libbison-dev libbsd-dev libdumbnet-dev
  libdumbnet1 libedit-dev libevent-core-2.0-5 libevent-dev libevent-extra-2.0-5
  libevent-openssl-2.0-5 libevent-pthreads-2.0-5 libltdl-dev libpcap-dev libpcap0.8
  libpcap0.8-dev libpcr3-dev libpcr3-dev libtinfo-dev libtool m4
0 upgraded, 24 newly installed, 0 to remove and 1 not upgraded.
Need to get 5,482 kB of archives.
After this operation, 14.1 MB of additional disk space will be used.
Do you want to continue [Y/n]?
```

Figure 6.2.3: Installation of pre-requisite for Honyd

libevent: It is defined as an event notification library [16]

libdumbnet: A dumb, portable networking library. This package contains the static library and the C header files [17]

Libpcap: It provides a transportable structure for low level network monitoring. Libpcap can offer network statistics compilation, monitoring and debugging [18]

Libpcr3-dev: This is a library of functions to carry usual expressions whose grammar and semantics to which the language Perl 5 are alike as possible. [19]

Libedit-dev: BSD library for editing lines and history [20]

The next step in installation is installing the git repository by the following command so that the honeyd source files can be cloned from the git repository

```
root@raspberrypi:/# sudo apt-get install git -y
```

The output of the above command can be seen in the figure 6.2.4 below

```
root@raspberrypi:/# sudo apt-get install git -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
git is already the newest version.
git set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 63 not upgraded.
```

Figure 6.2.4: Installing git

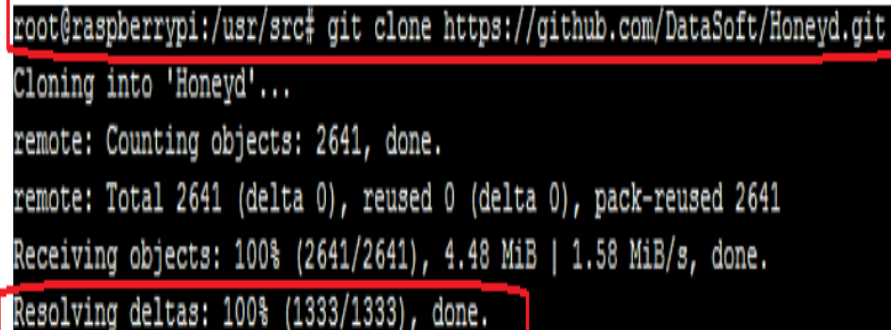
The next step after cloning the source files of the honeyd is to navigate to /usr/src directory

```
root@raspberrypi:/home/pi# ls
mylogs.log  python_games
root@raspberrypi:/home/pi# cd /
root@raspberrypi:/# ls
bin  dev  home  lost+found  mnt  proc  run  selinux  sys  usr
boot  etc  lib  media      opt  root  sbin  srv      tmp  var
root@raspberrypi:/# cd /usr/src/
root@raspberrypi:/usr/src#
```

Figure 6.2.5: Changing the directory

The next step is the step of cloning the Honeyd source files to the local machine by means of Git by typing the subsequent command in the terminal and the output can be seen into the figure 6.2.6 below.

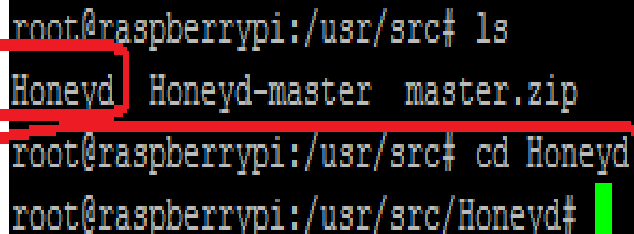

```
sudo git clone https://github.com/DataSoft/Honeyd.git
```



```
root@raspberrypi:/usr/src# git clone https://github.com/DataSoft/Honeyd.git
Cloning into 'Honeyd'...
remote: Counting objects: 2641, done.
remote: Total 2641 (delta 0), reused 0 (delta 0), pack-reused 2641
Receiving objects: 100% (2641/2641), 4.48 MiB | 1.58 MiB/s, done.
Resolving deltas: 100% (1333/1333), done.
```

Figure 6.2.6: Cloning the Honeyd to the local machine .e.i Raspberry Pi

After cloning the git repository to the raspberry pi navigate to the Honeyd directory which can be seen in the figure 6.2.7 below



```
root@raspberrypi:/usr/src# ls
Honeyd  Honeyd-master  master.zip
root@raspberrypi:/usr/src# cd Honeyd
root@raspberrypi:/usr/src/Honeyd#
```

Figure 6.2.7: Navigating to the directory Honeyd

After getting into the Honeyd directory the next command needed to run is as follows

```
root@raspberrypi:/usr/src/Honeyd# sudo ./autogen.sh
```

And the output of the above command can be seen in the figure 6.2.8 below

```
root@raspberrypi:/usr/src/Honeyd# sudo ./autogen.sh
libtoolize: Consider adding 'AC_CONFIG_MACRO_DIR([m4])' to configure.in and
libtoolize: rerunning libtoolize, to keep the correct libtool macros in-tree.
libtoolize: Consider adding '-I m4' to ACLOCAL_AMFLAGS in Makefile.am.
configure.in:5: installing './config.guess'
configure.in:5: installing './config.sub'
configure.in: installing './ylwrap'
root@raspberrypi:/usr/src/Honeyd#
```

Figure 6.2.8: Running the *autogen.sh* script

- **Autogen.sh:** It offer automatic assembling system grounding and is usually useful to projects that use GNU build system [21]

The next step is the configuration of the build parameters by typing the following command into the terminal which can be seen in the Figure 6.2.9 below

```
root@raspberrypi:/usr/src/Honeyd# sudo ./configure
```

```
root@raspberrypi:/usr/src/Honeyd# sudo ./configure
checking build system type... armv7l-unknown-linux-gnueabi
checking host system type... armv7l-unknown-linux-gnueabi
checking target system type... armv7l-unknown-linux-gnueabi
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... no
checking for mawk... mawk
checking whether make sets $(MAKE)... yes
checking whether to enable maintainer-specific portions of Makefiles... no
checking how to print strings... printf
checking for style of include used by make... GNU
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking dependency style of gcc... none
checking for a sed that does not truncate output... /bin/sed
checking for grep that handles long lines and -e... /bin/grep
checking for egrep... /bin/grep -E
checking for fgrep... /bin/grep -F
checking for ld used by gcc... /usr/bin/ld
checking if the linker (/usr/bin/ld) is GNU ld... yes
checking for BSD- or MS-compatible name lister (nm)... /usr/bin/nm -B
checking the name lister (/usr/bin/nm -B) interface...
```

Figure 6.2.9: Configuring the build parameters

Type the following command in the terminal to make the installation directories and the output is shown in the Figure 6.2.10 below

```
root@raspberrypi:/usr/src/Honeyd# sudo make
```

```
root@raspberrypi:/usr/src/Honeyd# sudo make
make all-recursive
make[1]: Entering directory '/usr/src/Honeyd'
Making all in .
make[2]: Entering directory '/usr/src/Honeyd'
gcc -DHAVE_CONFIG_H -I. -I./compat/libdnet -I./compat -I/usr/include -I/usr/include
yd\" -DPATH_HONEYDDATA=\"/usr/share/honeyd\" -DPATH_HONEYDLIB=\"/usr/lib/honeyd
-D_GNU_SOURCE -c honeyd.c
honeyd.c: In function 'handle_udp_packet':
honeyd.c:2569:6: warning: unused variable 'i' [-Wunused-variable]
gcc -DHAVE_CONFIG_H -I. -I./compat/libdnet -I./compat -I/usr/include -I/usr/include
yd\" -DPATH_HONEYDDATA=\"/usr/share/honeyd\" -DPATH_HONEYDLIB=\"/usr/lib/honeyd
-D_GNU_SOURCE -c command.c
gcc -DHAVE_CONFIG_H -I. -I./compat/libdnet -I./compat -I/usr/include -I/usr/include
```

Figure 6.2.10: Screenshot of the make installation

After this step the next step is installing the Honeyd the command needed to be typed in the terminal is as follows

```
root@raspberrypi:/usr/src/Honeyd# sudo make install
```

And the output can be seen in the Figure 6.2.11 below. In the figure below there is a red mark down in the bottom in the screen shot is (honey**d**) this bold **d** is the indication that honeyd software is installed on the local machine. After writing honey press tab and the **d** will automatically appear in the end of the honey command.

```

root@raspberrypi:/usr/src/Honeyd# sudo make install
Making install in .
make[1]: Entering directory '/usr/src/Honeyd'
make[2]: Entering directory '/usr/src/Honeyd'
/bin/mkdir -p '/usr/bin'
/bin/bash ./libtool --mode=install /usr/bin/install -c honeyd honeydctl hone
ydstats hsniff '/usr/bin'
libtool: install: /usr/bin/install -c honeyd /usr/bin/honeyd
libtool: install: /usr/bin/install -c honeydctl /usr/bin/honeydctl
libtool: install: /usr/bin/install -c honeydstats /usr/bin/honeydstats
libtool: install: /usr/bin/install -c hsniff /usr/bin/hsniff
/bin/mkdir -p "/usr/share/honeyd"
(cd . && tar -cf - ./webserver) | \
(cd /usr/share/honeyd && tar -xf -)
find /usr/share/honeyd/webserver -type f | xargs chmod a+r
find /usr/share/honeyd/webserver -type d | xargs chmod a+rx
(cd . && tar -cf - ./scripts) | \
(cd /usr/share/honeyd && tar -xf -)
python /usr/share/honeyd/scripts/lib/init.py
/bin/mkdir -p '/usr/share/honeyd'
/usr/bin/install -c -m 644 README nmap.assoc xprobe2.conf nmap-os-db config.sam
ple config.ethernet pf.os nmap-mac-prefixes '/usr/share/honeyd'
/bin/mkdir -p '/usr/share/honeyd'
/bin/mkdir -p '/usr/lib/honeyd'
/usr/bin/install -c -m 644 libhoneyd.so '/usr/lib/honeyd'
/bin/mkdir -p '/usr/share/man/man1'
/usr/bin/install -c -m 644 honeydctl.1 '/usr/share/man/man1'
/bin/mkdir -p '/usr/share/man/man8'
/usr/bin/install -c -m 644 honeyd.8 '/usr/share/man/man8'
/bin/mkdir -p '/usr/include/honeyd'
/usr/bin/install -c -m 644 hooks.h plugins.h plugins_config.h debug.h '/usr/inc
lude/honeyd'
make[2]: Leaving directory '/usr/src/Honeyd'
make[1]: Leaving directory '/usr/src/Honeyd'
root@raspberrypi:/usr/src/Honeyd# honeyd

```

Figure 6.2.11: Installation of Honeyd software

At this point one can be able to run HoneyD very easily and firmly. Furthermore to get it up and running type the following command in the terminal of the raspberry pi

```
root@raspberrypi:/usr/src/Honeyd# sudo honeyd -i eth0
```

This command will start the honeyd as a daemon listening on device's eth0 interface, by means of uncovered default settings.

The output of this command can be seen in the Figure 6.2.12 below

```
root@raspberrypi:/usr/src/Honeyd# sudo honeyd -i eth0
Honeyd v1.6d Copyright (c) 2002-2007 Niels Provos
honeyd[2282]: started with -i eth0
honeyd[2282]: listening promiscuously on eth0: (arp or
ip proto 47 or (udp and src port 67 and dst port 68)
or (ip )) and not ether src b8:27:eb:83:7a:38
Honeyd starting as background process
root@raspberrypi:/usr/src/Honeyd#
```

Figure 6.2.12: Honeyd starting as background process

The final point where one can have fully running Honeyd software is adding up a new command as the last above command will not get the emulated machines and networks noticed alone. For this purpose a package named 'farpd' is used for the purpose of directing the ARP requests to the HoneyD daemon.

```
root@raspberrypi:/usr/src/Honeyd# sudo apt-get install farpd
```

```
root@raspberrypi:/usr/src/Honeyd# sudo apt-get install farpd
Reading package lists... Done
Building dependency tree
Reading state information... Done
farpd is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 62 not upgrade
root@raspberrypi:/usr/src/Honeyd#
```

Figure 6.2.13: Installation of 'farpd'

- **Farpd**- Fake ARP user space daemon [22]

6.2.1 Setting up the fake virtual windows machine using HoneyD

For this setup the needed machines are one windows machine and one raspberry pi which are using raspbian wheezy distribution of linux. Raspberry pi is the machine that is running honeyd.

The idea can be seen in the Figure 6.2.1.1 below

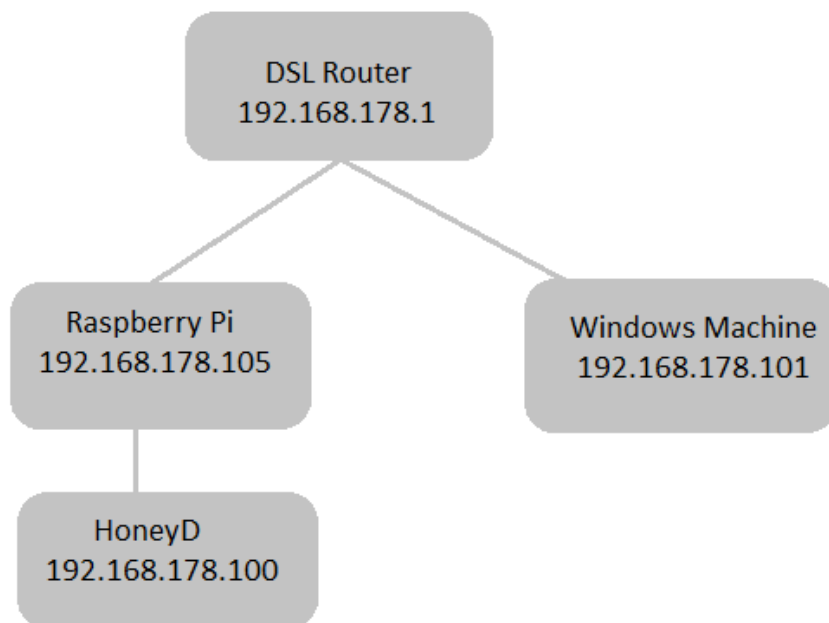


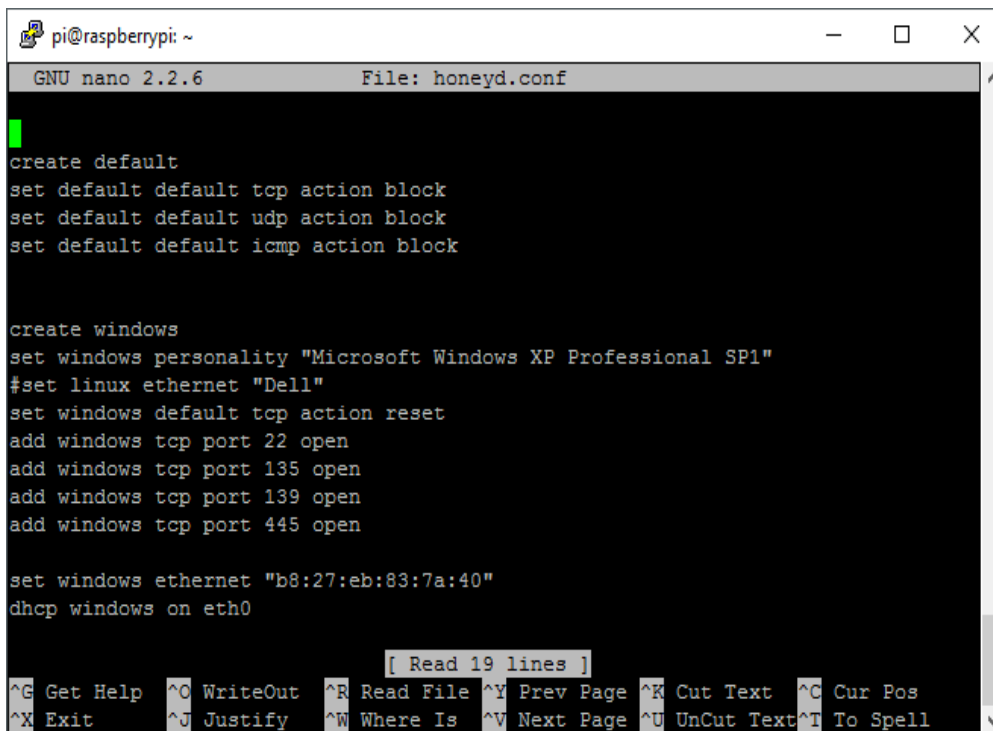
Figure 6.2.1.1: Conceptual setup of the HoneyD Implementation

After the successful installation of honeyd on raspberry pi the next thing that is needed to do is to create a configuration file. A honeyd configuration file is

the heart of the honeypot. The configuration file guides honeyd which type of operating system to emulate, which ports to open and which type of services should be executed etc. This configuration file can be changed to emulate all sorts of setups but for right now let's focus on the conceptual setup which is a simple setup and get that up and running.

Below in figure 6.2.1.2 is the configuration file used to set up the virtual machine

This configuration file can be created by any of the editor of choice here nano is used as a text editor to create this configuration file.



```
GNU nano 2.2.6      File: honeyd.conf

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

create windows
set windows personality "Microsoft Windows XP Professional SP1"
#set linux ethernet "Dell"
set windows default tcp action reset
add windows tcp port 22 open
add windows tcp port 135 open
add windows tcp port 139 open
add windows tcp port 445 open

set windows ethernet "b8:27:eb:83:7a:40"
dhcp windows on eth0

[ Read 19 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

Figure 6.2.1.2: Configuration file screen shot

The “**create default**” is a statement which informs the honeyd to drop traffic unless and until it is defined in the configuration file. These lines are needed when one let the honeypot to acquire an IP address via dhcp. Every time one look “**create**” within the configuration file means creating a template for a honeypot, so one can create as many honeypots as it is essential within the **honeyd.conf** file. For this windows template let's define some important

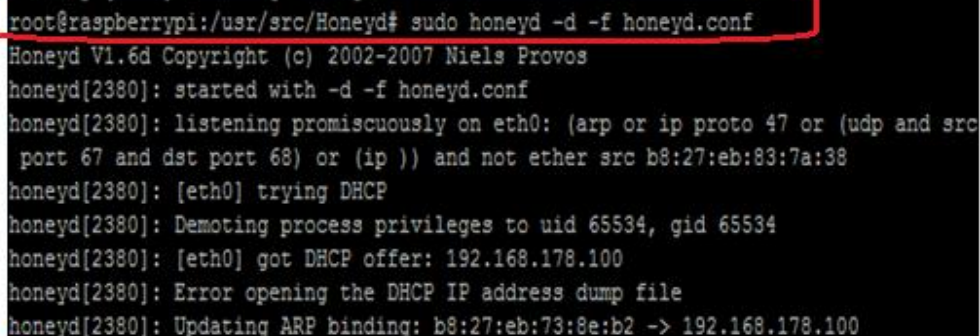
things. First it is needed to set the “**personality**”. Personality means when another device on the network connects to this honeypot it will emerge to be a **Windows XP Pro SP1** machine. This is done by emulating the network stack fingerprints. The windows template contains four ports (22,135, 139, and 445) which are kept open. These are common ports that are open on a windows system, other than SSH port 22.

This “**action reset**” word plays a fundamental role in this configuration file as it will not take traffic in account if it is not targeted at the open ports mentioned in this configuration file. Here the “**set windows ethernet**” command will set the MAC address for the virtual honeypot host. This will be necessary if one is running the honeypot by means of dhcp. One can plainly make up any MAC address of their choice, it is best to keep it nearly similar to the physical MAC address of the linux machine on which the honeypot is running.

At the end “**dhcp**” declaration instructs the windows template to obtain an IP address from dhcp. Subsequent to the honeyd.config configuration file is properly setup, the next thing is to launch the honeyd, below is the command that is used when initially getting honeyd up and running.

Furthermore it is very important to launch the honeyd with the logging **-l** flag so that all the connections that are made from the attacker on the honeypot can be logged and stored in the log folder. This will be shown and discussed in later section.

```
root@raspberrypi:/usr/src/Honeyd# sudo honeyd -d -f honeyd.conf
```



```
root@raspberrypi:/usr/src/Honeyd# sudo honeyd -d -f honeyd.conf
Honeyd V1.6d Copyright (c) 2002-2007 Niels Provos
honeyd[2380]: started with -d -f honeyd.conf
honeyd[2380]: listening promiscuously on eth0: (arp or ip proto 47 or (udp and src
port 67 and dst port 68) or (ip )) and not ether src b8:27:eb:83:7a:38
honeyd[2380]: [eth0] trying DHCP
honeyd[2380]: Demoting process privileges to uid 65534, gid 65534
honeyd[2380]: [eth0] got DHCP offer: 192.168.178.100
honeyd[2380]: Error opening the DHCP IP address dump file
honeyd[2380]: Updating ARP binding: b8:27:eb:73:8e:b2 -> 192.168.178.100
```

Figure 6.2.1.3: Launching screen shot of HoneyD

In the above command the **-d** is used so that it should not run in the background (or should not run as a daemon process in Linux terminology). This will allow for more verbose output so that it can be troubleshoot when needed.

