MSc Project Report

Study on Web application

Honey pots

Submitted By

KAMALDEEP SEHGAL

[1135553]

Course	**:**	MSc Computer Science
Supervisor	**:**	**D**r. Ali Mansour
Year	**:**	2013

Thesis Author Consent Form

AUTHOR'S NAME: Kamaldeep Sehgal

TITLE OF THESIS: Study on Web application Honey pots

DEGREE: MSc Computer Science

Please read carefully and sign the following as appropriate.

I have read and understood the University's regulations and procedures concerning the submission of my thesis.

I understand that I have already signed a declaration agreeing to my dissertations being kept in the Learning Resources Centre (LRC) when I enrolled.

We would like now, to extend this agreement by making the thesis available online.

Further to this,

I AGREE AS FOLLOWS:
- That I am the author of the work.
- That I have exercised reasonable care to ensure that the Work is original, and does not to the best of my knowledge break any UK law or infringe any third party's copyright or other Intellectual Property Right.
- The LRC and BREO administrators do not hold any obligation to take legal action on behalf of the

Depositor (you), or other rights holders, in the event of breach of intellectual property rights, or any other right, in the material deposited.

I hereby extend my consent to this thesis being included in the LRC as well as on BREO via online access.

AUTHOR'S PERSONAL SIGNATURE: Kamaldeep Sehgal

AUTHOR'S STUDENT NUMBER: 1135553

DATE: 22/05/2013

Acknowledgement

It would be hard to complete my project without the guidance of several people who helped me to make my dissertation successful.

First of all, I wish to thank my supervisor Dr. Ali Mansour, who gave me permission to make the project on “Web application honey pot”. With his sincerity and encouragement, it is possible to complete my project on time. He gave me all possible information and suggestion to make and improve my project.

Moreover, I would like to thank “The University of Bedfordshire”, which gave me an opportunity to study here to improve my future. The Administration of the university is very helpful which encourage students to make their profession. It provides several services, which are useful for my study and project.

It is a pleasure to say thanks to my family and friends who support me all the time in my study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 Background

Honey pots are decoy systems that are used in the network to divert attacks from real systems and/or to study tools and techniques used by “blackhat hackers”. Hence, it is desirable that such systems be probed, attacked and compromised by cyber criminals. The level of interaction allowed by such systems helps us categorize the honey pots into two different categories viz. low interaction honey pot and high interaction honey pot.

Generally honey pots are designed to detect and report attacks against network and network systems like DNS, DHCP, DoS/DDoS etc. Such systems cannot detect and report web applications specific attacks like SQL injection, cross site scripting, command injection etc. To detect such attacks we need to construct and deploy a different kind on honey pot called web application honey pots. These honey pots can mimic a web-application or are themselves a vulnerable web application that allows attackers to attack them (Provos and Holz, 2008).

These days all systems are online and prefer to be accessed via the internet and web. Web applications are in high demand these days where people also prefer to do their finance access like banking and investments online or may be check their bank details or transfer money through bank using web. This makes the web application more attractive to the attackers who always look down for any information that might help them get unauthorized access to the resources which they can use for their own benefits (Honeynet Project, 2005).

These scenarios make the attackers to peep into the web applications and attack them which is growing day by day. There are also anti-hacking tools that are used against such attacks however they still do grow with new techniques and method. This is where honey pots come into the picture as they are then used as a decoy systems to trap the activities of the attacker.

The web application honey pots attract the attackers and also respond them well for the attacks they do, so that attackers try new techniques and methods to get the information from the web application. These web application honey pots then log all the attacks and techniques to its log and later on send them to the web administrators for review and learning the pattern of attacks. These implementations help the administrators to strategize their anti-hacking methods and incorporate them to the web applications making them more secure (Holz, 2007).