Running in this mode will also show the IP that was given to the honeypot via dhcp.

```
honeyd[2380]: Updating ARP binding: b8:27:eb:73:8e:b2 -> 192.168.178.100
honeyd[2380]: Sending ICMP Echo Reply: 192.168.178.100 -> 192.168.178.101
honeyd[2380]: arp_send: who-has 192.168.178.101 tell 192.168.178.100
honeyd[2380]: arp_rcv_cb: 192.168.178.101 at 70:f1:a1:c8:71:fb
honeyd[2380]: Sending ICMP Echo Reply: 192.168.178.100 -> 192.168.178.101
honeyd[2380]: Sending ICMP Echo Reply: 192.168.178.100 -> 192.168.178.101
honeyd[2380]: Sending ICMP Echo Reply: 192.168.178.100 -> 192.168.178.101
honeyd[2380]: arp_reply 192.168.178.100 is-at b8:27:eb:73:8e:b2
honeyd[2380]: arp_rcv_cb: 192.168.178.101 at 70:f1:a1:c8:71:fb
```

Figure 6.2.1.4: Ping output from the Windows machine

In the figure 6.2.1.4 above it can be clearly seen that dhcp allowed the honeypot the address of **192.168.178.100**. The connectivity can be tested from the windows machine **192.168.178.101** by ping the honeypot.

6.2.2 Scanning the functionality of HoneyD by performing Nmap Foot printing

At this point the honeyd is successfully deployed and tested. Let's perform a scan through the well known scanning tool NMAP [10]. This will prove in checking the availability of the fake system that is created by the Honeyd software.

This test of scanning is performed by the separate Linux machine running on the same network.

```
root@raspberrypi:/usr/src/Honeyd# sudo honeyd -d -f honeyd.conf -l  
/home/pi/mylogs.log  
Honeyd V1.6d Copyright (c) 2002-2007 Niels Provos  
honeyd[2346]: started with -d -f honeyd.conf -l /home/pi/mylogs.lo  
g  
honeyd[2346]: listening promiscuously on eth0: (arp or ip proto 47  
or (udp and src port 67 and dst port 68) or (ip )) and not ether  
src b8:27:eb:83:7a:38  
honeyd[2346]: [eth0] trying DHCP  
honeyd[2346]: Demoting process privileges to uid 65534, gid 65534  
honeyd[2346]: [eth0] got DHCP offer: 192.168.178.104  
honeyd[2346]: Error opening the DHCP IP address dump file  
honeyd[2346]: Updating ARP binding: b8:27:eb:6b:c1:60 -> 192.168.1  
78.104
```

Figure 6.2.2.1: Screen shot of Honeyd launching with `-l` flag

In the above figure 6.2.2.1 it can be clearly seen that one additional line of command is used that is the `-l` flag to stored the logs of the connection made to the honeypot and can be stored on the separate folder “**logs.log**”

```
root@kali:/# nmap -v --open -O 192.168.178.100
Starting Nmap 7.00 ( https://nmap.org ) at 2016-02-21 14:01 CET
Initiating ARP Ping Scan at 14:01
Scanning 192.168.178.100 [1 port]
Completed ARP Ping Scan at 14:01, 0.01s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 14:01
Completed Parallel DNS resolution of 1 host. at 14:01, 0.00s elapsed
Initiating SYN Stealth Scan at 14:01
Scanning 192.168.178.100 [1000 ports]
Discovered open port 22/tcp on 192.168.178.100
Discovered open port 139/tcp on 192.168.178.100
Discovered open port 135/tcp on 192.168.178.100
Discovered open port 445/tcp on 192.168.178.100
Completed SYN Stealth Scan at 14:01, 1.81s elapsed (1000 total ports)
Initiating OS detection (try #1) against 192.168.178.100
Nmap scan report for 192.168.178.100
Host is up (0.027s latency).
Not shown: 996 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
135/tcp    open  msrpc
139/tcp    open  netbios-ssn
445/tcp    open  microsoft-ds
MAC Address: B8:27:EB:8E:08:53 (Raspberry Pi Foundation)
Device type: general purpose
Running: Microsoft Windows XP
OS CPE: cpe:/o:microsoft:windows_xp::sp1:professional
OS details: Microsoft Windows XP Professional SP1
```

Figure 6.2.2.2: Nmap scanned output

In the figure 6.2.2.2 above the results that are presented are from the network scanning application Nmap. In the above screen shot it can be clearly seen that after scanning on the IP address **192.168.178.100** which is the IP address of the HoneyD software and the fake virtual system deployed on the honeyD is Windows XP professional SP1 with the common services and ports opened.

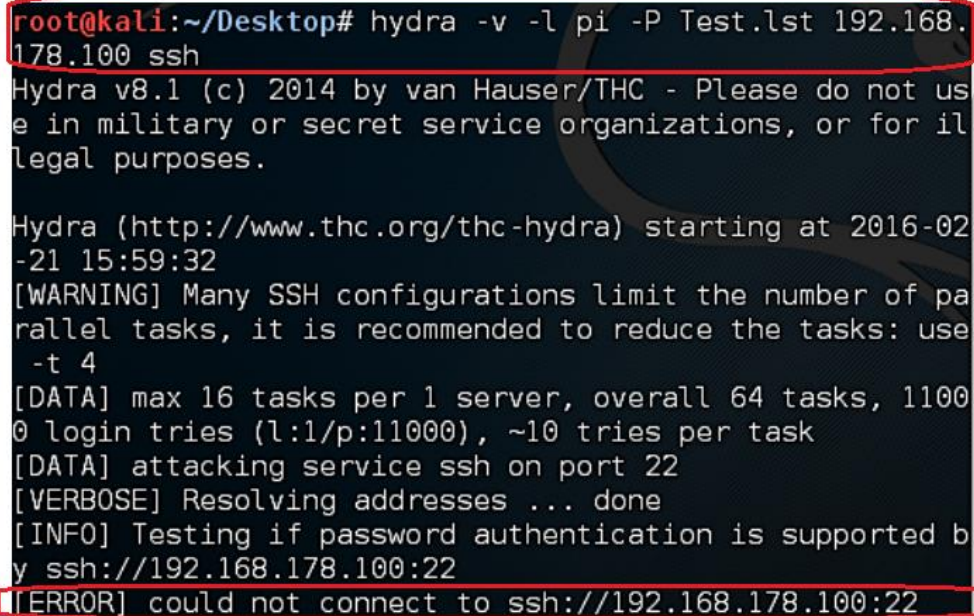
6.2.3 Testing the functionality of HoneyD via Brute Force Attack using Hydra

In this part of the report let's perform a brute force attack using hydra tool on port 22 to test the functionality of the virtual host created on HoneyD and analysing the logs created by HoneyD.

This test is performed on a separate Linux machine running on the same network.

In the terminal of the Linux machine acting as an attacker machine the command needed to perform the test is as follows

```
root@kali:/#hydra -v -l -pi -P Test.lst 192.168.178.100 ssh
```



```
root@kali:~/Desktop# hydra -v -l pi -P Test.lst 192.168.178.100 ssh
Hydra v8.1 (c) 2014 by van Hauser/THC - Please do not use in military or secret service organizations, or for illegal purposes.

Hydra (http://www.thc.org/thc-hydra) starting at 2016-02-21 15:59:32
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t 4
[DATA] max 16 tasks per 1 server, overall 64 tasks, 11000 login tries (l:l/p:11000), ~10 tries per task
[DATA] attacking service ssh on port 22
[VERBOSE] Resolving addresses ... done
[INFO] Testing if password authentication is supported by ssh://192.168.178.100:22
[ERROR] could not connect to ssh://192.168.178.100:22
```

Figure 6.2.3.1: Screenshot of the Attack

As it can be seen in the Figure 6.2.3.1 above that after detecting the open ports on the fake host using Nmap tool the attack has been performed using Hydra on the IP address **192.168.178.100** on port **22** which is the SSH port. As the system created on HoneyD was a fake system it has no real existence it's just a trap made to fool the attacker and made his efforts and time useless who tries to attack this fake system.

The statement in the red marking is the proof that this system is nothing but a fake host and the attack was unsuccessful and could not connect through SSH on the destined IP address **192.168.178.100**.

In the Figure 6.2.3.2 below the logs which were maintained by HoneyD software can be analysed that were generated using the -l flag at the time of HoneyD launching.

```
root@raspberrypi:/home/pi# cat logs.log
2016-02-21-13:35:49.5595 honeyd log started -----
2016-02-21-13:35:57.9756 tcp(6) S 192.168.178.103 43466 192.168.178.100 22 [Linux 2.2 20-25]
2016-02-21-13:36:05.4614 igmp(2) - 224.0.0.1 192.168.178.1: 32
2016-02-21-13:36:06.2106 igmp(2) - 224.0.0.251 192.168.178.103: 32 [Linux 2.2 20-25]
2016-02-21-13:36:06.7856 igmp(2) - 224.0.0.253 192.168.178.101: 32
2016-02-21-13:36:08.2799 tcp(6) E 192.168.178.103 43466 192.168.178.100 22: 0 0
2016-02-21-13:36:12.2826 igmp(2) - 224.0.0.252 192.168.178.101: 32
```

Figure 6.2.3.2: Logs created by HoneyD

In the screenshot above the logs are saved in **logs.log** folder under the home directory.

After using the **cat** command the logs are displayed. These logs clearly show that at which date and time the **TCP** connection for the attack was made. The attacker source **IP 192.168.178.103** , the attacker random source port 43466 , the destination IP address **192.168.178.100** which is the honeyD IP address and the port destination port 22 which was mainly targeted by the attacker.

With these information the logs displays the type of the system which was used by the attacker in this case it's a Linux 2.2 20-24 system. The **S** and **E** parameters are presenting the establishment of the tries and the cessation of the tries. After realising these logs the network administrator could block or mark it as black list IP address in the firewall of the network to prevent the network assets to be attacked or probed for the next time from this IP address.

6.3 Installation of Kippo Honeypot on Raspberry Pi 2

In this section let's look into the installation from the scratch to a fully running honeypot to actually record the attacks. The honeypot that will be implemented is kippo a straightforward SSH honeypot developed in Python with quite a lot of promises. The setup applies to Raspberry Pi 2 but the process on any other Ubuntu or Debian-based system will be the identical.

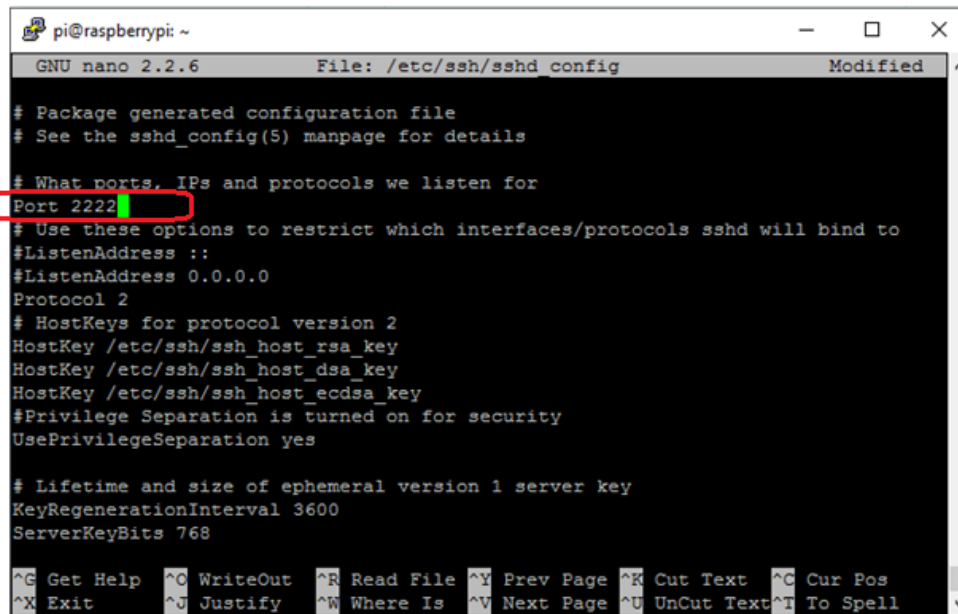
The first step required to start deploying the honeypot is the same as in the previous section 6.2 to make the SSH connection through Putty by getting the IP address **192.168.178.102** from the DSL router's Dhcp Client List.

After the update and upgrading of the newly installed raspberry pi system lets start the main code of lines to bring the kippo honeypot in running mode.

Kippo by default "listens" on port **2222**, which is all right for testing causes, but this particularly reduces the probability to trace any attacks, because the common computerized network tools that are used by attacker targets the default SSH port **22**. Therefore, it is good to allow Kippo listen on port **22**. For this one must first alter the port normally the SSH server has, which is **22**. To be able to connect back properly with the system port should be set to **2222**. So let's change the configuration file of the kippo SSH server by simply typing the following command in the terminal of raspberry pi.

```
root@raspberry pi:/# sudo nano /etc/ssh/sshd_config
```

The changed configuration file can be seen in the figure 6.3.1 below



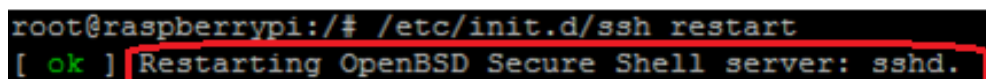
```
pi@raspberrypi: ~  
GNU nano 2.2.6 File: /etc/ssh/sshd_config Modified  
  
# Package generated configuration file  
# See the sshd_config(5) manpage for details  
  
# What ports, IPs and protocols we listen for  
Port 2222  
# Use these options to restrict which interfaces/protocols sshd will bind to  
#ListenAddress ::  
#ListenAddress 0.0.0.0  
Protocol 2  
# HostKeys for protocol version 2  
HostKey /etc/ssh/ssh_host_rsa_key  
HostKey /etc/ssh/ssh_host_dsa_key  
HostKey /etc/ssh/ssh_host_ecdsa_key  
#Privilege Separation is turned on for security  
UsePrivilegeSeparation yes  
  
# Lifetime and size of ephemeral version 1 server key  
KeyRegenerationInterval 3600  
ServerKeyBits 768  
  
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos  
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

Figure 6.3.1: Snapshot of the configuration file

In the above screen shot the port **22** is changed to port **2222** so that the SSH port 22 will stay free to attract the attacker.

After doing so the next command needed to restart the server is as follows and the output can be seen in the figure 6.3.2 below.

```
root@raspberrypi:/# sudo /etc/init.d/ssh restart
```



```
root@raspberrypi:/# /etc/init.d/ssh restart  
[ ok ] Restarting OpenBSD Secure Shell server: sshd.
```

Figure 6.3.2: Restarting the server

After running this command disconnects the server first and then connects it again through putty on the new port which can be seen in the figure 6.3.3 below.

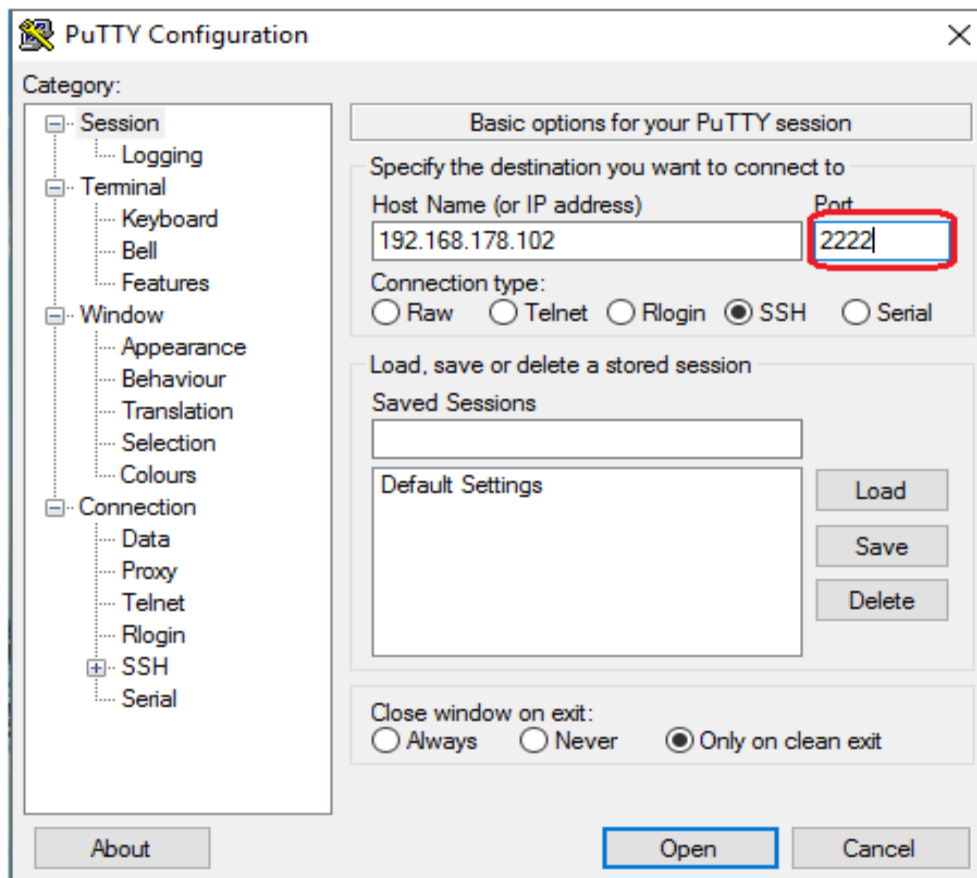


Figure 6.3.3: Snapshot with the new port

The next step required is the installation of necessary software packages for kippo server.

For this type the following command in the terminal window

```
root@raspberrypi:~# sudo apt-get install python-dev openssl python-  
openssl python-pyasn1 python-twisted
```

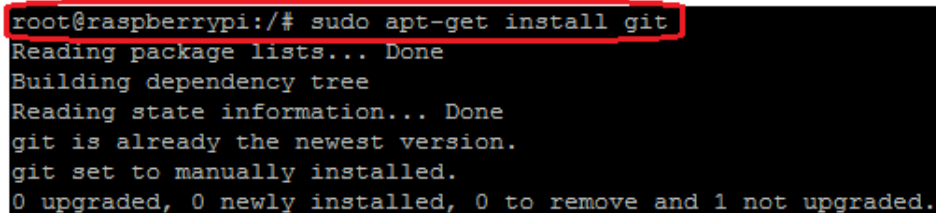
- **python-dev:** Header files and a static library for python. [23]
- **openssl:** It's a general purpose cryptography library that generates RSA keys and provides the implementation of the open SSL and TLS. [24]
- **python-openssl:** A python covering for the open ssl library. [25]

- **python-pyasn1:** This is an implementation of ASN.1 types and codec in Python programming language.[26]
- **python-twisted:** A special structure for python language with specially emphasized on event driven network programming. [27]

The next point includes the downloading of kippo for this here git is used to download it and this is done with the following command.

```
root@raspberrypi:# sudo apt-get install git
```

The output can be seen in the figure 6.3.4 below



```
root@raspberrypi:/# sudo apt-get install git
Reading package lists... Done
Building dependency tree
Reading state information... Done
git is already the newest version.
git set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 1 not upgraded.
```

Figure 6.3.4: Screenshot for installing git

There is one more issue in using port 22 which is a problem and it is that in Linux just the root user is granted to use the well known ports which are less than 1024 and one should not run Kippo as a root for the sake of security. To execute the honeypot on port 22 the easiest way is via the utility **authbind**.

To use the **authbind** utility one must to install it using the apt-get repository by typing the following command.

```
root@raspberrypi:/# sudo apt-get install authbind
```

The output screen shot can be seen in the figure 6.3.5 below:

```
root@raspberrypi:/# sudo apt-get install authbind
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  authbind
0 upgraded, 1 newly installed, 0 to remove and 1 not upgraded.
Need to get 18.6 kB of archives.
After this operation, 97.3 kB of additional disk space will be used.
Get:1 http://mirrordirector.raspbian.org/raspbian/ wheezy/main authbind armhf 2.1.1
[18.6 kB]
Fetched 18.6 kB in 0s (76.4 kB/s)
Selecting previously unselected package authbind.
Reading database ... 80%
```

Figure 6.3.5: Screen shot of the authbind

Before moving further it is important to create a non-root user to made kippo run by typing the following command

```
root@raspberrypi:/# sudo adduser kippo
```

The output can be seen in the figure 6.3.6 below.

```
root@raspberrypi:/# sudo adduser kippo
Adding user `kippo' ...
Adding new group `kippo' (1002) ...
Adding new user `kippo' (1001) with group `kippo' ...
Creating home directory `/home/kippo' ...
Copying files from `/etc/skel' ...
Enter new UNIX password: █
```

Figure 6.3.6: Screen shot for adding the new user

For creating the new non-root user it asks about the new UNIX password which is **123456**. The password is made very easy so that the system can be compromise easily as it is a decoy system.

In the next figure 6.3.7 the server asks about the basic information for creating the new user which should be left as default value by just pressing the Enter button.

```
root@raspberrypi:/# sudo adduser kippo
Adding user `kippo' ...
Adding new group `kippo' (1002) ...
Adding new user `kippo' (1001) with group `kippo' ...
Creating home directory `/home/kippo' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for kippo
Enter the new value, or press ENTER for the default
    Full Name []:
    Room Number []:
    Work Phone []:
    Home Phone []:
    Other []:
Is the information correct? [Y/n]
```

Figure 6.3.7: Screen shot of the user values

The next thing that should be done is adding this new user in the list of users that has access to use Sudo. Type the following command in the terminal

```
root@raspberrypi:/# sudo nano visudo
```

This above line of command will open the file where it is needed to add the **kippo** ALL= (ALL: ALL) ALL to have **sudo** access to the new user. In figure 6.3.8 it can be seen.

```
pi@raspberrypi: ~  
GNU nano 2.2.6 File: /etc/sudoers.tmp  
#  
# This file MUST be edited with the 'visudo' command as root.  
#  
# Please consider adding local content in /etc/sudoers.d/ instead of  
# directly modifying this file.  
#  
# See the man page for details on how to write a sudoers file.  
#  
Defaults        env_reset  
Defaults        mail_badpass  
Defaults        secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"  
  
# Host alias specification  
  
# User alias specification  
  
# Cmnd alias specification  
  
# User privilege specification  
root    ALL=(ALL:ALL) ALL  
  
# Allow members of group sudo to execute any command  
%sudo    ALL=(ALL:ALL) ALL  
  
# See sudoers(5) for more information on "#include" directives:  
  
#includedir /etc/sudoers.d  
pi ALL=(ALL) NOPASSWD: ALL  
kippo ALL=(ALL:ALL) ALL
```

Figure 6.3.8: Screenshot for sudo user

After the typing the above lines under the root user one is needed to finish the steps by providing the commands below.

```
root@raspberrypi pi: /# touch /etc/authbind/byport/22  
root@raspberrypi pi: /# chown kippo:kippo /etc/authbind/byport/22  
root@raspberrypi pi: /# chmod 777 /etc/authbind/byport/22
```

At this point it's important to move into as kippo user and navigate to **/home** directory and clone the **Git** which can be seen in the figure 6.3.9 below.

```
kippo@raspberrypi ~ $ sudo git clone https://github.com/desaster/kippo.git

We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.

[sudo] password for kippo:
Cloning into 'kippo'...
remote: Counting objects: 1540, done.
remote: Total 1540 (delta 0), reused 0 (delta 0), pack-reused 1540
Receiving objects: 100% (1540/1540), 2.64 MiB | 1.53 MiB/s, done.
Resolving deltas: 100% (928/928), done.
```

Figure 6.3.9: Screenshot for git cloning

After cloning it is important to copy the **kippo.cfg.dist** to **kippo.cfg** file by typing the following command in to terminal and then open the **kippo.cfg** file in **nano** editor and change the port from **2222** to **22**.

```
kippo@raspberrypi ~ $ sudo nano kippo.cfg
```

The following figure 6.3.10 will be appeared on the terminal screen in which the ports has to be changed with the hostname to **File Server** form the default **svr03** to attract the attacker.

```
kippo@raspberrypi: ~/kippo
GNU nano 2.2.6 File: kippo.cfg

#
# Kippo configuration file (kippo.cfg)
#

[honeypot]

# IP addresses to listen for incoming SSH connections.
#
# (default: 0.0.0.0) = any address
#ssh_addr = 0.0.0.0

# Port to listen for incoming SSH connections.
#
# (default: 2222)
ssh_port = 22

# Hostname for the honeypot. Displayed by the shell prompt of the virtual
# environment.
#
# (default: svr03)
hostname = File Server

# Directory where to save log files in.
#
```

Figure 6.3.10: Screen shot of kippo.cfg file

At last it is needed to edit the kippo start script by typing the command

```
kippo@raspberrypi ~$ sudo nano start.sh
```

To change the lines marked in red in the figure 6.3.11 below. In this just put the hash sign (#) in the start of the above command and add the next command.

```

kippo@raspberrypi: ~/kippo
GNU nano 2.2.6                                     File: start.sh

#!/bin/sh

set -e

cd $(dirname $0)

if [ "$1" != "" ]
then
    VENV="$1"

    if [ ! -d "$VENV" ]
    then
        echo "The specified virtualenv \"$VENV\" was not found!"
        exit 1
    fi

    if [ ! -f "$VENV/bin/activate" ]
    then
        echo "The specified virtualenv \"$VENV\" was not found!"
        exit 2
    fi

    echo "Activating virtualenv \"$VENV\""
    . $VENV/bin/activate
fi

twistd --version

echo "Starting kippo in the background..."
#twistd -y kippo.tac -l log/kippo.log --pidfile kippo.pid
authbind --deep twistd -y kippo.tac -l log/kippo.log --pidfile kippo.pid

```

Figure 6.3.11: Screenshot of start.sh

Let's check that kippo is running on port 22 or 2222 by typing the following command into the terminal and the output can be seen in figure 6.3.12.

```

kippo@raspberrypi ~ /kippo$ sudo netstat -antp
kippo@raspberrypi ~/kippo $ sudo netstat -antp
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/
Program name
tcp        0      0 0.0.0.0:2222            0.0.0.0:*               LISTEN      8272
/sshd
tcp        0      0 192.168.178.103:2222    192.168.178.101:61727   ESTABLISHED 8297
/sshd: pi [priv

```

Figure 6.3.12: Screenshot of netstat output

As it can be seen in the screenshot above that there is no line found in which kippo server is running on the port 22. For this it is important to run the kippo server on port 22 by typing the following command into the terminal

```
kippo@raspberrypi ~ /kippo$ ./start.sh
```

```
kippo@raspberrypi ~/kippo $ ./start.sh
twistd (the Twisted daemon) 12.0.0
Copyright (c) 2001-2012 Twisted Matrix Laboratories.
See LICENSE for details
Starting kippo in the background...
```

Figure 6.3.13: Kippo running in the background

After running the kippo server lets one again check the status using netstat –antp command in the figure 6.3.14 below in which it is clearly seen that the python script which is actually kippo server is listening on port 22.

```
kippo@raspberrypi ~/kippo $ sudo netstat -antp
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:2222            0.0.0.0:*               LISTEN      2142/sshd
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      2332/python
tcp        0      0 192.168.178.102:2222    192.168.178.101:50047   ESTABLISHED 2279/sshd: pi [priv
```

Figure 6.3.14: Screenshot of netstat output

6.3.1 Integrating MySQL Database with Kippo

At this point kippo can log all the attacks into the **Log** directory and can show all the attacks session on the terminal.

To see the logs in a more controlled way it's significant to store the logs in the databases. For this here MySql database is used to see the logs into proper format and for easiness. For installing MySql server into kippo the following command should be run as root user.

```
root@raspberrypi: /# apt-get install python-mysqldb mysql-server
root@raspberrypi:/# apt-get install python-mysqldb mysql-server
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  heirloom-mailx libaio1 libdbd-mysql-perl libdbi-perl libhtml-template-perl
  libmysqlclient16 libmysqlclient18 mysql-client-5.5 mysql-common mysql-server-5.5
  mysql-server-core-5.5
Suggested packages:
  exim4 mail-transport-agent libipc-sharedcache-perl libterm-readkey-perl tinycat
  python-egenix-mxdatetime python-mysqldb-dbg
Recommended packages:
  mailx
The following NEW packages will be installed:
  heirloom-mailx libaio1 libdbd-mysql-perl libdbi-perl libhtml-template-perl
  libmysqlclient16 libmysqlclient18 mysql-client-5.5 mysql-common mysql-server
  mysql-server-5.5 mysql-server-core-5.5 python-mysqldb
0 upgraded, 13 newly installed, 0 to remove and 1 not upgraded.
Need to get 7,275 kB/10.5 MB of archives.
After this operation, 94.3 MB of additional disk space will be used.
Do you want to continue [Y/n]? y
```

Figure 6.3.1.1: Screenshot for installing MySQL

After the above step the window for Mysql database password for root user will be appeared on the terminal screen which can be seen in the next figure 6.3.1.2 below.

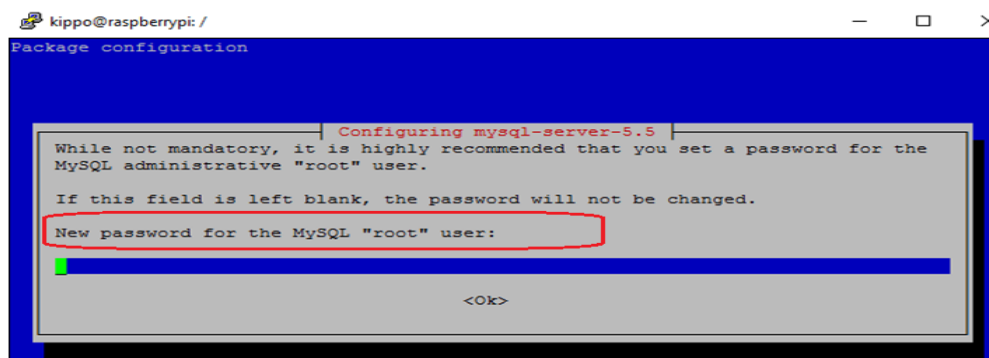


Figure 6.3.1.2: Screenshot for MySQL root user

The password set for this is **kippo123456789** although the password must be set a bit complex.

After installing and setting up the password the next step is to enter into the mysql and create the database under the name **kippo** by typing the following command

```
root@raspberrypi: ~/mysql -u root -p
root@raspberrypi:~/mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 47
Server version: 5.5.47-0+deb7u1 (Debian)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> CREATE DATABASE kippo;
Query OK, 1 row affected (0.00 sec)

mysql> GRANT ALL ON kippo.* TO 'kippo'@'localhost' IDENTIFIED BY 'kippo123456789';
Query OK, 0 rows affected (0.00 sec)
```

Figure 6.3.1.3: Screenshot for creating database for Kippo

The database needs to be created only once, but one must select it for use each time one begin a mysql session. One can do this by issuing a **USE** statement. Then integrate it with the source file **mysql.sql** so that it can extract the tables and fields for the kippo database from the script **mysql.sql** automatically, but first it should be made executable before integrating it. The command needed is to be executed under the kippo user under the **/home/kippo/kippo** directory.

```
kippo@raspberrypi ~/kippo $ mysql -u kippo -p
mysql> USE kippo;
mysql> source ./dot/sql/mysql.sql
mysql> exit
```

The output can be seen into the figure 6.3.1.4 below.

```
kippo@raspberrypi ~/kippo $ mysql -u kippo -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 59
Server version: 5.5.47-0+deb7u1 (Debian)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> USE kippo;
Database changed
mysql> source ./doc/sql/mysql.sql;
Query OK, 0 rows affected (0.03 sec)

Query OK, 0 rows affected (0.03 sec)

Query OK, 0 rows affected (0.03 sec)

Query OK, 0 rows affected (0.04 sec)

Query OK, 0 rows affected (0.02 sec)

Query OK, 0 rows affected (0.09 sec)

Query OK, 0 rows affected (0.03 sec)

mysql> exit
Bye
```

Figure 6.3.1.4: Screenshot of selecting kippo as a database

At this stage re-login as kippo user into the system, but first it is important kill the kippo in order to change the configurations and start it again. This can be achieve by running the following command into the terminal

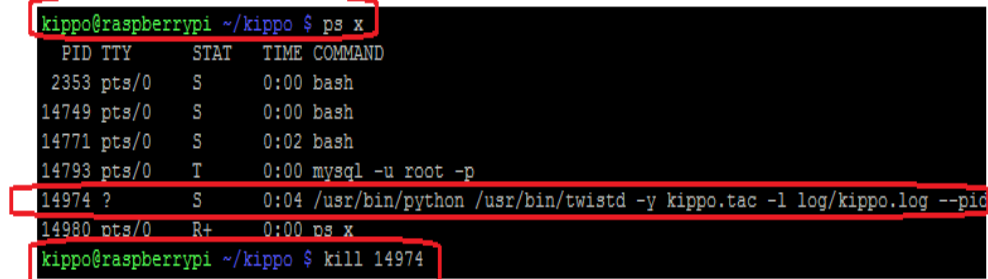
```
kippo@raspberrypi ~/kippo $ ps x
```

Look for a line like this:

```
14974 ? S 0:04 /usr/bin/python /usr/bin/python/twisted -y kippo.tac -1
log/kippo.log --pid
```

In the above line the first column is the process ID and this ID is important to kill the process. This can be seen done by typing the command in the terminal and can be seen in the figure 6.3.1.5 below.

```
kippo@raspberrypi ~/kippo $ Kill 14974
```



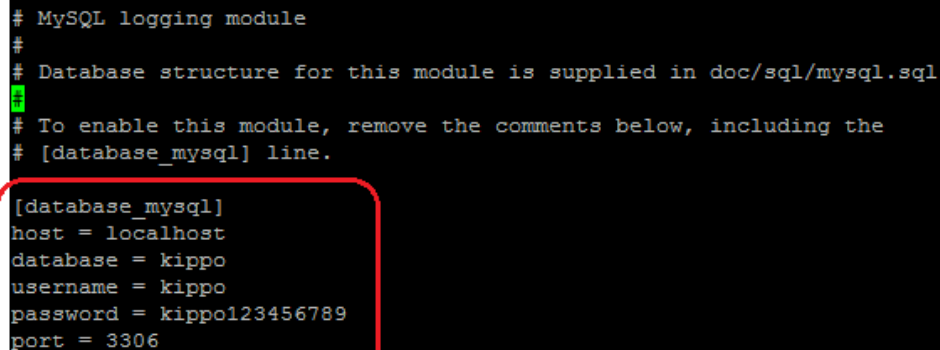
```
kippo@raspberrypi ~/kippo $ ps x
  PID TTY          STAT       TIME COMMAND
  2353 pts/0        S          0:00 bash
 14749 pts/0        S          0:00 bash
 14771 pts/0        S          0:02 bash
 14793 pts/0        T          0:00 mysql -u root -p
 14974 ?            S          0:04 /usr/bin/python /usr/bin/twisted -y kippo.tac -l log/kippo.log --pid
 14980 pts/0        R+         0:00 ps x
kippo@raspberrypi ~/kippo $ kill 14974
```

Figure 6.3.1.5: Screenshot of killing kippo process

The next step is the step of configuration of the database into the **kippo.cfg** file by the following command into the terminal.

```
kippo@raspberrypi ~/kippo $ sudo nano kippo.cfg
```

This will open the window in which the following changes are needed to run the MySQL database which can be seen in the figure 6.3.1.6 below.



```
# MySQL logging module
#
# Database structure for this module is supplied in doc/sql/mysql.sql
#
# To enable this module, remove the comments below, including the
# [database_mysql] line.

[database_mysql]
host = localhost
database = kippo
username = kippo
password = kippo123456789
port = 3306
```

Figure 6.3.1.6: Screenshot of Database configuration

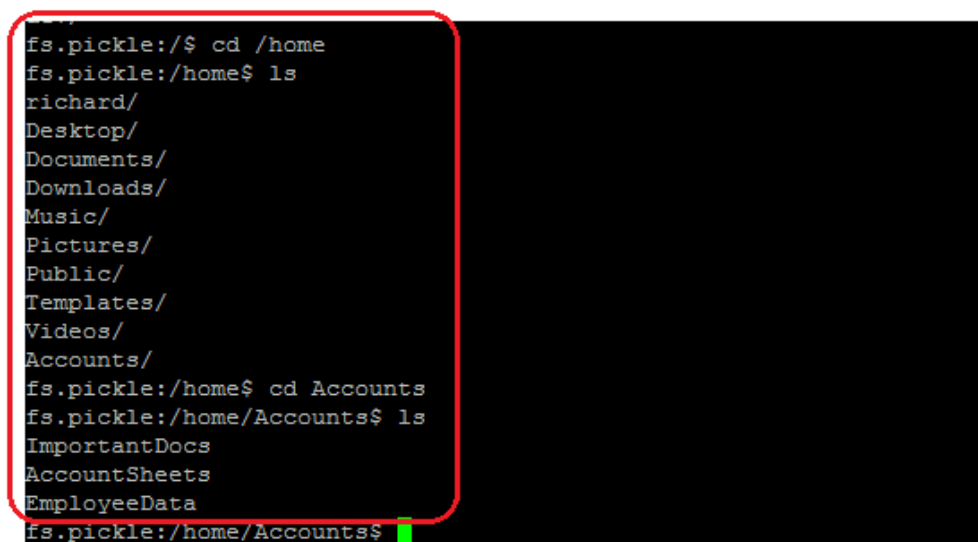
6.3.2 Creating the Fake File System using fs.pickle

Here it is important to setup the fake file system facility provided by the Kippo honeypot. One can create a full fake file system that will attract the attacker and to fool him that he is successful and he got the access to the real production system of the company.

For this type the following command into the terminal to get into the **fs.pickle** directory of the kippo server.

```
kippo@raspberrypi~/kippo/utls$ sudo /home/kippo/kippo/utls/fsctl.py  
/home/kippo/kippo/fs.pickle
```

This will open the new window where one can make a complete fake file system this can be seen in the figure 6.3.2.1 below.



```
fs.pickle:/$ cd /home  
fs.pickle:/home$ ls  
richard/  
Desktop/  
Documents/  
Downloads/  
Music/  
Pictures/  
Public/  
Templates/  
Videos/  
Accounts/  
fs.pickle:/home$ cd Accounts  
fs.pickle:/home/Accounts$ ls  
ImportantDocs  
AccountSheets  
EmployeeData  
fs.pickle:/home/Accounts$
```

Figure 6.3.2.1: Screenshot of the fake file system in fs.pickle

All these above directories shown in the figure 6.3.2.1 are created manually and they are nothing but the fake directories that has no values. The attacker will see these directories after entering into the system and can navigate in between them can harm them by deleting.

With this functionality of kippo , kippo provides its own fake directories that are also to fool the attackers who can **cat** different directories and can delete them like **passwd** under the **/etc** and **/proc** directories.

In addition kippo supports a facility to add new users and their passwords through which the intruder or attacker can get the access into the fake kippo server. To do this navigate to **data** directory under the **/kippo** user and then navigate to **userdb.txt** file.

This can be done by typing the following command into the terminal

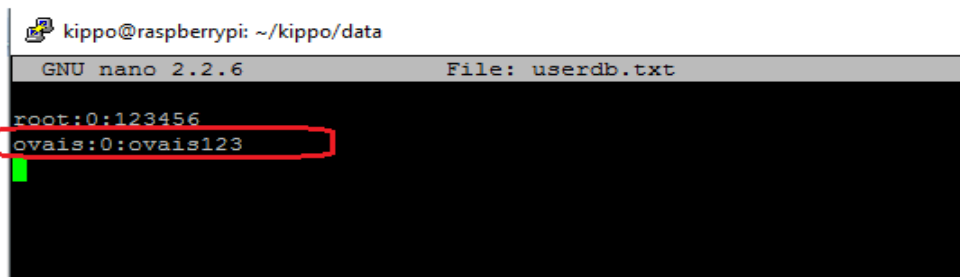
```
kippo@raspberrypi ~/kippo/data $sudo nano userdb.txt
```

Here let's create a new user and password for the kippo server.

Here the user is: **ovais**

And the password is: **ovais123**

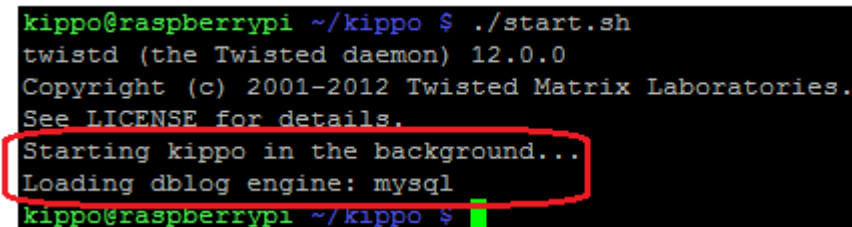
It is shown in the figure 6.3.2.2 below.