Web application honey pots are replica of the real application; it attracts the attackers to attack them. Identification of the attacks along with their patters is classified information. However zero vulnerability attacks that are new born attacks are sometime exceptions. This set of information help in building the defending strategies against such attacks. But, with the extensive range of attacks, attackers intrusion, variety of attacks and vulnerabilities combine together to make it complex to address all the problems. Moreover, new vulnerabilities make these things more difficult (Provos, 2004).

Honey pots are classified based on the their level of interaction with the outside world like,

- High level interaction,
- Medium level interaction, and
- Low level interaction honey pots.

Low level interaction honey pots do only limited interaction with the attackers and are one of the easiest honey pots to deploy and configure. These honey pots are designed to interact with the attackers and log the information which is later on sent to the administrators for study and analysis (Joho, 2004).

Medium level interaction honey pots provide somewhat more level of capabilities as compared to the low level interaction honey pot. In these honey pots when attackers do their attacks the application responds back to them with some bogus information (Spitzner, 2002).

On the other hand, high level interaction honey pots are highly interactive with the attackers and respond them well telling the attackers to do more attacks which helps the administrator later on to define their strategies to mitigate such attacks. They are also capable of capturing the traffic from the network and get extensive detailed information from the attackers skill level to psychology (Joho, 2004).

Interaction	Installation/configuration	Deployment/Maintenance	Information gathering	Risk
Low	Easy	Easy	Limited	Low
Medium	Involved	Involved	Variable	Medium
High	Difficult	Difficult	Extensive	High

Table 1: Tradeoffs of honey pot level of interaction (Spitzner, 2002).

Honey pot implementation came with an idea and approach of catching the enemies by understanding their weapons and then be ready for the tangible counter measures for protection. A Honey can be defined as a "security resource whose value lies in being probed, attacked or compromised" (Valli, 2007). There had been lot of research work that is being done over the honey pots to help defect different attacks on the web applications, services and networks based on the interaction level with the user.

Web application honey pots are those that collect the information about the various attacks on the websites classified into high and low interaction. Web honey pots of low interaction are designed for emulating the websites that host vulnerable web applications, however in case of web honeys of high interaction nature consist of real time vulnerable systems like decoy websites. It is complex to manage high interaction honey pots however on the other hand low interaction honey pots are easy to maintain as these systems does not execute malwares. In case of low interaction system they cannot be actually hacked by the attackers as they are just simulations but high interaction web application honey pots are capable of capturing much more information as they can execute malware for temporary time (Yagi, 2010).

Some of the existing web honey pot applications that exist are: High Interaction Honey pot Analysis Tool [HIHAT], Google hack honey pot, DShield web honey pot and PHP honey pot project. These honey pots implementations utilized the customized templates from the real time web applications so that they can pretend like them and attract the attackers and be vulnerable. These web applications that are set-up are always intentionally done for trapping the attackers and uses modified implementation of original/real web application. These implementations when attacked by the attackers do not impact the underlying original web application.

HiHAT (High interaction Honeypot Analysis tool) -

HIHAT is one of the high interaction honey pot which uses services like PHPMyAdmin, PHPNuke or OSCommerce converting PHP application to a high interaction honey pot. This tool provides a maximum level of interaction with an attacker as it provides complete functionality of the PHP services. This analysis tool is self capable of detecting known as well as new attacks. Due to its high interaction nature, this tool manages to extract detailed information of every attacks imposed on the web application (Mao, 2009). This HiHAT framework is PHP based and is physical honeypot that uses its real machine (i.e. its own IP address) information (Saat, et. al., 2007).

DShield web application honeypot:

DShields scripted in PHP and is one of the good web application honey pot project. This web application honey pot served as a very good system for web applications since it was smart enough to modify the templates and respond to the attackers. These features made it strong to be used and log many vulnerabilities and attacks. This web application honey pot basically comprises of 3 major elements called,

1) A client – setup the location for honey pot, a trap for the attackers.
2) Set of template(s) – these templates based on PHP is used to respond back to the attackers.
3) Logs – used for capturing the activities of the attacker on the web application. (DShield Web Honeypot Project, Anon).

Glastopf web application honeypot, is a low level interaction and dynamic web application honey pot and is capable to responding to many types of vulnerabilities so that it could gather data from different web application attacks that target them. It works

on a simple principle which says that respond back to the attacker in the best way so that he could exploit the web application more. Moreover, Glastopf also supports various forms of multi stage attacks, list of vulnerable attacks and vulnerable emulator (Vetsch, Kobin and Mauer, 2010).

Goggle hack honeypot, is now one of the outdated we application honey pots which utilizes the modified templates for the detection of the attacks. Since it had a lack of support from the community so it can be used only for capturing old attacks and it outdated as of now.

Some of the other honey pots are as follows,

1. **Specter,** is one of the low interaction honey pot used at production and detecting unauthorized activities is its purpose. This honey port monitor the IP address of the computer once installs and listens to the Transport Control Protocol (TCP) services. Specter manages to monitor 14 TCP ports. Whenever any attacker attempts to interact with any of the TCP services, specter is responsible for logging all the malicious activities and generates an alert to the owner (Huang et, al., 2003).
2. **Honeyd**, is also one of the popular low interaction honey pot which is widely used and can generate any simulated network along with many/multiple virtual honey pots inside a network using single host. This solution called Honeyd can be used for simulating different operating Systems like different versions of Windows, MAC, Linux/Unix or different type of network services like FTP (File Transfer Protocol), HTTP (Hypertext Transfer Protocol), SSH (Secure shell) etc. (William, 2006).

This kind of honey pots are deployed and used with the web applications to analyze the attacks in terms of malicious malwares or viruses injected through the web applications. Honey pots are deployed at the production servers to deflect the attacks towards a dedicated server who helps analyze, detect and log the attack behaviour and can also prevent Distributed denial of service attacks (Kieyzun, et al., 2009).

1.2 Problem Statement

These days web application are being extensively used over the internet which also indirectly gives the invitation to the attackers to hack the websites and retrieve confidential information and data. The vulnerabilities of the web application let the attackers intrude to servers or can even get inside the network environments. Attackers then can exploit the web sites with different attacks and can cause heavy damage to the owners of the website or the corporate holders. In this situation, the owners of the website have to stay updated all times so that they can fight against the vulnerabilities of the websites. However it also gets very difficult to protect the websites against zero-day exploits which are newly developed and invented.

It gets very important and critical to understand the different vulnerabilities and kind of attacks that an attacker can do on the web applications. Due to these problems, solution is required to detect the different kind of attacks the attackers can do and how to study them which could be helpful for preventing such attacks in the future. This is when the honey pots were developed for the web applications which would attract the attackers to do attacks and these honey pot application would also respond accordingly and log all the attacks that are performed by the attackers.

These honey pot applications are the simulation of the real time websites which attract the attackers. Now at this stage it gets very critical to understand the web application honey pots and how they respond to the attackers and are able to log all the simulations performed by them. These logs are then later on sent to the administrator who can analyze the attacks and apply the counter measures to the original web application. This is where it is important to study about the honey pots and understand their capabilities with strengths and weakness which would help the users to deploy the honey pots according to the evaluated data. This projects aims to study about the different aspects of web application honey pots available.

1.3 Objectives Of Research

Following are the list of objectives that will be accomplished during the project,

General Objectives

To study about the web application honey pots and their applications with critical evaluation.

Specific Objectives

1. Researching on the subject.
2. Study about the various web application honey pot technologies available.
3. Research about the web application honey pots with specific focus on two deployments.
4. Research about the various available solution of web application honey pots and their applications.
5. Perform deployments of two web application honey pots.
6. Evaluate and conclude the results on the web application honey pots.

1.4 Scope And Limitation Of The Research Work

This project will be done under a limited scope and would be performed on the virtual environments. Free available web application honey pots solutions will be used for deployments on the virtual machines to set up the environments. For simulating the different type of attacks on these honey pot deployments open source scanning tools like Arachini, W3AF, tools available in BackTrack would be used. This project would not use any licensed tools except VMWare that would be used for running the virtual machines.