```
kippo@raspberrypi: ~/kippo/data
GNU nano 2.2.6 File: userdb.txt
root:0:123456
ovais:0:ovais123
```

Figure 6.3.2.2: Screenshot for new user and password

After creating the new user and configuring the **kippo.cfg** file for the database settings let's start the kippo server again in which the mysql server will be also started that is shown in the next figure 6.3.2.3 below.



```
kippo@raspberrypi ~/kippo $ ./start.sh
twisted (the Twisted daemon) 12.0.0
Copyright (c) 2001-2012 Twisted Matrix Laboratories.
See LICENSE for details.
Starting kippo in the background...
Loading dblog engine: mysql
kippo@raspberrypi ~/kippo $
```

Figure 6.3.2.3: Screenshot for loading mysql and starting kippo

6.3.3 Testing the functionality of Kippo via Brute Force Attack using Medusa from Attacker Machine

In the figure 6.3.3.1 the conceptual scenario is shown of the attack. The Kippo server is running on raspberry Pi 2 on the IP address **192.168.178.102** and listening on port **22**. The attacker machine is a also another Linux machine who's IP address is **192.168.178.103** and the DSL router is acting as a gateway for both of the machines in the LAN.

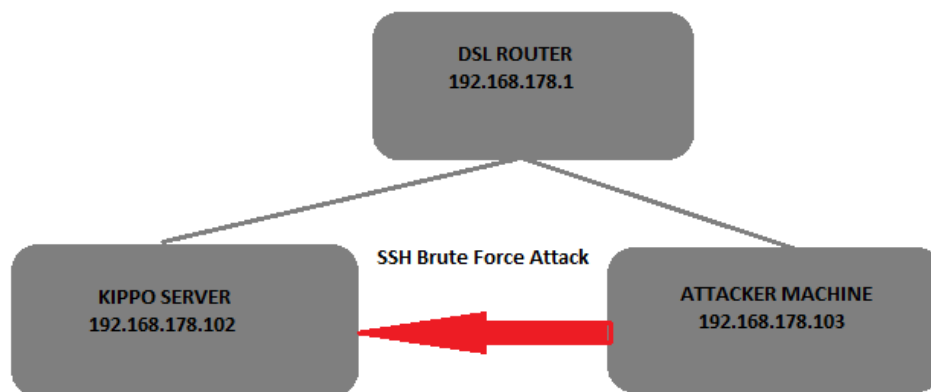


Figure 6.3.3: Conceptual diagram of brute force attack on kippo

In this test the first thing needs to do is to scan the network from a separate Linux machine for the available systems and the open ports on the same LAN. As here it is known that the IP address on which Kippo server is running is **192.168.178.102** with some open ports. The scanning will be performed through the Nmap tool. The command needed to run this foot printing is as follows and the result can be seen in the figure 6.3.3.1 below.

```
root@kali:~# nmap -S -P0 -A -v -p 22, 80, 3302, 21 192.168.178.102
```



```
root@kali:~# nmap -sS -P0 -A -v -p 22,80,3302,21 192.168.178.102
Warning: The -P0 option is deprecated. Please use -Pn

Starting Nmap 7.00 ( https://nmap.org ) at 2016-02-24 23:12 CET
NSE: Loaded 132 scripts for scanning.
NSE: Script Pre-scanning.
Initiating NSE at 23:12
Completed NSE at 23:12, 0.00s elapsed
Initiating NSE at 23:12
Completed NSE at 23:12, 0.00s elapsed
Initiating ARP Ping Scan at 23:12
Scanning 192.168.178.102 [1 port]
Completed ARP Ping Scan at 23:12, 0.01s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 23:12
Completed Parallel DNS resolution of 1 host. at 23:12, 0.07s elapsed
Initiating SYN Stealth Scan at 23:12
Scanning 192.168.178.102 [4 ports]
Discovered open port 22/tcp on 192.168.178.102
Completed SYN Stealth Scan at 23:12, 0.01s elapsed (4 total ports)
Initiating Service scan at 23:12
Scanning 1 service on 192.168.178.102
Completed Service scan at 23:12, 0.02s elapsed (1 service on 1 host)
Initiating OS detection (try #1) against 192.168.178.102
NSE: Script scanning 192.168.178.102.
Initiating NSE at 23:12
Completed NSE at 23:12, 0.87s elapsed
Initiating NSE at 23:12
Completed NSE at 23:12, 0.00s elapsed
Nmap scan report for 192.168.178.102
Host is up (0.0040s latency).
PORT      STATE SERVICE VERSION
21/tcp    closed ftp
22/tcp    open  ssh      OpenSSH 5.1p1 Debian 5 (protocol 2.0)
| ssh-hostkey:
|_ 1024 86:26:79:86:08:80:40:17:f2:b6:be:04:e5:98:da:18 (DSA)
|_ 2048 7f:21:60:56:9c:c4:ff:bb:44:df:a7:09:90:03:ff:1f (RSA)
80/tcp    closed http
3302/tcp  closed unknown
```

Figure 6.3.3.1: Screenshot of Nmap foot printing

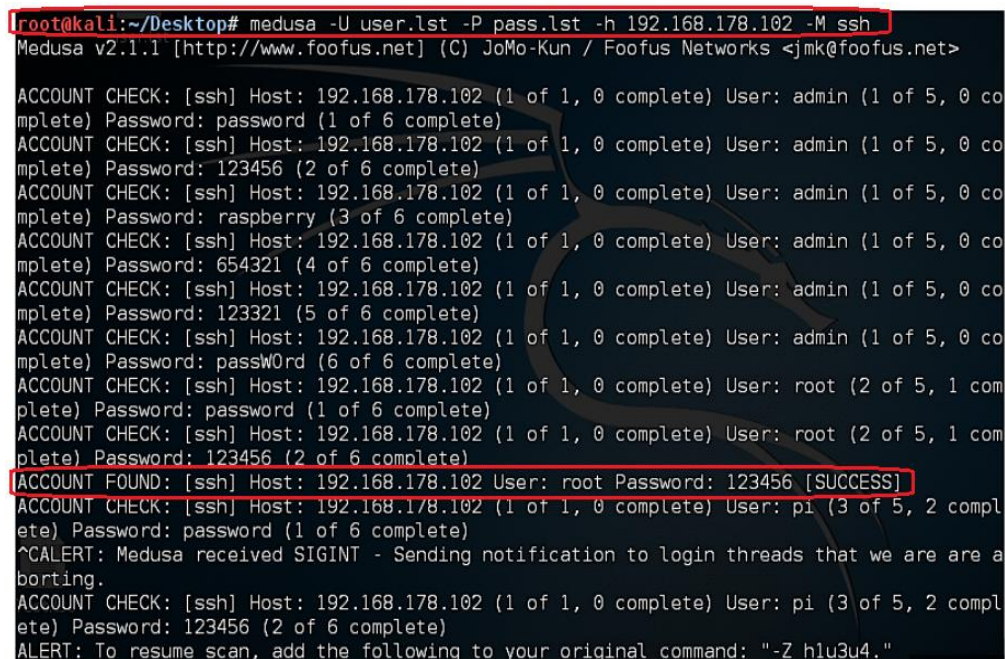
In the above figure 6.3.3.1 it is clearly seen that the port **22** is open which is the SSH port on which the Kippo server is running.

After scanning the kippo server lets perform an attack from the Linux machine having IP address **192.168.178.103**. This attack has been performed by one of the hacking tool **Medusa** often used by hackers to gain the access of SSH servers through brute forcing.

The command used to hack the kippo server needs some parameters to hack the server which includes the target IP address which is **192.168.178.102**, the **user.lst**, **pass.lst** and the protocol.

The command needed to perform the hack is as follows.

```
root@kali:~/Desktop# medusa -U user.lst -P pass.lst -h 192.168.178.102 -M ssh
```



```
root@kali:~/Desktop# medusa -U user.lst -P pass.lst -h 192.168.178.102 -M ssh
Medusa v2.1.1 [http://www.foofus.net] (C) JoMo-Kun / Foofus Networks <jmk@foofus.net>

ACCOUNT CHECK: [ssh] Host: 192.168.178.102 (1 of 1, 0 complete) User: admin (1 of 5, 0 complete) Password: password (1 of 6 complete)
ACCOUNT CHECK: [ssh] Host: 192.168.178.102 (1 of 1, 0 complete) User: admin (1 of 5, 0 complete) Password: 123456 (2 of 6 complete)
ACCOUNT CHECK: [ssh] Host: 192.168.178.102 (1 of 1, 0 complete) User: admin (1 of 5, 0 complete) Password: raspberry (3 of 6 complete)
ACCOUNT CHECK: [ssh] Host: 192.168.178.102 (1 of 1, 0 complete) User: admin (1 of 5, 0 complete) Password: 654321 (4 of 6 complete)
ACCOUNT CHECK: [ssh] Host: 192.168.178.102 (1 of 1, 0 complete) User: admin (1 of 5, 0 complete) Password: 123321 (5 of 6 complete)
ACCOUNT CHECK: [ssh] Host: 192.168.178.102 (1 of 1, 0 complete) User: admin (1 of 5, 0 complete) Password: passW0rd (6 of 6 complete)
ACCOUNT CHECK: [ssh] Host: 192.168.178.102 (1 of 1, 0 complete) User: root (2 of 5, 1 complete) Password: password (1 of 6 complete)
ACCOUNT CHECK: [ssh] Host: 192.168.178.102 (1 of 1, 0 complete) User: root (2 of 5, 1 complete) Password: 123456 (2 of 6 complete)
ACCOUNT FOUND: [ssh] Host: 192.168.178.102 User: root Password: 123456 [SUCCESS]
ACCOUNT CHECK: [ssh] Host: 192.168.178.102 (1 of 1, 0 complete) User: pi (3 of 5, 2 complete) Password: password (1 of 6 complete)
^CALERT: Medusa received SIGINT - Sending notification to login threads that we are aborting.
ACCOUNT CHECK: [ssh] Host: 192.168.178.102 (1 of 1, 0 complete) User: pi (3 of 5, 2 complete) Password: 123456 (2 of 6 complete)
ALERT: To resume scan, add the following to your original command: "-Z h1u3u4."
```

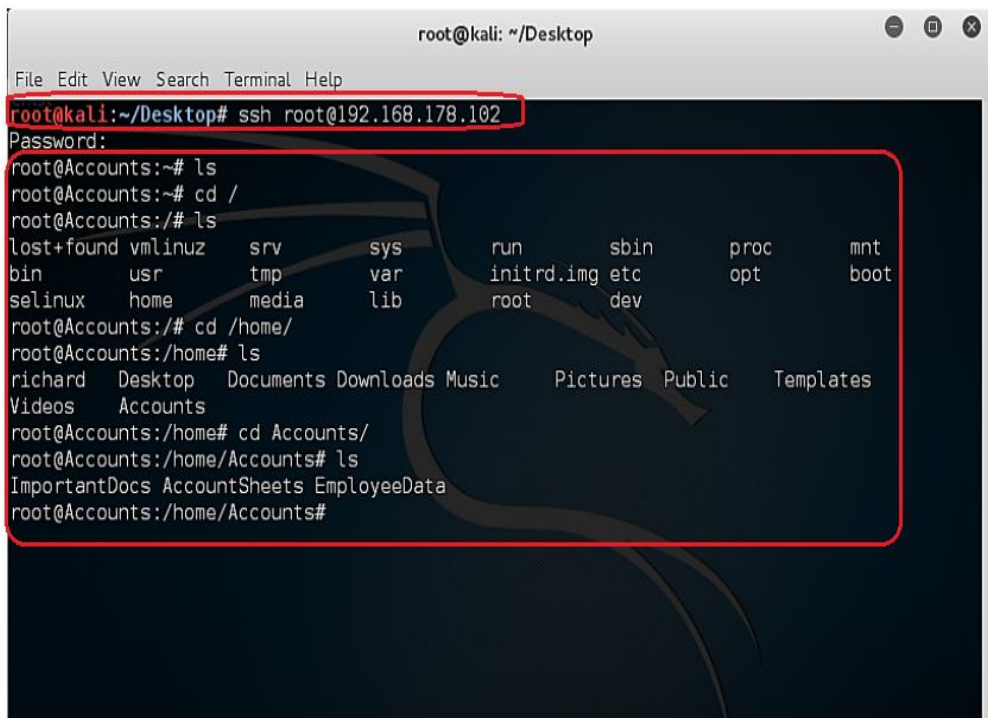
Figure 6.3.3.2: Screenshot of the attack from Linux machine

In the above figure 6.3.3.2 it is seen that after some unsuccessful tries of the attack the attacker got success in finding the correct username and the password of the kippo server. The default username of the kippo server is **root** and the default password set was **123456**.

In the next part the attacker will get into the fake kippo server by providing the correct username **root** and password **123456** found by the successful brute force attack.

The command needed to get into the system is as follows

```
root@kali~:/Desktop# ssh root@192.168.178.102
```



```
root@kali: ~/Desktop
File Edit View Search Terminal Help
root@kali:~/Desktop# ssh root@192.168.178.102
Password:
root@Accounts:~# ls
root@Accounts:~# cd /
root@Accounts:/# ls
lost+found vmlinuz  srv      sys      run      sbin     proc     mnt
bin         usr      tmp      var      initrd.img etc      opt      boot
selinux    home     media    lib      root     dev
root@Accounts:/# cd /home/
root@Accounts:/home# ls
richard  Desktop  Documents Downloads Music    Pictures Public  Templates
Videos  Accounts
root@Accounts:/home# cd Accounts/
root@Accounts:/home/Accounts# ls
ImportantDocs AccountSheets EmployeeData
root@Accounts:/home/Accounts#
```

Figure 6.3.3.3: Screenshot of taking the access of kippo server

In the above figure 6.3.3.3 it is shown that the attacker is entered into the kippo server. Here the attacker or intruder finds some useful directories which is actually the fake file system. The intruder will navigate between different directories thinking that it's a real server but it is a fake system to fool the attacker. The attacker can delete any of the files and directories but they will be not deleted as they don't have the real existence.

The attacker can **cat** the **passwd** and the result that will be available to the attacker which is a fake file generated by kippo's **honeyfs** directory is shown in the figure 6.3.3.4 below.

```
root@Accounts:/etc# cat passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
sshd:x:101:65534::/var/run/sshd:/usr/sbin/nologin
richard:x:1000:1000:Richard Texas,,,:/home/richard:/bin/bash
root@Accounts:/etc#
```

Figure 6.3.3.4: Screenshot of fake file system

6.3.4 Logging Feature of Kippo Server

The logs that are generated by the kippo are stored under the **logs** directory of the kippo server. For a better understanding of the logs, previously MySQL database was integrated with kippo server. To access the logs through MySQL the command needed is as follows.

```
kippo@raspberrypi ~/kippo $ mysql -u kippo -p
```

Where this enters into mysql terminal, to access the database under the name of kippo the following command is needed

```
mysql > USE kippo;
```

Use is a MySQL query used to change to the database selected in this case it is kippo. This can be seen in the figure 6.3.4.1 below.


```
kippo@raspberrypi ~/kippo $ mysql -u kippo -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 63
Server version: 5.5.47-0+deb7u1 (Debian)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> USE kippo;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
```

Figure 6.3.4.1: Screenshot of accessing the MySQL database

After accessing the kippo data base lets select the field **sessions** to show the attacker IP address which is **192.168.178.103** the start time of the attack session and the end time of the attack session. This is also presented in figure 6.3.4.2 below.

This is done by using the MySQL query as follows

```
mysql > SELECT * FROM sessions;
```

```
mysql> SELECT * FROM sessions;
```

id	starttime	endtime	sensor	ip	termsize	client
ceb9d5f4db1711e59155b827eba48398	2016-02-24 16:58:20	2016-02-24 16:58:37	1	192.168.178.103	NULL	1
e3e3cc5adb1711e59155b827eba48398	2016-02-24 16:58:55	2016-02-24 16:59:14	1	192.168.178.103	NULL	1
e7b11dd2daf511e5ae38b827eba48398	2016-02-24 12:55:39	2016-02-24 15:40:03	1	192.168.178.103	79x20	1
f96b7348db1711e59155b827eba48398	2016-02-24 16:59:32	NULL	1	192.168.178.103	79x20	1

4 rows in set (0.00 sec)

Figure 6.3.4.2: Screenshot of the sessions log created

The next thing to look into the logs is the **auth** field in the kippo database, this field of log will help to see the attacker hacking activity means which successful usernames and password he tried and at which time stamp.

This is done by using the MySQL query as follows

```
mysql > SELECT * FROM auth;
```

The output of this field can be seen in the figure 6.3.4.3 below.

```
mysql> SELECT * FROM auth;
```

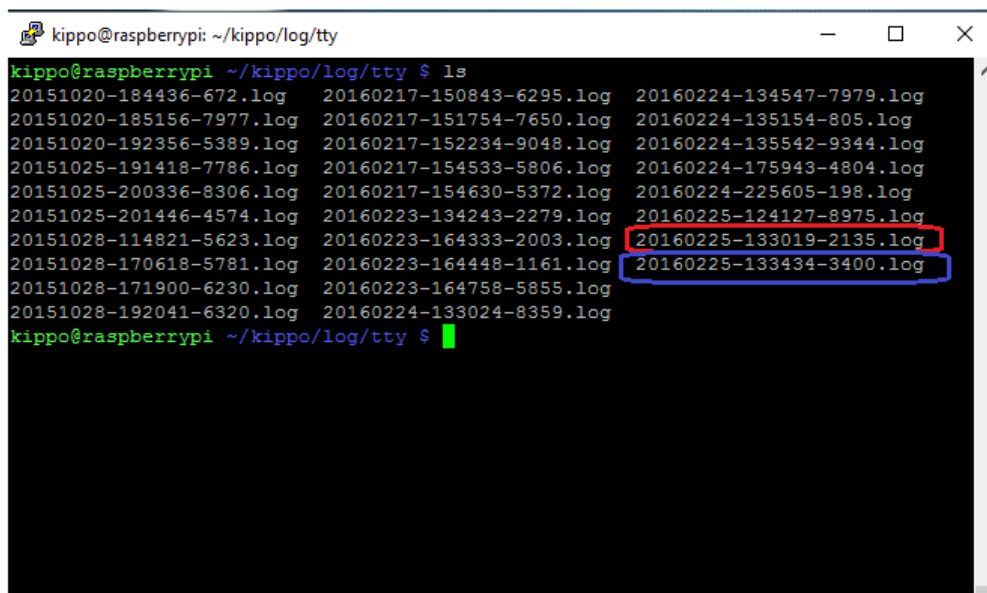
id	session	success	username	password	timestamp
1	e7b11dd2daf511e5ae38b827eba48398	1	root	123456	2016-02-24 12:55:42
2	ceb9d5f4db1711e59155b827eba48398	0	admin	hfyenmjdk	2016-02-24 16:58:24
3	ceb9d5f4db1711e59155b827eba48398	0	admin	445567h	2016-02-24 16:58:32
4	e3e3cc5adb1711e59155b827eba48398	0	pi	qwasqgs	2016-02-24 16:58:59
5	e3e3cc5adb1711e59155b827eba48398	0	pi	qwsQRe232e32	2016-02-24 16:59:00
6	e3e3cc5adb1711e59155b827eba48398	0	pi	sxqd213323	2016-02-24 16:59:01
7	e3e3cc5adb1711e59155b827eba48398	0	pi	sasasxsax	2016-02-24 16:59:04
8	e3e3cc5adb1711e59155b827eba48398	0	pi	saxaqdxas	2016-02-24 16:59:10
9	f96b7348db1711e59155b827eba48398	0	root	12sdfeg	2016-02-24 16:59:37
10	f96b7348db1711e59155b827eba48398	0	root	nhk8ui0	2016-02-24 16:59:41
11	f96b7348db1711e59155b827eba48398	1	root	123456	2016-02-24 16:59:43

11 rows in set (0.00 sec)

Figure 6.3.4.3: Screenshot of auth log created

The last thing that is very important to mention about kippo is its real-time sessions replay feature. Every time when the attacker gets into the server and look around into the system, whatever he does all his activities are recorded as a movie even a single word typing of the command is recorded which can be played to analyse the activities performed by the attacker on the kippo ssh server.

The logs of the replay sessions are stored under the `/log/tty/` directory