The virtual machines will be Linux based which are available open source and all the deployments would be done on those images. The focus of this project would only be on 2 web application honey pots called DShield and Glastopf.

1.5 Thesis Organization

Chapter-1: Introduction

This chapter will be an introduction to the topic of the project covering the background, purpose, aims and objectives of the project i.e. a brief introduction about the web application honey pots available. This chapter will give an overview to the readers to understand the significance of honey pot, their implementation and applications in the current scenario.

Chapter-2: Literature Review

This chapter will give an overview on the literature of web application honey pots that would also reflect the previous work done on the web application honey pots. This research will be base subject that we will adopt to continue our research and use the elements as a base. This literature would also reflect the research and previous work that was done on them. This will help me in selecting and deciding about the web application honey pots that I will use to do my research and do their comparative analysis.

Chaper-3: Design of the System

This chapter will reflect the design of the desired system that will be implemented and tested for evaluating web application honey pots. This design will be using the virtual environments and all open source tools and environments necessary for implementing the web application honey pots.

Chapter-4: Implementation of the System

This chapter will have information about the implementation performed on the systems using a different web application honey pots. These implementation steps will be of high level and will show the technique and method by which these honey pots were installed, deployed and configured on the virtual environments.

Chapter-5: Experimentation

This chapter will show information about the experiments that will be performed on the implemented systems using a third party tools or comparative analysis methods. This

chapter would show the details of the different experiments performed on the system and record the output of the system behaviour in return.

Chapter-6: Analysis and evaluation of the experimental results

This chapter will show the results after the experiments and critical analysis on those results which would be evaluated. These results would provide an insight to the different web application honey pots demonstrating their strengths and weaknesses.

Chapter 7: Conclusions and future work

This is the final chapter that would comprise of the conclusion and work for the future which any one can use this results and research to extent theirs.

CHAPTER-2: Literature Review

The attack on web-applications has increased and constitutes 60% of the total attacks carried out on the internet (Rist, 2010). To provide countermeasure for this attack and to have proactive defence against well known web-application attacks like SQL injection, Cross Site Scripting, Local File inclusion etc. web-application honey pots can be deployed.

In previous work carried out by Anuar and other researcher, they presented architecture of a low interaction honeypot that can be used in production. It is called "Honeyd@WEB". This honeypot can be used in production as it is lightweight, secure and provides minimal interaction with the attacker. There is no risk of this honeypot being compromised and then being used a launching pad for other attacks. One of the striking features of this honeypot is that it can be managed via a web interface. The "honeyd@web" honeypot can mimic HTTP server and can be used to mimic a vulnerable web application server on the network. For example, it is possible to configure this honeypot to mimic a vulnerable Microsoft IIS 5.0 web-server. This is a very low level form of a web-application honeypot (Anuar, 2010).
The architecture of this honeypot is shown below:

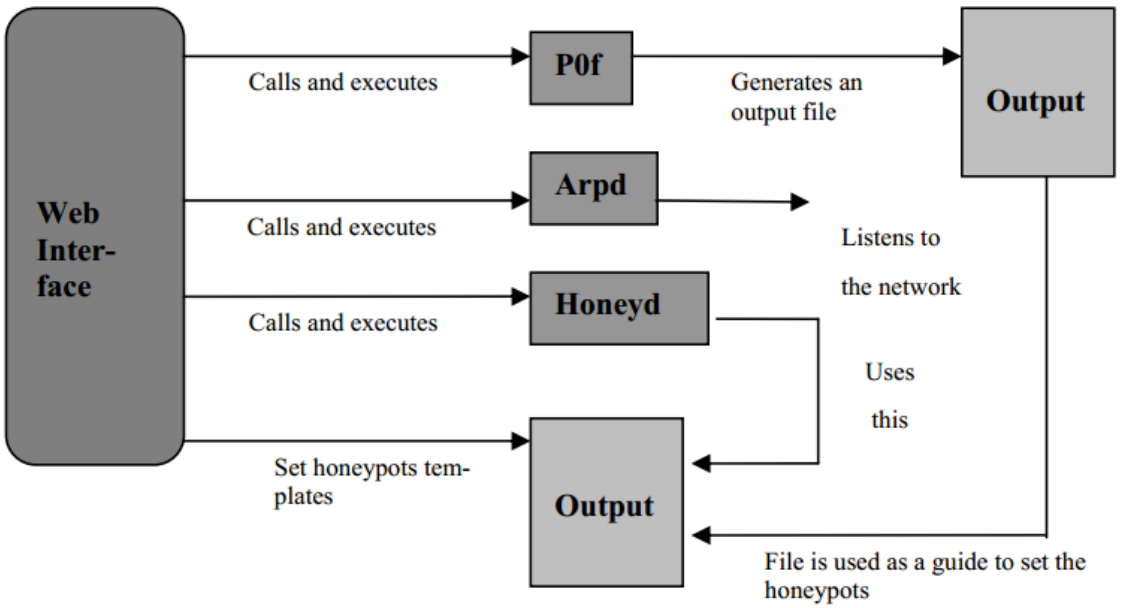


Figure 1: Honeyd@WEB Honeypot

All the components of the honey pot are well known open source tools (Anuar, 2010):

1. P0F: It is a passive operating system fingerprinting tool. It can sniff network traffic and examine the TCP/IP header information to determine which operating system has generated it.
2. Honeyd: It is well known light weight, low interaction honeypot that can mimic different services and servers like apache web server, Microsoft exchange server, Cisco router telnet service etc.
3. Arpd: This is an open source daemon that can provide fake arp replies in a network. It can be used to redirect traffic to a honeypot.
4. Web-Interface: It is a program written in PHP which can be used to manage the honeypot and deploy other honeypots.

The main component of the honeypot is the web interface that runs P0F utility to gather network information which is then used to deploy virtual honeypots in the production network. At the same time the web-interface calls arpd daemon to assign IP address and ARP response to the honeypot. The entire set of tools runs on the Linux/UNIX system. The advantage of having P0F in the tool set is that it can even detect subtle differences in the TCP/IP implementations of various operation systems and thus can generate a TCP/IP fingerprint that can mimic a vulnerable OS or service at the TCP/IP layer (Anuar, 2010).

The biggest weakness of this honeypot is that it cannot fully mimic a vulnerable web-application. It can just mimic a vulnerable web-server running a custom script which can fool a novice attacker but not an expert. Also, the honeypot doesn't change dynamically to adjust itself if there is any change in the network. For all this reasons, in the context of web-application honeypots , "honeyd@web" can be considered as a very basic and low-interaction honeypot (Anuar, 2010).

Previously researcher Sherif Khattab and his colleagues have used honeypots to protect network systems against distributed denial of service attacks (DDOS). The architecture they proposed can be applied to web-application security to protect legitimate web-applications against denial of service attack. Previously well known DDOS attacks were carried out against Amazon and Paypal. These attacks targeted the

web-application running on amazon and paypal and not the other server or services they hosted (Khattab et. Al, 2006)