```
kippo@raspberrypi: ~/kippo/log/tty
kippo@raspberrypi ~/kippo/log/tty $ ls
20151020-184436-672.log      20160217-150843-6295.log    20160224-134547-7979.log
20151020-185156-7977.log    20160217-151754-7650.log    20160224-135154-805.log
20151020-192356-5389.log    20160217-152234-9048.log    20160224-135542-9344.log
20151025-191418-7786.log    20160217-154533-5806.log    20160224-175943-4804.log
20151025-200336-8306.log    20160217-154630-5372.log    20160224-225605-198.log
20151025-201446-4574.log    20160223-134243-2279.log    20160225-124127-8975.log
20151028-114821-5623.log    20160223-164333-2003.log    20160225-133019-2135.log
20151028-170618-5781.log    20160223-164448-1161.log    20160225-133434-3400.log
20151028-171900-6230.log    20160223-164758-5855.log
20151028-192041-6320.log    20160224-133024-8359.log
kippo@raspberrypi ~/kippo/log/tty $
```

Figure 6.3.4.4: Screenshot for replay logs of the attacker sessions

In the above figure 6.3.4.4 there are two logs that are marked with different colours. The log marked with red colour is the log in which the attacker navigates around the fake system and deleted some of the directories in order to damage the system as he thought it's a real system, but with the new session again the new attacker finds the fake system is still on its position with the same fake file system and directories

The log marked in blue colour is the fresh session made by the attacker and where he founds that nothing has been deleted everything is back there as it was there before deleting and damaging.

To see the replay of the attacker session type the following command into the terminal

```
kippo@raspberrypi~/kippo/log/tty$ /home/kippo/kippo/utils/playlog.py  
20160225-133434-3400.log
```

The above command will play the replay of the attacker session on the kippo terminal window and one can see the exact activities what the attacker did into the fake server step by step.

6.4 Installation of Glastopf Honeypot on Raspberry Pi 2

The initial steps are the same as in the above sections 6.2 and 6.3 for setting up the pi and taking the remote connection via putty manager. The remote connection process on the IP address **192.168.178.102** can be seen in the figure 6.4.1 below.

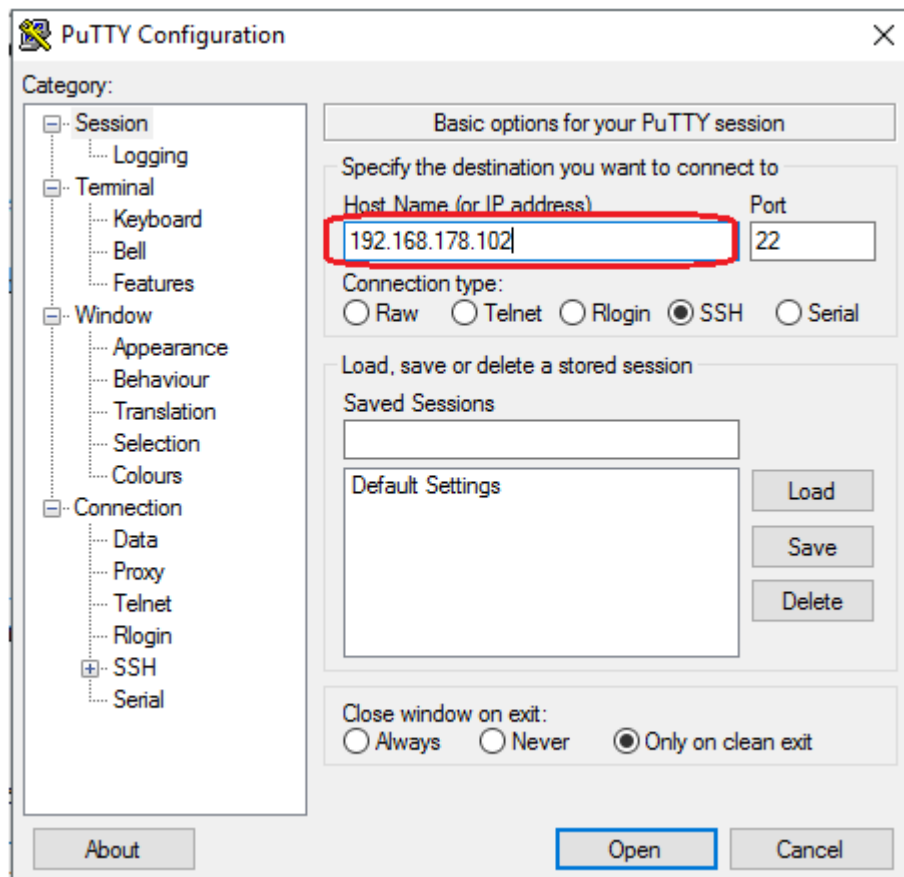


Figure 6.4.1: Screenshot of putty manager

The login and the password is same as by default comes with raspberry pi 2.

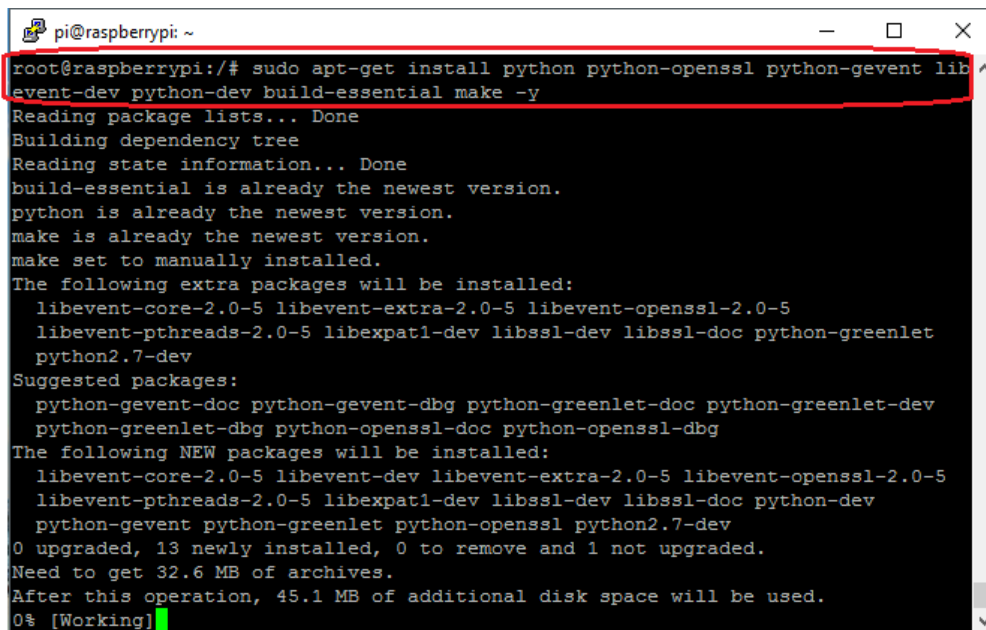
After setting up the raspberry pi the first step in the deployment of glastopf is installing the dependencies of the glastopf which are raspberry pi platform dependent.

```
root@raspberrypi:/# sudo apt-get install python python-openssl  
python-gevent libevent-dev build-essential make
```

- **python:** python language required for glastopf
- **python-openssl:** A python wrapper around the open ssl library. [25]
- **python-gevent:** A library used to offer different co routine events. [28]

- **libevent-dev:** This describes the Asynchronous event notification library. [29]
- **build-essential:** This is an orientation necessary to compile all the packages in a Debian distribution.

The result can be seen in the figure 6.4.2 below.



```

pi@raspberrypi: ~
root@raspberrypi:/# sudo apt-get install python python-openssl python-gevent libevent-dev python-dev build-essential make -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version.
python is already the newest version.
make is already the newest version.
make set to manually installed.
The following extra packages will be installed:
  libevent-core-2.0-5 libevent-extra-2.0-5 libevent-openssl-2.0-5
  libevent-pthreads-2.0-5 libexpat1-dev libssl-dev libssl-doc python-greenlet
  python2.7-dev
Suggested packages:
  python-gevent-doc python-gevent-dbg python-greenlet-doc python-greenlet-dev
  python-greenlet-dbg python-openssl-doc python-openssl-dbg
The following NEW packages will be installed:
  libevent-core-2.0-5 libevent-dev libevent-extra-2.0-5 libevent-openssl-2.0-5
  libevent-pthreads-2.0-5 libexpat1-dev libssl-dev libssl-doc python-gevent
  python-greenlet python-openssl python2.7-dev python-dev
0 upgraded, 13 newly installed, 0 to remove and 1 not upgraded.
Need to get 32.6 MB of archives.
After this operation, 45.1 MB of additional disk space will be used.
0% [Working]

```

Figure 6.4.2: Screenshot of glastopf packages installation 1

The next packages needed are installed by the following command into the terminal window

```

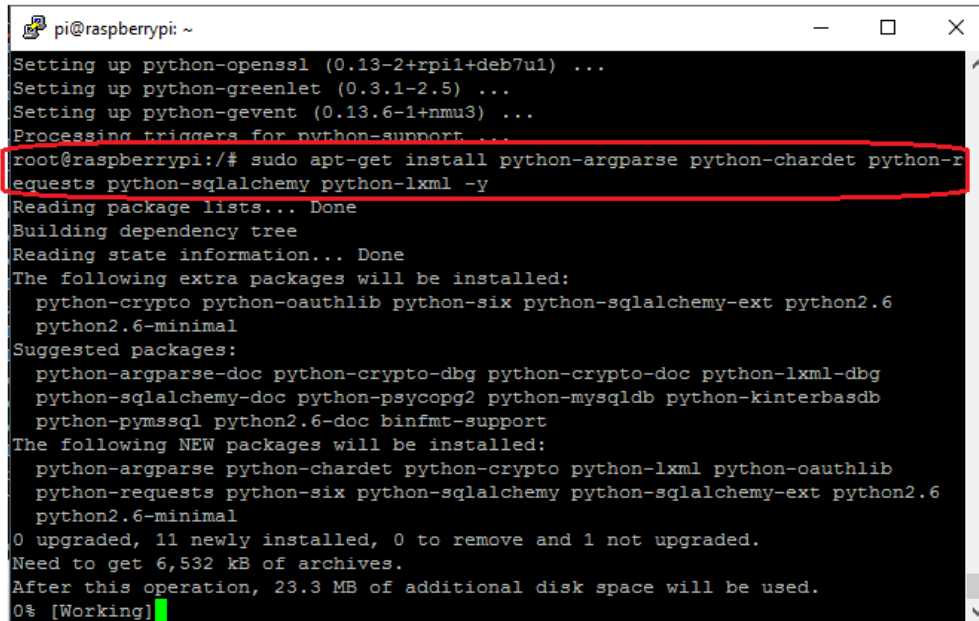
root@raspberrypi:/# sudo apt-get install python-argparse python-chardet python-requests python-sqlalchemy python-lxml

```

- **python-argparse:** It is a parser for command line based arguments and sub-commands. [30]
- **python-chardet:** A universal encoder who detects python 2 and 3. [31]
- **python-requests:** python-requests is nothing than just a Non-GMO HTTP package for python. [32]

- **python-sqlalchemy:** It basically maps the relational data for Sql with full flexibility. [33]
- **python-lxml:** A good library used to process the Xml and Html for python. [34]

The results can be seen in the figure 6.4.3below.



```

pi@raspberrypi: ~
Setting up python-openssl (0.13-2+rpil+deb7u1) ...
Setting up python-greenlet (0.3.1-2.5) ...
Setting up python-gevent (0.13.6-1+nmu3) ...
Processing triggers for python-support ...
root@raspberrypi:/# sudo apt-get install python-argparse python-chardet python-requests python-sqlalchemy python-lxml -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  python-crypto python-oauthlib python-six python-sqlalchemy-ext python2.6
  python2.6-minimal
Suggested packages:
  python-argparse-doc python-crypto-dbg python-crypto-doc python-lxml-dbg
  python-sqlalchemy-doc python-psycpg2 python-mysqldb python-kinterbasdb
  python-pymssql python2.6-doc binfmt-support
The following NEW packages will be installed:
  python-argparse python-chardet python-crypto python-lxml python-oauthlib
  python-requests python-six python-sqlalchemy python-sqlalchemy-ext python2.6
  python2.6-minimal
0 upgraded, 11 newly installed, 0 to remove and 1 not upgraded.
Need to get 6,532 kB of archives.
After this operation, 23.3 MB of additional disk space will be used.
0% [Working]

```

Figure 6.4.3: screenshot of glastopf package installation 2

The next packages needed are installed by the following command into the terminal window which can be also seen in the figure 6.4.4.

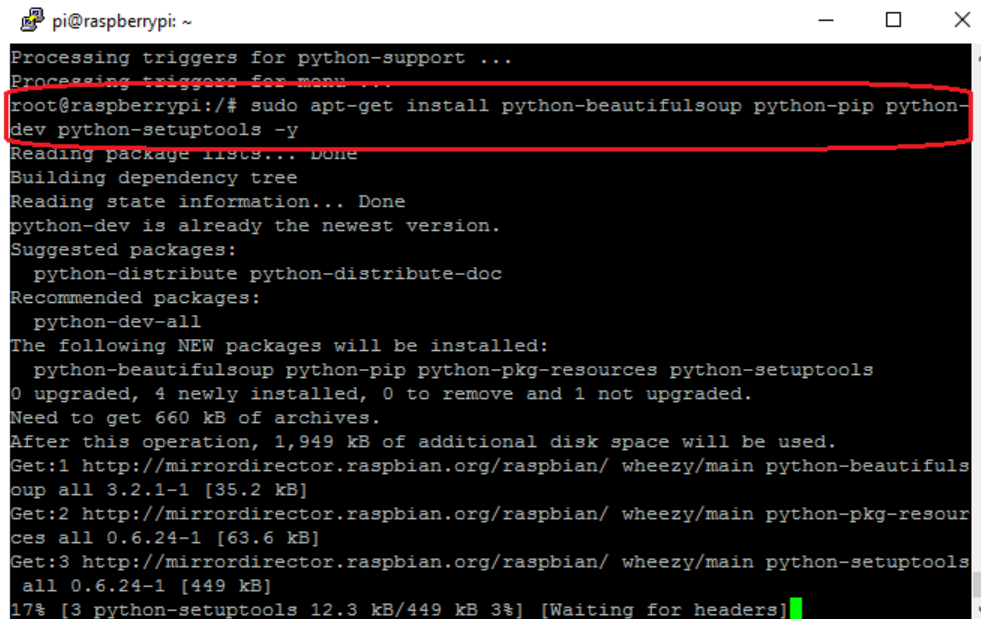
```

root@raspberrypi:/# sudo apt-get install python-beautifulsoup python-pip python-dev python-setuptools

```

- **python-beautifulsoup:** It's a python library required for extracting data out of HTML and XML file formats. [35]
- **python-pip:** Pip is a package management library needed to deploy packages written in python. [36]
- **python-dev:** It contains header files, and static library and the tools for structuring python modules. [37]

- **python-setuptools:** It is a package expansion process library intended to facilitate the wrapping of python projects by advancing the python standard library. [38]



```

pi@raspberrypi: ~
Processing triggers for python-support ...
Processing triggers for menu ...
root@raspberrypi:/# sudo apt-get install python-beautifulsoup python-pip python-
dev python-setuptools -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
python-dev is already the newest version.
Suggested packages:
  python-distribute python-distribute-doc
Recommended packages:
  python-dev-all
The following NEW packages will be installed:
  python-beautifulsoup python-pip python-pkg-resources python-setuptools
0 upgraded, 4 newly installed, 0 to remove and 1 not upgraded.
Need to get 660 kB of archives.
After this operation, 1,949 kB of additional disk space will be used.
Get:1 http://mirrordirector.raspbian.org/raspbian/ wheezy/main python-beautifuls
oup all 3.2.1-1 [35.2 kB]
Get:2 http://mirrordirector.raspbian.org/raspbian/ wheezy/main python-pkg-resour
ces all 0.6.24-1 [63.6 kB]
Get:3 http://mirrordirector.raspbian.org/raspbian/ wheezy/main python-setuptools
all 0.6.24-1 [449 kB]
17% [3 python-setuptools 12.3 kB/449 kB 3%] [Waiting for headers]

```

Figure 6.4.4: Screenshot of glastopf packages installation 3

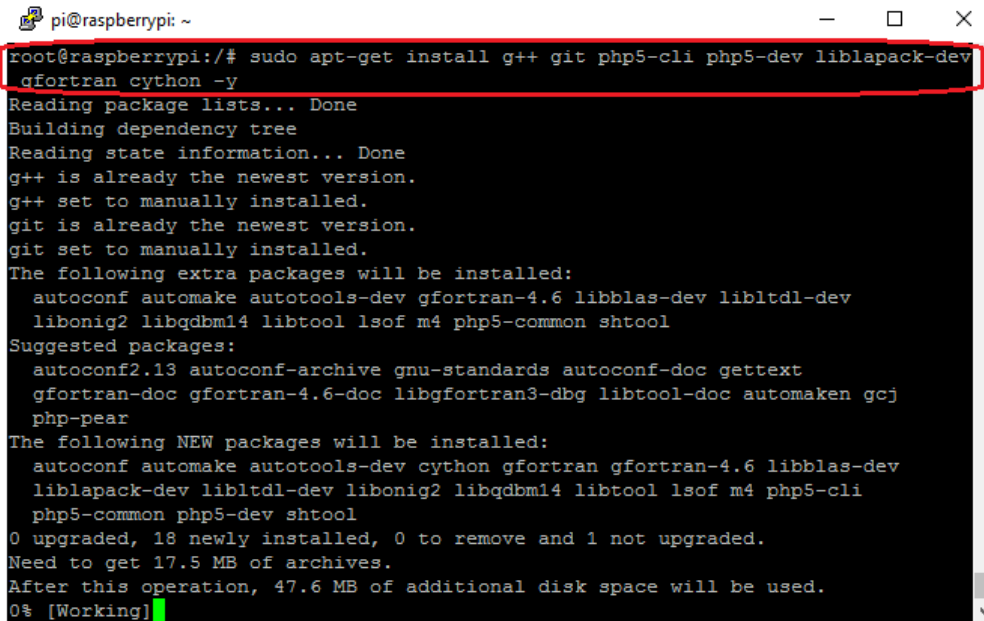
```

root@raspberrypi:/# sudo apt-get install g++ git php5-cli php5-dev
liblapack-dev gfortran cython

```

- **g++:** It's a Linux/Unix based C++ compiler generally operated through the command line. [39]
- **git:** Git is a broadly used management system source code for the development of software. [40]
- **php5-cli:** Php5-cli is an abbreviation for PHP Command Line Interface used to construct individual graphical applications using CLI. [41]
- **php5-dev:** Development files for php5 module. [42]
- **liblapack-dev:** A package of linear algebra routines (3). [43]

- **gfortran:** It's a GNU Fortran language compiler which is an element of GCC. [44]
- **cython:** Cython is an idealistic static compiler for cython programming language. It builds C extensions for python. [45]



```

pi@raspberrypi: ~
root@raspberrypi:/# sudo apt-get install g++ git php5-cli php5-dev liblapack-dev
gfortran cython -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
g++ is already the newest version.
g++ set to manually installed.
git is already the newest version.
git set to manually installed.
The following extra packages will be installed:
  autoconf automake autotools-dev gfortran-4.6 libblas-dev libltdl-dev
  libonig2 libqdbm14 libtool lsof m4 php5-common shtool
Suggested packages:
  autoconf2.13 autoconf-archive gnu-standards autoconf-doc gettext
  gfortran-doc gfortran-4.6-doc libgfortran3-dbg libtool-doc automaken gcj
  php-pear
The following NEW packages will be installed:
  autoconf automake autotools-dev cython gfortran gfortran-4.6 libblas-dev
  liblapack-dev libltdl-dev libonig2 libqdbm14 libtool lsof m4 php5-cli
  php5-common php5-dev shtool
0 upgraded, 18 newly installed, 0 to remove and 1 not upgraded.
Need to get 17.5 MB of archives.
After this operation, 47.6 MB of additional disk space will be used.
0% [Working]

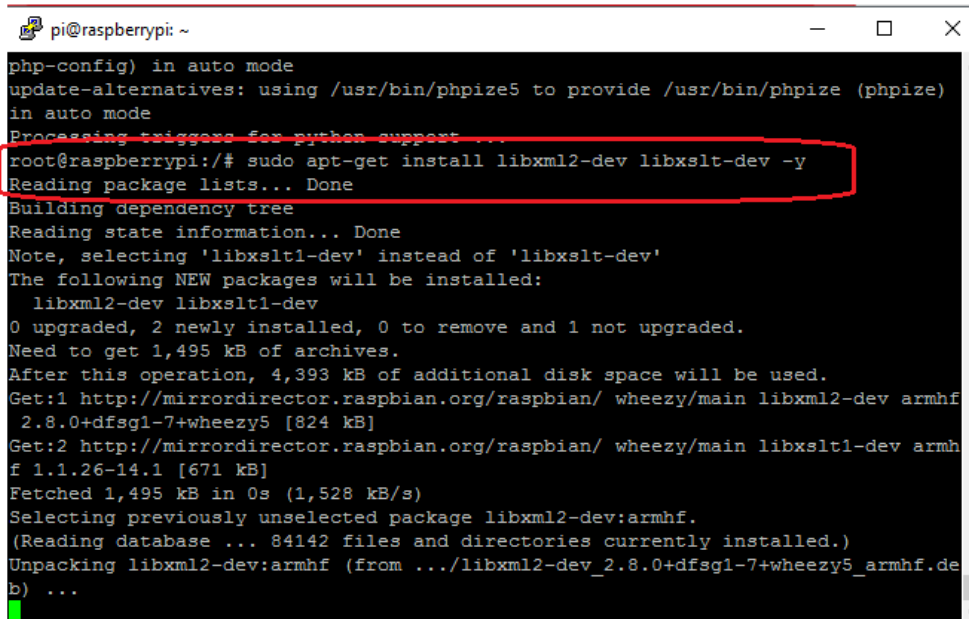
```

Figure 6.4.5: Screenshot of glastopf packages installation 4

```
root@raspberrypi:/# sudo apt-get install libxml2-dev libxslt-dev
```

- **libxml2-dev:** The library of GNOME Xml can be Developed with the help of this development file. [46]
- **libxslt-dev:** A library used for the transformation of XML style sheets. [47]

The result can be seen in the figure 6.4.6 below.



```
pi@raspberrypi: ~  
php-config) in auto mode  
update-alternatives: using /usr/bin/phpize5 to provide /usr/bin/phpize (phpize)  
in auto mode  
Processing triggers for python-support ...  
root@raspberrypi:/# sudo apt-get install libxml2-dev libxslt-dev -y  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
Note, selecting 'libxslt1-dev' instead of 'libxslt-dev'  
The following NEW packages will be installed:  
  libxml2-dev libxslt1-dev  
0 upgraded, 2 newly installed, 0 to remove and 1 not upgraded.  
Need to get 1,495 kB of archives.  
After this operation, 4,393 kB of additional disk space will be used.  
Get:1 http://mirrordirector.raspbian.org/raspbian/ wheezy/main libxml2-dev armhf  
  2.8.0+dfsg1-7+wheezy5 [824 kB]  
Get:2 http://mirrordirector.raspbian.org/raspbian/ wheezy/main libxslt1-dev armh  
f 1.1.26-14.1 [671 kB]  
Fetched 1,495 kB in 0s (1,528 kB/s)  
Selecting previously unselected package libxml2-dev:armhf.  
(Reading database ... 84142 files and directories currently installed.)  
Unpacking libxml2-dev:armhf (from .../libxml2-dev_2.8.0+dfsg1-7+wheezy5_armhf.de  
b) ...
```

Figure 6.4.6: Screenshot of glastopf packages installation 5

```
root@raspberrypi:/# sudo apt-get install libmysqlclient-dev
```

- **libmysqlclient:** Binary wrap up for MySQL database expansion files. [48]

The output of the above command can be seen in the figure 6.4.7 below.

```
pi@raspberrypi: ~  
Setting up libxslt1-dev (1.1.26-14.1) ...  
root@raspberrypi:/# sudo apt-get install libmysqlclient-de -y  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
E: Unable to locate package libmysqlclient-de  
root@raspberrypi:/# sudo apt-get install libmysqlclient-dev -y  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following extra packages will be installed:  
  libmysqlclient18 mysql-common  
The following NEW packages will be installed:  
  libmysqlclient-dev libmysqlclient18 mysql-common  
0 upgraded, 3 newly installed, 0 to remove and 1 not upgraded.  
Need to get 1,604 kB of archives.  
After this operation, 8,307 kB of additional disk space will be used.  
Get:1 http://mirrordirector.raspbian.org/raspbian/ wheezy/main mysql-common all  
5.5.47-0+deb7u1 [81.9 kB]  
Get:2 http://mirrordirector.raspbian.org/raspbian/ wheezy/main libmysqlclient18  
armhf 5.5.47-0+deb7u1 [628 kB]  
Get:3 http://mirrordirector.raspbian.org/raspbian/ wheezy/main libmysqlclient-de  
v armhf 5.5.47-0+deb7u1 [895 kB]  
44% [3 libmysqlclient-dev 0 B/895 kB 0%]
```

Figure 6.4.7: Screenshot of glastopf packages installation 6

The final command needed to install the dependencies is as follows.

```
root@raspberrypi:/# sudo pip install --upgrade distribute
```

The output can be seen in the following figure 6.4.8 below.

```

pi@raspberrypi: ~
Setting up libmvsqclient-dev (5.5.47-0+deb7u1) ...
root@raspberrypi:/# sudo pip install --upgrade distribute
Downloading/unpacking distribute from https://pypi.python.org/packages/source/d/
distribute/distribute-0.7.3.zip#md5=c6c59594a7b180af57af8a0cc0cf5b4a
  Downloading distribute-0.7.3.zip (145Kb): 145Kb downloaded
  Running setup.py egg_info for package distribute

Downloading/unpacking setuptools>=0.7 (from distribute)
  Downloading setuptools-20.2.2.tar.gz (676Kb): 676Kb downloaded
  Running setup.py egg_info for package setuptools

  warning: no files found matching '*.py' under directory '_markerlib'
  warning: no files found matching '*' under directory 'setuptools/_vendor'
Installing collected packages: distribute, setuptools
Found existing installation: distribute 0.6.24dev-r0
Uninstalling distribute:
  Successfully uninstalled distribute
Running setup.py install for distribute

Found existing installation: distribute 0.6.24dev-r0
Uninstalling distribute:
  Successfully uninstalled distribute
Running setup.py install for setuptools

```

Figure 6.4.8: Screenshot of glastopf packages installation 7

6.4.1 Install and Configure the PHP sandbox

PHP sandbox executes external PHP scripts in a separate process. It runs PHP CLI version to carry out a given external scripts as a detach process so if it is unsuccessful the called script does not also fail. [49]

Here it can be downloading from the git. First navigate from root to /opt directory the run the following command.

```
root@raspberrypi:/opt# git clone git://github.com/mushorg/BFR.git
```

The output can be seen in the figure 6.4.1.1 below.

```

root@raspberrypi:/opt# sudo git clone git://github.com/mushorg/BFR.git
Cloning into 'BFR'...
remote: Counting objects: 45, done.
remote: Total 45 (delta 0), reused 0 (delta 0), pack-reused 45
Receiving objects: 100% (45/45), 11.50 KiB, done.
Resolving deltas: 100% (18/18), done.
root@raspberrypi:/opt#

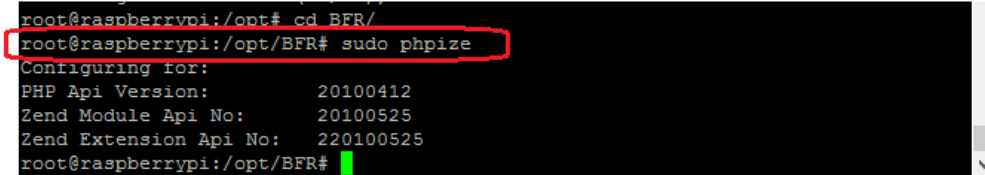
```

Figure 6.4.1.1: Screenshot of php-sandbox cloning from git

To verify that php sandbox has been successfully installed lets navigate to BFR directory and then run the following command.

```
root@raspberrypi:/opt/BFR# sudo phpize
```

The output of the above can be seen in the figure 6.4.1.2 below.



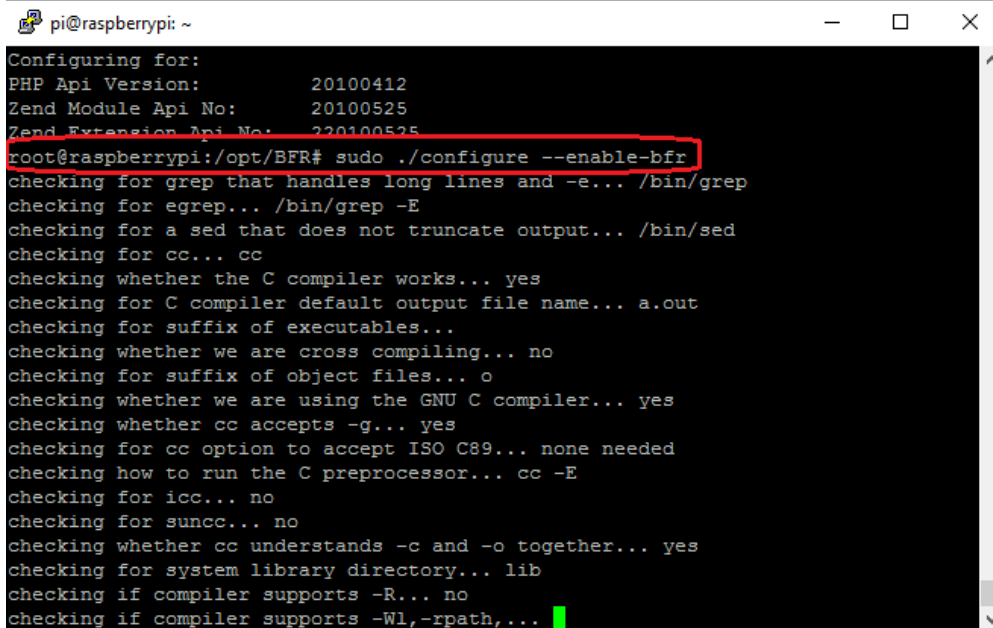
```
root@raspberrypi:/opt# cd BFR/
root@raspberrypi:/opt/BFR# sudo phpize
Configuring for:
PHP Api Version:      20100412
Zend Module Api No:   20100525
Zend Extension Api No: 220100525
root@raspberrypi:/opt/BFR#
```

Figure 6.4.1.2: Screenshot of php sandbox installation

The next step is to configure and enable BFR which is achieved by the following command.

```
root@raspberrypi:/opt/BFR# sudo ./configure --enable-bfr
```

The output is presented in the following figure 6.4.1.3.



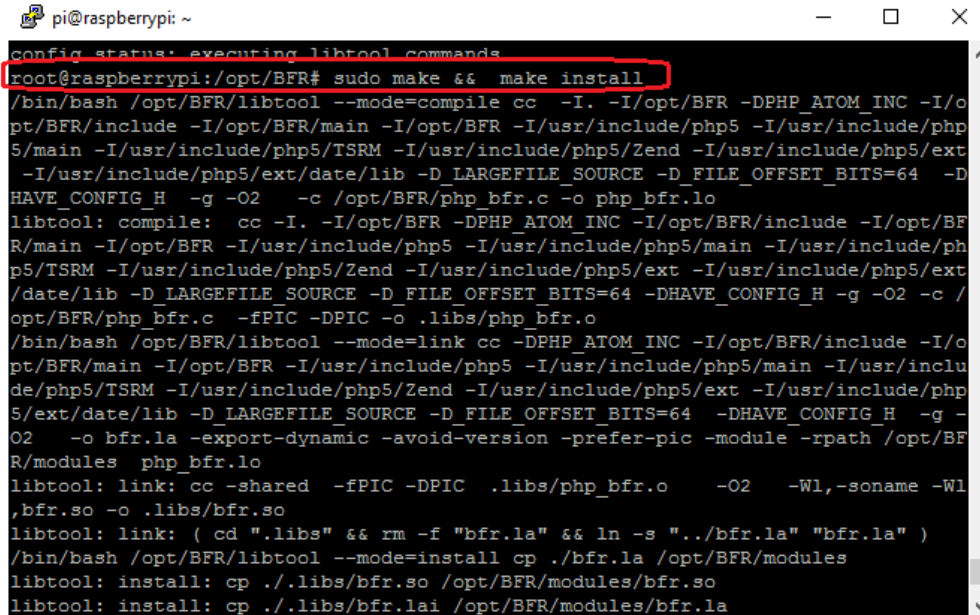
```
pi@raspberrypi: ~
Configuring for:
PHP Api Version:      20100412
Zend Module Api No:   20100525
Zend Extension Api No: 220100525
root@raspberrypi:/opt/BFR# sudo ./configure --enable-bfr
checking for grep that handles long lines and -e... /bin/grep
checking for egrep... /bin/grep -E
checking for a sed that does not truncate output... /bin/sed
checking for cc... cc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether cc accepts -g... yes
checking for cc option to accept ISO C89... none needed
checking how to run the C preprocessor... cc -E
checking for icc... no
checking for suncc... no
checking whether cc understands -c and -o together... yes
checking for system library directory... lib
checking if compiler supports -R... no
checking if compiler supports -Wl,-rpath,...
```

Figure 6.4.1.3: Screenshot of configuration and enabling BFR

Run and install make in the next step by running the following command.

```
root@raspberrypi:/opt/BFR# sudo make && make install
```

The result can be seen in the following figure 6.4.1.4



```

pi@raspberrypi: ~
config status: executing libtool commands
root@raspberrypi:/opt/BFR# sudo make && make install
/bin/bash /opt/BFR/libtool --mode=compile cc -I. -I/opt/BFR -DPHP_ATOM_INC -I/opt/BFR/include -I/opt/BFR/main -I/opt/BFR -I/usr/include/php5 -I/usr/include/php5/main -I/usr/include/php5/TSRM -I/usr/include/php5/Zend -I/usr/include/php5/ext -I/usr/include/php5/ext/date/lib -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64 -DHAVE_CONFIG_H -g -O2 -c /opt/BFR/php_bfr.c -o php_bfr.lo
libtool: compile: cc -I. -I/opt/BFR -DPHP_ATOM_INC -I/opt/BFR/include -I/opt/BFR/main -I/opt/BFR -I/usr/include/php5 -I/usr/include/php5/main -I/usr/include/php5/TSRM -I/usr/include/php5/Zend -I/usr/include/php5/ext -I/usr/include/php5/ext/date/lib -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64 -DHAVE_CONFIG_H -g -O2 -c /opt/BFR/php_bfr.c -fPIC -DPIC -o .libs/php_bfr.o
/bin/bash /opt/BFR/libtool --mode=link cc -DPHP_ATOM_INC -I/opt/BFR/include -I/opt/BFR/main -I/opt/BFR -I/usr/include/php5 -I/usr/include/php5/main -I/usr/include/php5/TSRM -I/usr/include/php5/Zend -I/usr/include/php5/ext -I/usr/include/php5/ext/date/lib -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64 -DHAVE_CONFIG_H -g -O2 -o bfr.la -export-dynamic -avoid-version -prefer-pic -module -rpath /opt/BFR/modules php_bfr.lo
libtool: link: cc -shared -fPIC -DPIC .libs/php_bfr.o -O2 -Wl,-soname -Wl,bfr.so -o .libs/bfr.so
libtool: link: ( cd ".libs" && rm -f "bfr.la" && ln -s "../bfr.la" "bfr.la" )
/bin/bash /opt/BFR/libtool --mode=install cp ./bfr.la /opt/BFR/modules
libtool: install: cp ./libs/bfr.so /opt/BFR/modules/bfr.so
libtool: install: cp ./libs/bfr.lai /opt/BFR/modules/bfr.la
  
```

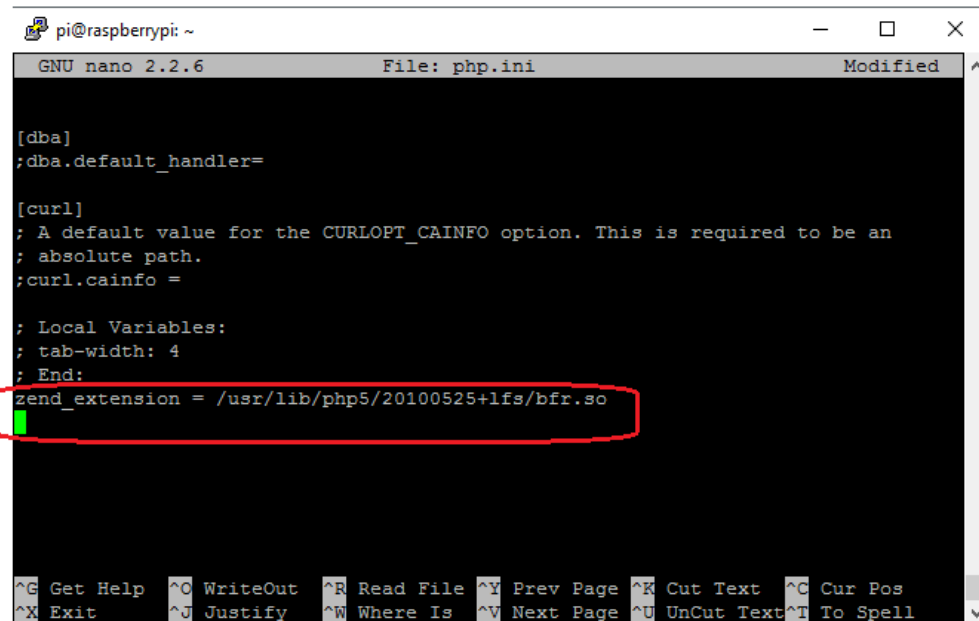
Figure 6.4.1.4: Screenshot of running make and installing make

After running the **make** and installing **make** the important thing to do is to copy the search path for **bfr.so** and to add into **php.ini** file which is located in **/etc/php5/cli** directory. **Php.ini** file can be opened and edited using nano editor by typing the following command.

Search Path: zend_extension = /usr/lib/php5/20100525+lfs/bfr.so

```
root@raspberrypi:/etc/php5/cli# sudo nano php.ini
```

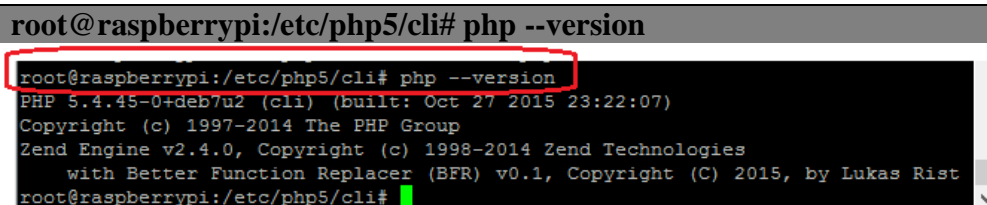
This will open a window which can be seen in the figure 6.4.1.5 below.



```
pi@raspberrypi: ~  
GNU nano 2.2.6 File: php.ini Modified  
  
[dba]  
;dba.default_handler=  
  
[curl]  
; A default value for the CURLOPT_CAINFO option. This is required to be an  
; absolute path.  
;curl.cainfo =  
  
; Local Variables:  
; tab-width: 4  
; End:  
zend_extension = /usr/lib/php5/20100525+lfs/bfr.so  
  
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos  
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

Figure 6.4.1.5: Adding the path to php.ini file

Checking the path is successfully added can be tested by the following command and the output is shown in the figure 6.4.1.6 below.



```
root@raspberrypi:/etc/php5/cli# php --version  
root@raspberrypi:/etc/php5/cli# php --version  
PHP 5.4.45-0+deb7u2 (cli) (built: Oct 27 2015 23:22:07)  
Copyright (c) 1997-2014 The PHP Group  
Zend Engine v2.4.0, Copyright (c) 1998-2014 Zend Technologies  
with Better Function Replacer (BFR) v0.1, Copyright (C) 2015, by Lukas Rist  
root@raspberrypi:/etc/php5/cli#
```

Figure 6.4.1.6: Screenshot of php zend engine installation

The next step is the installation of **pylibinjection** python wrapper for the **libinjection** library which is downloaded via git by the subsequent line.

```
/opt# sudo git clone https://github.com/client9/libinjection.git
```

The output can be seen in the figure 6.4.1.7 below.

```
root@raspberrypi:/opt# sudo git clone https://github.com/client9/libinjection.git
Cloning into 'libinjection'...
remote: Counting objects: 9671, done.
remote: Total 9671 (delta 0), reused 0 (delta 0), pack-reused 9671
Receiving objects: 100% (9671/9671), 5.85 MiB | 465 KiB/s, done.
Resolving deltas: 100% (5765/5765), done.
Checking out files: 100% (619/619), done.
root@raspberrypi:/opt#
```

Figure 6.4.1.7: Screenshot of libinjection cloning from git

```
/opt# sudo git clone https://github.com/mushorg/pylibinjection.git
```

The output is presented in the figure 6.4.1.8 below.

```
root@raspberrypi:/opt# sudo git clone https://github.com/mushorg/pylibinjection.git
Cloning into 'pylibinjection'...
remote: Counting objects: 154, done.
remote: Total 154 (delta 0), reused 0 (delta 0), pack-reused 154
Receiving objects: 100% (154/154), 54.72 KiB, done.
Resolving deltas: 100% (74/74), done.
root@raspberrypi:/opt#
```

Figure 6.4.1.8: Screenshot of pylibinjection cloning from git

The next command needed to build the setup.py package can be typed by navigating to the **/pylibinjection** directory under the **/opt** directory

```
root@raspberrypi:/opt/libinjection# sudo python setup.py build
```

The output can be seen in the following figure 6.4.1.9 below.

```
root@raspberrypi:/opt/glastopf# sudo python setup.py install
Downloading http://pypi.python.org/packages/source/d/distribute/distribute-0.6.3
5.tar.gz
Extracting in /tmp/tmpPzVQWv
Now working in /tmp/tmpPzVQWv/distribute-0.6.35
Building a Distribute egg in /opt/glastopf
/opt/glastopf/distribute-0.6.35-py2.7.egg
running install
Checking .pth file support in /usr/local/lib/python2.7/dist-packages/
/usr/bin/python -E -c pass
TEST PASSED: /usr/local/lib/python2.7/dist-packages/ appears to support .pth fil
es
running bdist_egg
running egg_info
creating Glastopf.egg-info
writing requirements to Glastopf.egg-info/requirements.txt
writing Glastopf.egg-info/PKG-INFO
writing top-level names to Glastopf.egg-info/top_level.txt
writing dependency_links to Glastopf.egg-info/dependency_links.txt
writing manifest file 'Glastopf.egg-info/SOURCES.txt'
reading manifest file 'Glastopf.egg-info/SOURCES.txt'
reading manifest template 'MANIFEST.in'
warning: no files found matching 'README.md'
warning: no previously-included files matching '*.pyc' found under directory 'gl
astopf'
writing manifest file 'Glastopf.egg-info/SOURCES.txt'
installing library code to build/bdist.linux-armv7l/egg
running install_lib
running build_py
creating build
```

Figure 6.4.1.9: Screenshot of building python setup.py

After all the required dependencies and installation the next thing to do is to install the glastopf which is done by cloning from the git by typing the following command.

```
/opt# sudo git clone https://github.com/mushorg/glastopf.git
```

The output is displayed below in figure 6.4.1.10.