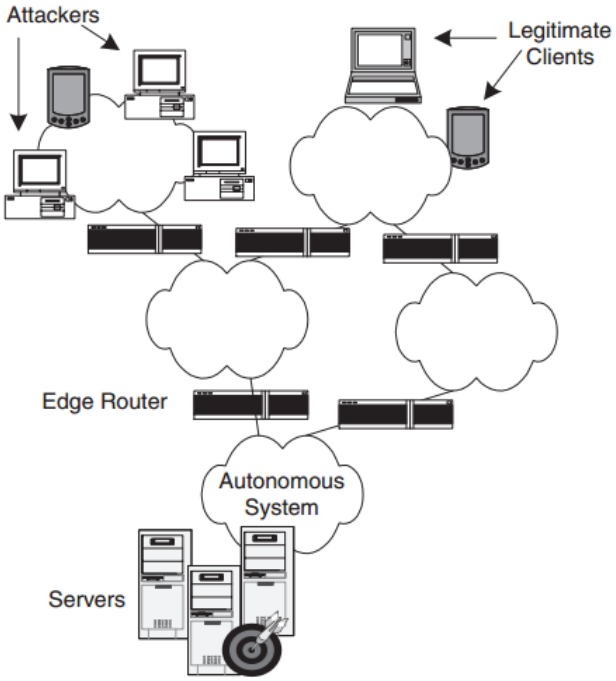


Figure 2 - Distributed DoS attack Protection

In the proposed architecture, to defend against DDoS attack, honeypots are installed among production servers, which act as a decoy. There can be one or many honey pots within the allocated server IP pool. The whole logic is that the traffic received by such decoy honeypots is most likely an attack packet. Also, these decoy honey pots are roaming honeypots i.e. their IP addresses and location keep on changing with time. This ensures that such honeypots cannot be tracked easily and makes their location unpredictable. In roaming honeypots, each server work as a honeypot for some periods of time, or honeypot epochs i.e. the time period of which is decided by a pseudo-random schedule joint among servers and legitimate clients. In the proposed architecture, a pool of honey pots is placed in different locations. These honey pots can be web-application honeypots running a vulnerable web-application or a low interaction honeypot mimicking a production web-application (Khattab et. Al, 2006).

In this design, it is difficult for an attacker to pin point the location of honeypots, as the honeypots are camouflaged within an IP address pool that belongs to a server block.

These honeypots can be replicas of the original servers that are chosen randomly to act as a honeypot but only for a selected time. Later, the IP belonging to the honeypot is released and is used by a legitimate server. In other words, all the servers in the IP pool are active, but some of them assume the role of honeypot and some of them assume the role of real servers. The time for which a server in the IP pool acts as a honeypot is called the honeypot epochs. The whole aim of this setup is to defend private cloud services against source-address spoofing DDoS attacks. The legitimate clients are always allowed access to the actual server and only malicious hackers engage with the honeypots. This architecture also implements hierarchical trace back scheme, which allows a security engineer to backtrack the IP address of the individual that is engaged with the honeypot. To achieve this each roaming honeypot when attacked initiates a recursive IP traceback to the attacker's IP address. All the Autonomous systems (AS) edge routers that are traversed, when tracing the attacker from the honeypot are notified about the DDoS attack. And consequently all the routers automatically identify and stop the packet flow from the attackers IP to the honeypot machine by inserting proper firewall rules. This scheme for aggressive trackback enables the honeypot system and the edge routers to identify and drop malicious DoS traffic while allowing legitimate traffic (business data) (Khattab et. Al, 2006).

The biggest challenge in this honeypot architecture is that the IP address of attackers belongs to multiple administrative domains and thus different AS numbers; it is difficult to co-ordinate the different ISPs to accept messages from decoy honey pots and then act according to it. To address this issue, the honeypot should have widespread deployment, and there should be close cooperation among ISPs (Khattab et. Al, 2006)

Web-Application honey pots that are deployed with this logic can indeed prevent DDoS attacks against the Web-Application farm. Also, the paper fails to address the fact that when messages are sent from decoy honey pots to the AS routers, the message should be encrypted, and the format of such a message is not clearly well defined. These messages should itself be protected against spoofing attacks.