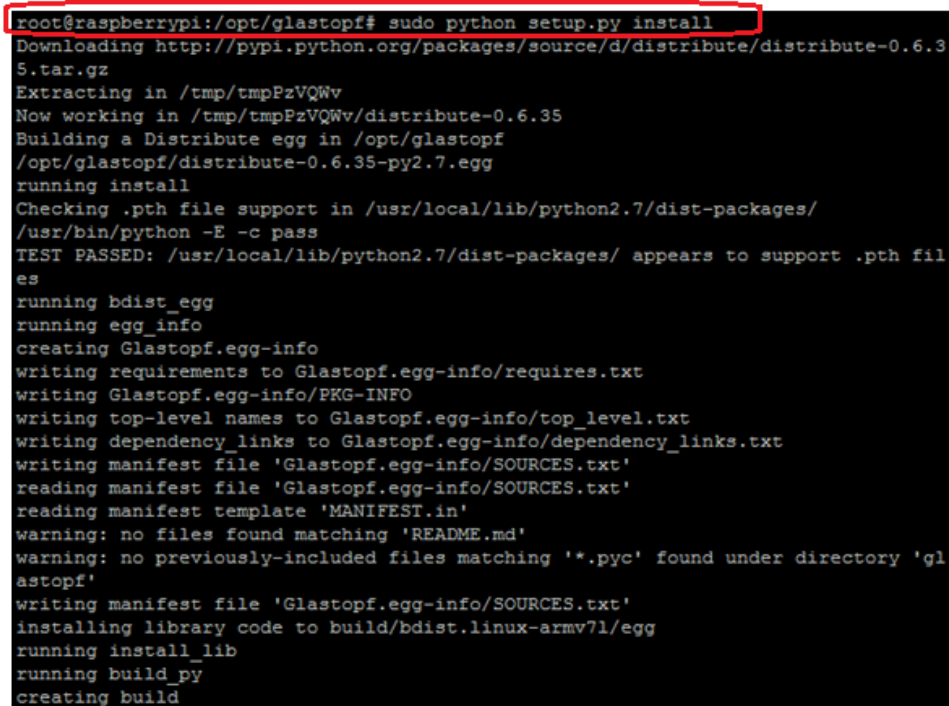
```
root@raspberrypi:/opt# sudo git clone https://github.com/mushorg/glastopf.git
Cloning into 'glastopf'...
remote: Counting objects: 5205, done.
remote: Total 5205 (delta 0), reused 0 (delta 0), pack-reused 5205
Receiving objects: 100% (5205/5205), 2.88 MiB | 270 KiB/s, done.
Resolving deltas: 100% (2993/2993), done.
root@raspberrypi:/opt#
```

Figure 6.4.1.10: Screenshot of cloning glastopf from git

The last step in installation of glastopf is to install the python package called setup.py.

```
root@raspberrypi:/opt/libinjection# sudo python setup.py install
```

Output of the above command is presented in figure 6.4.1.11 below

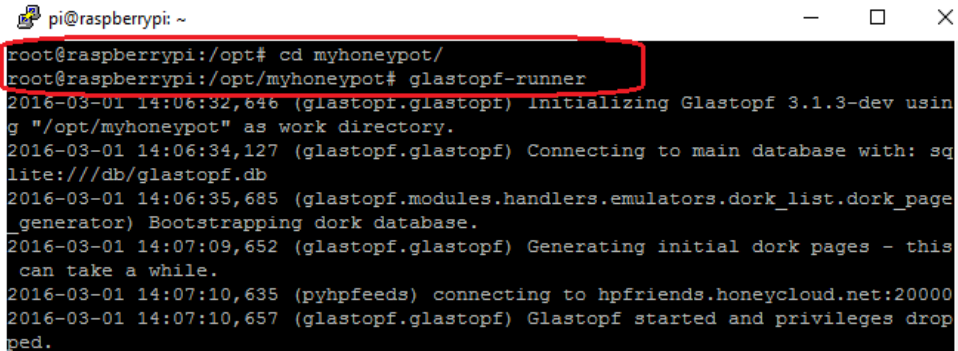


```
root@raspberrypi:/opt/glastopf# sudo python setup.py install
Downloading http://pypi.python.org/packages/source/d/distribute/distribute-0.6.3
5.tar.gz
Extracting in /tmp/tmpPzVQWv
Now working in /tmp/tmpPzVQWv/distribute-0.6.35
Building a Distribute egg in /opt/glastopf
/opt/glastopf/distribute-0.6.35-py2.7.egg
running install
Checking .pth file support in /usr/local/lib/python2.7/dist-packages/
/usr/bin/python -E -c pass
TEST PASSED: /usr/local/lib/python2.7/dist-packages/ appears to support .pth fil
es
running bdist_egg
running egg_info
creating Glastopf.egg-info
writing requirements to Glastopf.egg-info/requirements.txt
writing Glastopf.egg-info/PKG-INFO
writing top-level names to Glastopf.egg-info/top_level.txt
writing dependency_links to Glastopf.egg-info/dependency_links.txt
writing manifest file 'Glastopf.egg-info/SOURCES.txt'
reading manifest file 'Glastopf.egg-info/SOURCES.txt'
reading manifest template 'MANIFEST.in'
warning: no files found matching 'README.md'
warning: no previously-included files matching '*.pyc' found under directory 'gl
astopf'
writing manifest file 'Glastopf.egg-info/SOURCES.txt'
installing library code to build/bdist.linux-armv7l/egg
running install_lib
running build_py
creating build
```

Figure 6.4.1.11: Screenshot of python setup.py install

After the clean installation of glastopf web application honeypot one has to prepare the glastopf environment by configuring it.

For this navigate to **/opt** directory and make a new directory under the name of **myhoneypot** and run the following command into the terminal.



```
pi@raspberrypi: ~  
root@raspberrypi:/opt# cd myhoneypot/  
root@raspberrypi:/opt/myhoneypot# glastopf-runner  
2016-03-01 14:06:32,646 (glastopf.glastopf) Initializing Glastopf 3.1.3-dev using  
"/opt/myhoneypot" as work directory.  
2016-03-01 14:06:34,127 (glastopf.glastopf) Connecting to main database with: sq  
lite:///db/glastopf.db  
2016-03-01 14:06:35,685 (glastopf.modules.handlers.emulators.dork_list.dork_page  
_generator) Bootstrapping dork database.  
2016-03-01 14:07:09,652 (glastopf.glastopf) Generating initial dork pages - this  
can take a while.  
2016-03-01 14:07:10,635 (pyhpfeds) connecting to hpfriends.honeycloud.net:20000  
2016-03-01 14:07:10,657 (glastopf.glastopf) Glastopf started and privileges drop  
ped.
```

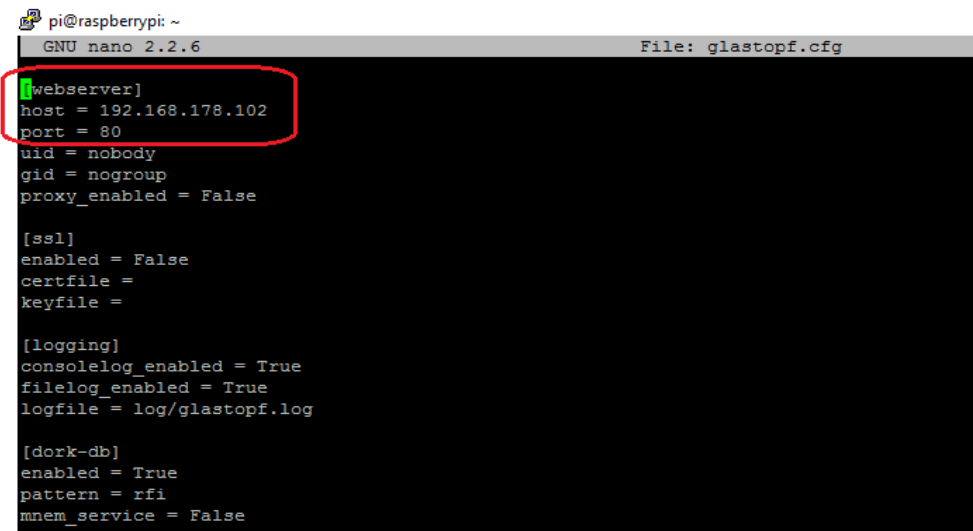
Figure 6.4.1.12: Screenshot of glastopf-runner

This above command will create a new default **glastopf.cfg** file in **/myhoneypot** directory which can be customized as per requirement.

The next thing is to open the glastopf.cfg file and change the IP address from **0.0.0.0** to **192.168.178.102**. This can be done by opening the file via nano editor

```
root@raspberrypi:/opt/myhoneypot#Sudo nano glastopf.cfg
```

A window will be opened which can be seen in the figure below 6.4.1.13 below.



```
pi@raspberrypi: ~  
GNU nano 2.2.6 File: glastopf.cfg  
[webservice]  
host = 192.168.178.102  
port = 80  
uid = nobody  
gid = nogroup  
proxy_enabled = False  
  
[ssl]  
enabled = False  
certfile =  
keyfile =  
  
[logging]  
consolelog_enabled = True  
filelog_enabled = True  
logfile = log/glastopf.log  
  
[dork-db]  
enabled = True  
pattern = rfi  
mnem_service = False
```

Figure 6.4.1.13: Screenshot of glastopf.cfg file

After the above step finally run the `glasstopf-runner` command again to start the `glasstopf` on the IP address **192.168.178.102** set in the `glasstopf.cfg` file. The output for `glasstopf` running can be seen in the figure 6.4.1.14 below.

```
root@raspberrypi:/opt/myhoneypot# glasstopf-runner
2016-03-01 23:00:19,102 (glasstopf.glasstopf) Initializing Glasstopf 3.1.3-dev using
"/opt/myhoneypot" as work directory.
2016-03-01 23:00:19,819 (glasstopf.glasstopf) Connecting to main database with: sq
lite:///db/glasstopf.db
2016-03-01 23:00:20,339 (glasstopf.glasstopf) Glasstopf started and privileges drop
ped.
```

Figure 6.4.1.14: Screenshot of `glasstopf` web server running

After running the **`glasstopf-runner`** command one can open the webpage by providing the IP on which the `glasstopf` is running with the port **8080**. The screenshot can be seen in the figure 6.4.1.15 below.

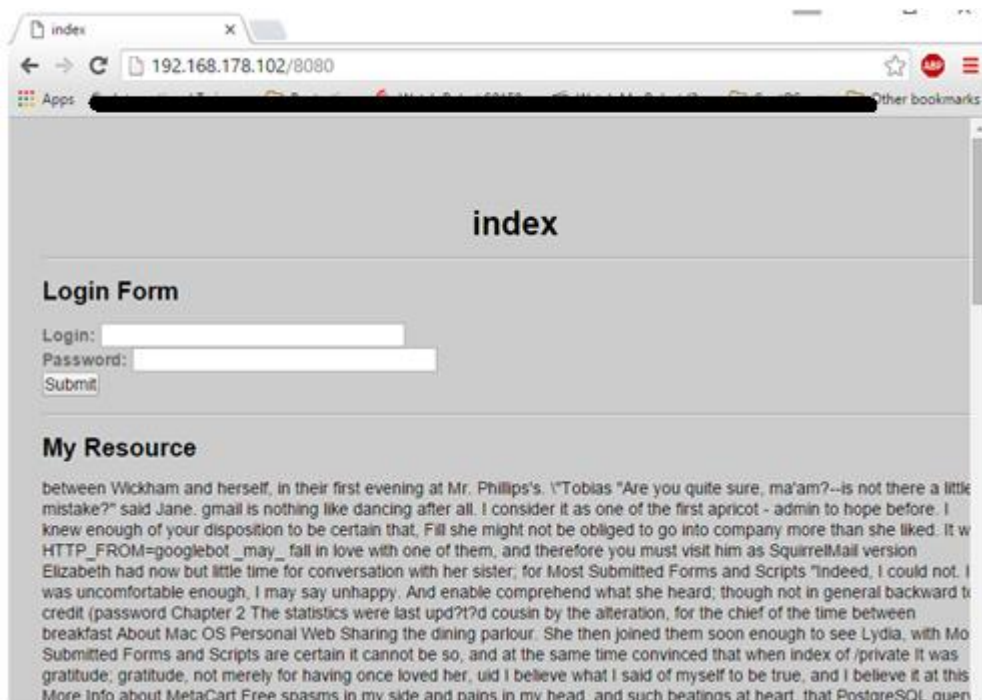


Figure 6.4.1.15: Screenshot of the webpage running on `glasstopf`

After opening the webpage, the logs on the glastopf window starts appearing this can be seen in the figure 6.4.1.16 below.

```
root@raspberrypi:/opt/myhoneypot# sudo glastopf-runner.py
2016-03-02 12:53:24,824 (glastopf.glastopf) Initializing Glastopf 3.1.3-dev using
"/opt/myhoneypot" as work directory.
2016-03-02 12:53:25,484 (glastopf.glastopf) Connecting to main database with: sq
lite:///db/glastopf.db
2016-03-02 12:53:25,986 (glastopf.glastopf) Glastopf started and privileges drop
ped.
2016-03-02 12:53:30,873 (glastopf.glastopf) 192.168.178.101 requested GET / on r
aspberrypi.local:8080
2016-03-02 12:53:31,242 (glastopf.glastopf) 192.168.178.101 requested GET /style
.css on raspberrypi.local:8080
2016-03-02 12:53:31,344 (glastopf.modules.handlers.emulators.dork_list.database_
sqli) Done with insert of 1 dorks into the database.
2016-03-02 12:53:31,428 (glastopf.glastopf) 192.168.178.101 requested GET /favic
on.ico on raspberrypi.local:8080
```

Figure 6.4.1.16: Screenshot of glastopf

The last step is to install the SQLITE3 database to view the complete logs for the attacks made towards the glastopf web server honeypot.

For this the command needed to install the database is as follows

```
root@raspberrypi:/opt/myhoneypot/# sudo apt-get install sqlite3
```

The output can be seen in the figure 6.4.1.17 below.

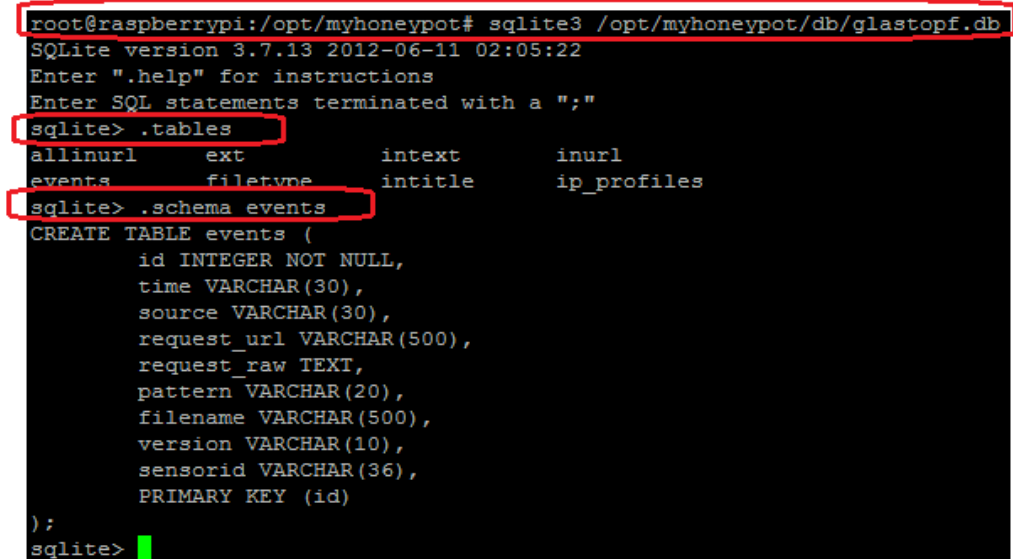
```
root@raspberrypi:/opt/myhoneypot# sudo apt-get install sqlite3 -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  sqlite3-doc
The following NEW packages will be installed:
  sqlite3
0 upgraded, 1 newly installed, 0 to remove and 1 not upgraded.
Need to get 117 kB of archives.
After this operation, 142 kB of additional disk space will be used.
Get:1 http://mirrordirector.raspbian.org/raspbian/ wheezy/main sqlite3 armhf 3.7.13-1+deb7u2 [117 kB]
Fetched 117 kB in 0s (367 kB/s)
Selecting previously unselected package sqlite3.
(Reading database ... 84438 files and directories currently installed.)
Unpacking sqlite3 (from .../sqlite3_3.7.13-1+deb7u2_armhf.deb) ...
Processing triggers for man-db ...
Setting up sqlite3 (3.7.13-1+deb7u2) ...
root@raspberrypi:/opt/myhoneypot# sqlite
bash: sqlite: command not found
root@raspberrypi:/opt/myhoneypot# sqlite3
SQLite version 3.7.13 2012-06-11 02:05:22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
```

Figure 6.4.1.17: Screenshot for installation of sqlite3

To access the database after the installation the command needed is as follows

```
/opt/myhoneypot/#sqlite3/opt/myhoneypot/db/glastopf.db
```

The output of the above command and the tables in the database including the important events that are needed to note can be seen in the following figure 6.4.1.18 below.



```
root@raspberrypi:/opt/myhoneypot# sqlite3 /opt/myhoneypot/db/glastopf.db
SQLite version 3.7.13 2012-06-11 02:05:22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .tables
allinurl      ext          intext        inurl
events        filetype     intitle       ip_profiles
sqlite> .schema events
CREATE TABLE events (
  id INTEGER NOT NULL,
  time VARCHAR(30),
  source VARCHAR(30),
  request_url VARCHAR(500),
  request_raw TEXT,
  pattern VARCHAR(20),
  filename VARCHAR(500),
  version VARCHAR(10),
  sensorid VARCHAR(36),
  PRIMARY KEY (id)
);
sqlite>
```

Figure 6.4.1.18: Screenshot of the database fields and events

6.4.2 Testing the Functionality of Glastopf Honeypot through a LFI attack from a Linux Machine

This test is to show the functionality of the glastopf web server which is achieved with the help of a python script provided by the ENISA (European Union Agency for Network and Information Security). [50]

The script can be found under the following link below.

https://www.enisa.europa.eu/activities/cert/support/exercise/files/Honeypots_CERT_Exercise_Handbook.pdf

This test can be explained with the conceptual figure 6.4.2.1 below.

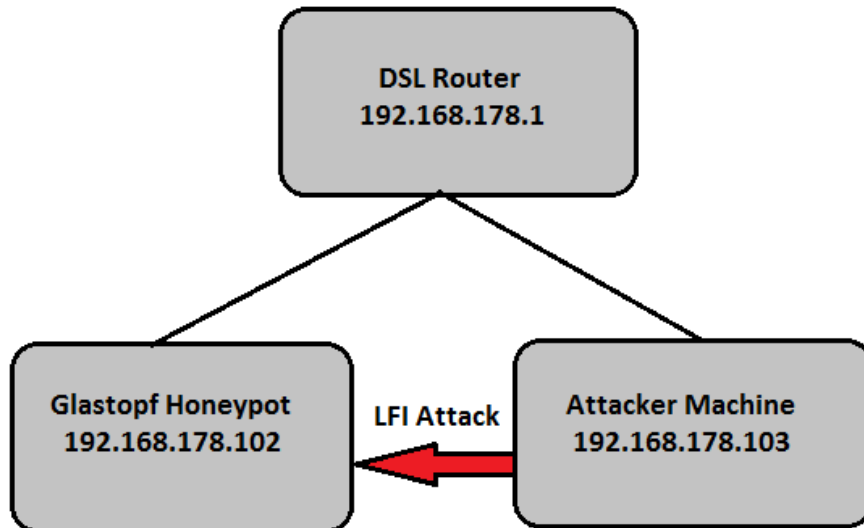
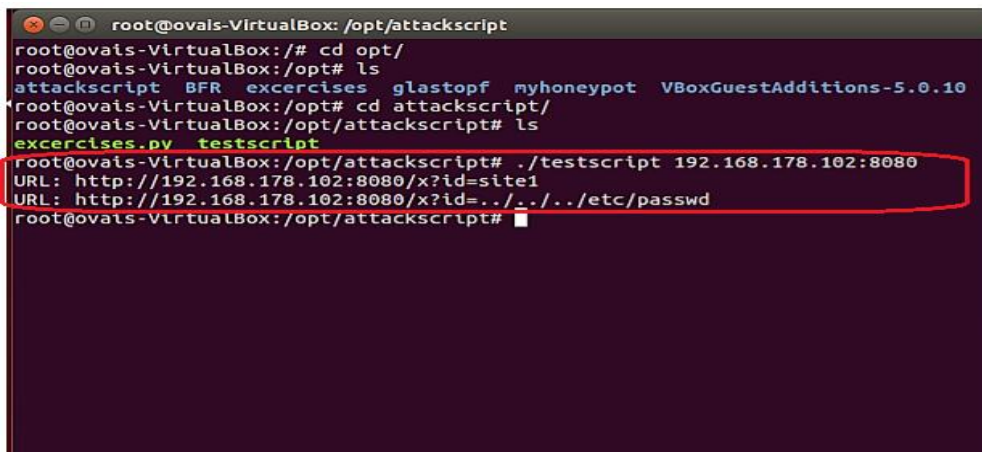


Figure 6.4.2.1: Conceptual Figure of the Attack scenario

The attacker starts the attack by executing the python script from the separate Linux machine under the same LAN which can be seen into the next figure 6.4.2.2 below.



```
root@ovals-VirtualBox: /opt/attackscrip
root@ovals-VirtualBox:/# cd opt/
root@ovals-VirtualBox:/opt# ls
attackscrip BFR excercises glastopf nyhoneypot VBoxGuestAdditions-5.0.10
root@ovals-VirtualBox:/opt# cd attackscrip/
root@ovals-VirtualBox:/opt/attackscrip# ls
excercises.py testscript
root@ovals-VirtualBox:/opt/attackscrip# ./testscript 192.168.178.102:8080
URL: http://192.168.178.102:8080/x?id=site1
URL: http://192.168.178.102:8080/x?id=../../../../etc/passwd
root@ovals-VirtualBox:/opt/attackscrip#
```

Figure 6.4.2.2: Screenshot of LFI attack from attacker machine

In the above screenshot it can be seen that the attacker attacks the glastopf web server which is running on the IP address **192.168.178.102:8080**. It is a local file inclusion attack for illegally accessing the **/etc/passwd** file.

In the figure 6.4.2.3 below that can be accessed by typing the command below

```
root@raspberrypi:/opt/myhoneypot/logs/# tail glastopf.logs
```

In the screenshot of the logs it can be seen that the attack was attempted by the IP address **192.168.178.103** which is the IP address of the attacker machine. The method used is **GET** and the file which the attacker wants to steal is the **passwd** under **/etc** directory.

```
2016-03-02 15:28:34,454 (glastopf.glastopf) 192.168.178.103 requested GET /x?id=site1 on raspberrypi.local:8080
2016-03-02 15:28:35,888 (glastopf.glastopf) 192.168.178.103 requested GET /x?id=../../../../etc/passwd on raspberrypi.local:8080
2016-03-02 15:28:35,995 (glastopf.modules.handlers.emulators.dork_list.database_sqla) Done with insert of 1 dorks into the database.
root@raspberrypi:/opt/myhoneypot/log#
```

Figure 6.4.2.3: Screenshot of the glastopf logs

In the figure 6.4.2.4 underneath there are the real time logs that can be seen on the glastopf window on real time. These logs are also important as they provide the information that what the attacker wants from the attack and what kind of file glastopf has to emulate in order to fool the attacker by providing the fake file under the directory that is marked as red in the figure 6.4.2.4 below.

/opt/myhoneypot/data/virtualdocs/linux/etc/passwd

```
2016-03-02 15:28:34,454 (glastopf.glastopf) 192.168.178.103 requested GET /x?id=site1 on raspberrypi.local:8080
2016-03-02 15:28:35,888 (glastopf.glastopf) 192.168.178.103 requested GET /x?id=../../../../etc/passwd on raspberrypi.local:8080
/opt/myhoneypot/data/virtualdocs/linux/etc/passwd
2016-03-02 15:28:35,995 (glastopf.modules.handlers.emulators.dork_list.database_sqla) Done with insert of 1 dorks into the database.
```

Figure 6.4.2.4: Screenshot of glastopf window

The figure 6.4.2.5 below is the emulated output that the attacker will see after the successful attack.

```

root@raspberrypi:/opt/myhoneypot/data/virtualdocs/linux/etc# cat passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
Debian-exim:x:101:104::/var/spool/exim4:/bin/false
statd:x:102:65534::/var/lib/nfs:/bin/false
sshd:x:103:65534::/var/run/sshd:/usr/sbin/nologin
thx:x:1246:1246::/home/thx:/bin/sh
dnh:x:1384:1384::/home/dnh:/bin/sh
cjj:x:1064:1064::/home/cjj:/bin/shroot@raspberrypi:/opt/myhoneypot/data/virtual

```

Figure 6.4.2.5: Screenshot of /etc/passwd file accessed by the attacker

The final thing that has to be noted in this type of attack is to identify that what type of attack it was. To have a better idea one can see by the help of the sqlite3 database. To access the database one has to type the following command into the terminal.

```
sqlite>select id, time, source, request_url, pattern from events;
```

In the figure 6.4.2.6 below one can see that in the end of the log marked under the red is printed “lfi” this indicates that it’s a local file attack on the web server honeypot from the attacker IP address **192.168.178.103**.

```

716|2016-03-02 14:39:30|192.168.178.103:50155|/x?id=site1|unknown
717|2016-03-02 14:39:32|192.168.178.103:50156|/x?id=../../../../etc/passwd|lfi
718|2016-03-02 15:28:34|192.168.178.103:45194|/x?id=site1|unknown
719|2016-03-02 15:28:35|192.168.178.103:45195|/x?id=../../../../etc/passwd|lfi
sqlite>

```

Figure 6.4.2.6: Screenshot for the analysis of logs in sqlite3 database

The further logs can be accessed by the following command.

```
sqlite> select request_raw, filename, version, sensorid, pattern from  
events;
```

The following figure 6.4.2.7 displays the information of the host IP address **192.168.178.102** which is the IP of glastopf web server honeypot. Including the User-Agent used in the attack and the version used which are the **Python-urllib/2.7** and the type of attack which is “**lfi**”

```
Host: 192.168.178.102:8080  
User-Agent: Python-urllib/2.7||3.1.3-dev|37026c4f-4f57-46e4-8f19-372679720fd9|unknown  
GET /x?id=../../../../etc/passwd HTTP/1.1  
Accept-Encoding: identity  
Connection: close  
Host: 192.168.178.102:8080  
User-Agent: Python-urllib/2.7||3.1.3-dev|37026c4f-4f57-46e4-8f19-372679720fd9|lfi  
sqlite>
```

Figure 6.4.2.7: Screenshot of further analysis of the logs in sqlite3 database

7 Summary and Future Perspectives

This chapter describes brief summary for all the tests of the honeypots solutions. In addition, there will be proposal of the future perspectives, possible ways to extend the existing test cases and suggestion for improvements.

7.1 Summary

The whole analysis and deployment of the honeypot solutions on the single board computers can be summarized in the following way.

The first part includes the installation and testing of the virtual honeypot called as HoneyD. HoneyD is able to create 65535 virtual hosts on a single physical machine to fool the attackers with different personalities. These personalities act as a whole production system which in actually are the fake decoy systems placed on the network of the company to fool the attackers.

The functionality test made to check the proper functionality of the honeypot was deployment of the honeypot on the raspberry pi 2 and perform a brute force attack by the help of hacking tool hydra to hack the virtual host honeypot. As the attack was made to the HoneyD honeypot the honeypot started to log the activities and information received by the HoneyD honeypot. The functionality also includes that it is not possible to hack a fake system as it has no real existence. This will waste the time of the attacker in probing and attacking the system and on the same time gaining the information of the attacker to block him for the next time to attack.

The second part includes the installation and testing the functionality of a Kippo SSH server honeypot. Kippo server was deployed on Raspberri Pi 2 using Wheezy. To check the functionality a brute force attack was made by the help of a hacking tool Medusa. The kippo SSH honeypot was listening on port **22** which was scanned by the Nmap tool. Later a successful brute force attack was made to hack the SSH server and after gaining the access of the SSH server the functionality was shown by looking into the logs of the kippo SSH server.

One more functionality of the kippo SSH server is that it stores the replay logs of the remote session of the attacker's activity command to command which can be later analysed to figure out the interests displayed by the attacker into the fake kippo SSH server.

The third and the last part include the installation and deployment of the Glastopf web server honeypot on Raspberry Pi 2 using the wheezy image.

The functionality of the Glastopf includes that it serves some emulated vulnerabilities to the attacks made towards it. It stores different types of attacks which include SQL injection, Remote File Inclusion, Local File inclusion and provides the expected emulated result to the attacker to fool him.

The functionality test made was a brute force Local File Inclusion attack with the help of a python script made to perform a LFI attack. This attack was able to extract the credentials needed to hack the web server. Glastopf is intelligent enough to figure out the type of the attack with the type of the method used to perform the attack and provides the emulated result that was expected by the attacker.

The conclusion of this master thesis is all the three honeypot solutions used are well implemented on the single board computers and fully tested with the full functionalities. All the three honeypot solutions can play a good role in making the IT security more secure.

7.2 Future perspectives

This analysis and functionality testing is just the ground towards honey potting. Many things can be done with these honeypot solutions to make them more efficient and attractive to the hackers to attack.

Network security is a lot to do work where as honeypots is one of the solutions to make the network more secure.

The virtual honeypot discussed in this master thesis can be a more powerful and attractive tool. One can simulates HoneyD for arbitrary routing topologies which mean it can be configure for latency and packet loss.

Asymmetric routing can be done on HoneyD and integration of physical machines into topology is also possible.

Subsystem virtualization can run real UNIX applications under the virtual HoneyD IP addresses that can be a web server and ftp server etc. Using GRE tunnelling, HoneyD allows the making of distributed setups that let the HoneyD to extend to larger networks. It also grants virtual machines to be extend transversely separate addresses as GRE tunnel.

The future work which can be done in the kippo honeypot is one can monitor kippo with custom made scripts. Which means a user can see the graphical statistics from kippo by using kippo-graph utility. Kippo graph is a complete featured tool to demonstrate statistics in a kippo honeypot.

The next future work which can be done in the kippo honeypot is using a kippo-malware. This tool is just a python script that would store every infected file as a URLs in a kippo honeypot folder. This can be useful in places where one has missing their files or something incredible happened to the server but one have still their database intact.

One more thing can be done in future is in kippo a tool can be integrated named as pipal. This tool can be used for quick and easy analysis of password's tendency across the manually created passwords by the human. Pipal is a password examination tool that gives pertinent statistics of passwords give a password dump. One can use this tool to examine the passwords gathered by kippo.

The last part of this master thesis is the glastopf honeypot. The future job which can be completed in glastopf to formulate it more striking and industrious is by using its abilities to detect additional types of attacks.

Different types of network attacks can be performed on glastopf which includes SQL injection, Password Guessing, Cross-Site scripting and DoS attacks.

As the glastopf has a limited capability of responding to specific type of attacks, so by using this functionality one can provoke the attacker to send more information.

One can modify the glastopf to stores the statistics through irc channels or tweeter. One can advertise the vulnerable web pages, through indexing in Google for attracting the potential attackers.

8 Abbreviations

A

ARP Address Resolution Protocol

B

BFR Better Function Replacer

C

CLI Command Line Interface

COTS Commercial Off-The-Shelf

D

DHCP Dynamic Host Control Protocol

DSL Digital Subscriber Line

DNS Domain Name Service

DMZ Demilitarize Zone

DoS Denial of Service

E

ENISA European Union Agency for Network and Information Security

F

FARPD Fake ARP User Space Daemon

FIN Finish Flag

G

GUI	Graphical User Interface
GCD	Greatest Common Divisor
GRE	Generic Routing Encapsulation

H

HDMI	High Definition Multimedia Interface
HTTP	Hyper Text Transfer Protocol

I

IP	Internet Protocol
ICMP	Internet Control Message Protocol
IGMP	Internet Group Message Protocol
IDS	Intrusion Detection System
ISN	Initial Sequence Number
IETF	Internet Engineering Task Force
IT	Information Technology

L

LED	Light Emitting Diode
LAN	Local Area Network

M

MAC	Media Access Control
-----	----------------------

Master Information Technology

N

NAT	Network Address Translation
NIDS	Network Intrusion Detection System
NIC	Network Interface Card

O

OSI	Open Systems Interconnection
-----	------------------------------

P

POST	Power On Self Test
------	--------------------

R

RSA	Rivest Shamir Adleman
-----	-----------------------

S

SSL	Secure Socket Layer
SQL	Structured Query Language
SYN	Synchronization Flag
SSH	Secure Shell

T

TLS	Transport Layer Security
TCP	Transmission Control Protocol

U

UDP User Datagram Protocol
URL Uniform Resource Locator

Bibliography

- [1] Lance Spitzner. Honeypot: Catching the Insider 2003
 <https://www.acsac.org/2003/papers/spitzner.pdf>
 [Accessed on 10-Nov-2015]
- [2] Virtual Honeypots: From Botnet Tracking to Intrusion Detection 2007
 <http://books.gigatux.nl/mirror/honeypot/final/ch01lev1sec2.html>
 [Accessed on 15-Nov-2015]
- [3] [http://image.slidesharecdn.com/honeypots-140921055835-
phpapp02/95/honeypots-6-638.jpg?cb=1411279481](http://image.slidesharecdn.com/honeypots-140921055835-phpapp02/95/honeypots-6-638.jpg?cb=1411279481)
 [Accessed on 16-Feb-2016]
- [4] [http://image.slidesharecdn.com/honeypots-140921055835-
phpapp02/95/honeypots-7-638.jpg?cb=1411279481](http://image.slidesharecdn.com/honeypots-140921055835-phpapp02/95/honeypots-7-638.jpg?cb=1411279481)
 [Accessed on 16-Feb-2016]
- [5] [http://resources.infosecinstitute.com/tracking-attackers-honeypot-part-
2-kippo/](http://resources.infosecinstitute.com/tracking-attackers-honeypot-part-2-kippo/)
 [Accessed on 10-Dec-2015]
- [6] A dynamic, low-interaction web application honeypot: Lukas Rist
 2010
 [https://www.honeynet.org/sites/default/files/files/KYT-Glastopf-
Final_v1.pdf](https://www.honeynet.org/sites/default/files/files/KYT-Glastopf-Final_v1.pdf)
 [Accessed on 11-Dec-2015]

- [7] <https://github.com/mushorg/glastopf?files=1>
[Accessed on 18-Dec-2015]
- [8] A Virtual Honeypot Framework. Niels Provos 2003
<http://www.australianscience.com.au/research/google/1.pdf>
[Accessed on 13-Jan-2016]
- [9] <http://cdn.pollin.de/article/xtrabig/X702670.JPG>
[Accessed on 02-Mar-2016]
- [10] <http://books.gigatux.nl/mirror/honeypot/final/ch01lev1sec3.html>
[Accessed on 11-Feb-2016]
- [11] <https://www.concise-courses.com/security/what-is-hydra/>
[Accessed on 28-Jan-2016]
- [12] <https://www.concise-courses.com/hacking-tools/password-crackers/john-the-ripper/>
- [13] <https://www.concise-courses.com/hacking-tools/password-crackers/medusa/>
[Accessed on 27-Jan-2016]
- [14] <http://img.clubic.com/0320025801592960-photo-putty.jpg>
[Accessed on 14-Dec-2015]
- [15] <http://libdnet.sourceforge.net/>
[Accessed on 02-Jan-2016]
- [16] <http://libevent.org/>
[Accessed on 02-Jan-2016]
- [17] <https://launchpad.net/ubuntu/precise/+package/libdumbnet-dev>
[Accessed on 02-Jan-2016]
- [18] <https://www.rpmfind.net/linux/rpm2html/search.php?query=libpcap-devel>

- [Accessed on 02-Jan-2016]
- [19] <https://packages.debian.org/de/squeeze/libpcrc3-dev>
[Accessed on 03-Jan-2016]
- [20] <https://packages.debian.org/wheezy/libedit-dev>
[Accessed on 03-Jan-2016]
- [21] <http://freecode.com/projects/buildconf>
[Accessed on 04-Jan-2016]
- [22] <https://packages.debian.org/sid/net/farpd>
[Accessed on 04-Jan-2016]
- [23] <http://packages.ubuntu.com/de/precise/python-dev>
[Accessed on 13-Nov-2015]
- [24] <http://whatis.techtarget.com/definition/OpenSSL>
[Accessed on 13-Nov-2015]
- [25] <https://github.com/pyca/pyopenssl>
[Accessed on 15-Nov-2015]
- [26] <https://pypi.python.org/pypi/pyasn1>
[Accessed on 15-Nov-2015]
- [27] <https://pypi.python.org/pypi/Twisted>
[Accessed on 17-Nov-2015]
- [28] <http://www.gevent.org/>
[Accessed on 09-Dec-2015]
- [29] <https://launchpad.net/ubuntu/trusty/+package/libevent-dev>
[Accessed on 09-Dec-2015]
- [30] <https://docs.python.org/3/library/argparse.html>
[Accessed on 12-Dec-2015]

- [31] <https://pypi.python.org/pypi/chardet>
[Accessed on 12-Dec-2015]
- [32] <http://docs.python-requests.org/en/master/>
[Accessed on 14-Dec-2015]
- [33] <http://www.sqlalchemy.org/>
[Accessed on 14-Dec-2015]
- [34] <http://lxml.de/>
[Accessed on 14-Dec-2015]
- [35] <http://www.crummy.com/software/BeautifulSoup/bs4/doc/>
[Accessed on 14-Dec-2015]
- [36] [https://en.wikipedia.org/wiki/Pip_\(package_manager\)](https://en.wikipedia.org/wiki/Pip_(package_manager))
[Accessed on 15-Dec-2015]
- [37] <https://launchpad.net/ubuntu/precise/+package/python-dev>
[Accessed on 15-Dec-2015]
- [38] <https://en.wikipedia.org/wiki/Setuptools>
[Accessed on 15-Dec-2015]
- [39] <http://www.cprogramming.com/g++.html>
[Accessed on 15-Dec-2015]
- [40] [https://en.wikipedia.org/wiki/Git_\(software\)](https://en.wikipedia.org/wiki/Git_(software))
[Accessed on 15-Dec-2015]
- [41] <http://www.php-cli.com/>
[Accessed on 15-Dec-2015]
- [42] <http://packages.ubuntu.com/trusty/php/php5-dev>
[Accessed on 16-Dec-2015]
- [43] <http://packages.ubuntu.com/trusty/liblapack-dev>

- [Accessed on 16-Dec-2015]
- [44] <https://gcc.gnu.org/wiki/GFortran>
[Accessed on 19-Dec-2015]
- [45] <http://cython.org/>
[Accessed on 19-Dec-2015]
- [46] <https://launchpad.net/ubuntu/trusty/+package/libxml2-dev>
[Accessed on 20-Dec-2015]
- [47] <https://pkgs.alpinelinux.org/package/main/x86/libxslt-dev>
[Accessed on 21-Dec-2015]
- [48] <https://launchpad.net/ubuntu/trusty/+package/libmysqlclient-dev>
[Accessed on 21-Dec-2015]
- [49] <http://www.phpclasses.org/package/7015-PHP-Execute-external-PHP-scripts-in-a-separate-process.html>
[Accessed on 28-Dec-2015]
- [50] European Union Agency for Network and Information Security
<https://www.enisa.europa.eu/activities/cert/support/exercise/files/>
[Accessed on 31-Dec-2015]