One of the most difficult challenges faced during deployment of a high interaction web-application honeypot is to create one. Furthermore such web-application honey pots

might not be able to lure attackers who has tools and exploits written for known web-applications such as word press, drupal, joomla etc. To overcome this challenge Michael Muter and other researches created a generic toolkit that can convert any PHP application into a high interaction honeypot in an automated fashion. They also presented a method called transparent links which can be used to attract or lure attackers to such web-application honeypot. In their paper they introduced two new tools called Honeypot-Creator and HIHAT. The first tool Honeypot-Creator is a program that transforms an arbitrary web application into a high-interaction web-based honeypot. The program Honey pot-Creator is scripted in Java language and that is why it can run on any platform which has Java runtime set up. Such honeypots are created by analyzing the source code of the legitimate web-application and hence they can accurately mimic the behaviour of the original application while using only application level resources. The second tool is HIHAT (High Interaction Honeypot Analysis Tool) which is a PHP script. It helps a system administrator to analyze the log files that are created the web application honeypots (Muter, 2007).

Once the honeypots are created the next step is to lure attackers into the honeypots. In order to do so the authors recommend the usage of transparent links on web pages. A transparent link is a hidden link (written in HTML) and embedded in a web page which cannot be seen by a normal user. However, such links are used by search engines (google, yahoo and bing) to index a page. Now, when attacker uses a search engine like google to find a vulnerable target, google will display the honeypot as a potential target. This technique, when applied to google search engine is also called Google hacking. Using these tools and techniques the authors converted four known PHP applications existing applications PHPMyAdmin, PHPNuke, phphBB and PHPShell into high interaction honey pots and analyzed attacks on them.
(Muter, 2007)

They found out that the phpNuke got the highest number of successful attacks followed by phpShell, phpBB and phpMyAdmin (Muter, 2007).

Web-Honeypot Module	Number of Attacks	Percentage
phpNuke	45	64.29%
phpShell	13	18.57%
phpBB	7	10.00%
phpMyAdmin	5	7.14%

Table 2: Attacks as per web application modules

Also, they found out the most popular web application attack was SQL injection followed by File-Inclusion attack (Muter, 2007):

Name of Attack	Number of Attacks	Percentage
SQL-Injection	41	58.57%
File-Inclusion	13	18.57%
Command-Injection	8	11.43%
Directory-Traversal	3	4.29%
Others	5	7.15%

Table 3: Attacks percentage

In conclusion the research showed us the ways in which a known web-application can be converted into a honeypot and deployed in a production network. However, PHP based tools are in limit. Also, the HIHAT (High Interaction Honeypot Analysis Tool) cannot analyze logs that don't belong to apache web-server, which an again a significant short coming.

Another paper written by Takeshi Yagi focuses on data collection issues that plague the web-application honeypots. The biggest challenge in any honeypot system is the collection of attack logs. High interaction honeypots installed with an aim that the attackers will compromise the honeypot and launch attacks from it. Once compromised the attackers might install root kit, malware or backdoors on the web-application server or the web-application. Such conventional high-interactive web honey pots are only successful in collecting limited information about the attack, the attacker, root kits, malware or backdoors that install by the attacker. This is because, the URL scheme used for an attack does not match URL path structure served by the web honeypot. To solve this problem, Yagi proposes a scheme in which the destination URLs of attacks corrects by determining the correct path from the path structure of the web honeypot.

His investigation reveals that 97 percent of web-application attacks fail, but when the same attacks are used against the system designed, fifty percent of these attacks

succeed. This enables the user to gain much more information that can be used to protect websites than with conventional high-interaction web honeypots because one can collect all the information related with attacks and malware samples can be collected. In the proposed scheme, a HTTP forwarding function is installed between the web application honeypot and the internet. The HTTP forwarding function has three key components, a path analyzer, a cache table, and a conversion algorithm. The path analyzer checks the destination URL of an attack, and if the URL path is present in the web-application honeypot it allows the request to pass. However, if no such URL path exists on the actual web-application honeypot, then the HTTP forwarding function tries to change the path in the destination URL, to a correct path with the help of the cache table. A cache table basically has a list of attack path (URL) that are usually used by attackers and a corresponding entry which maps the attack URL to actual URL in the web honeypot. The HTTP forwarding function performs this conversion using conversion algorithm (Yagi et. al, 2010)

The schematic diagram of the same is given below (Yagi et. al, 2010).

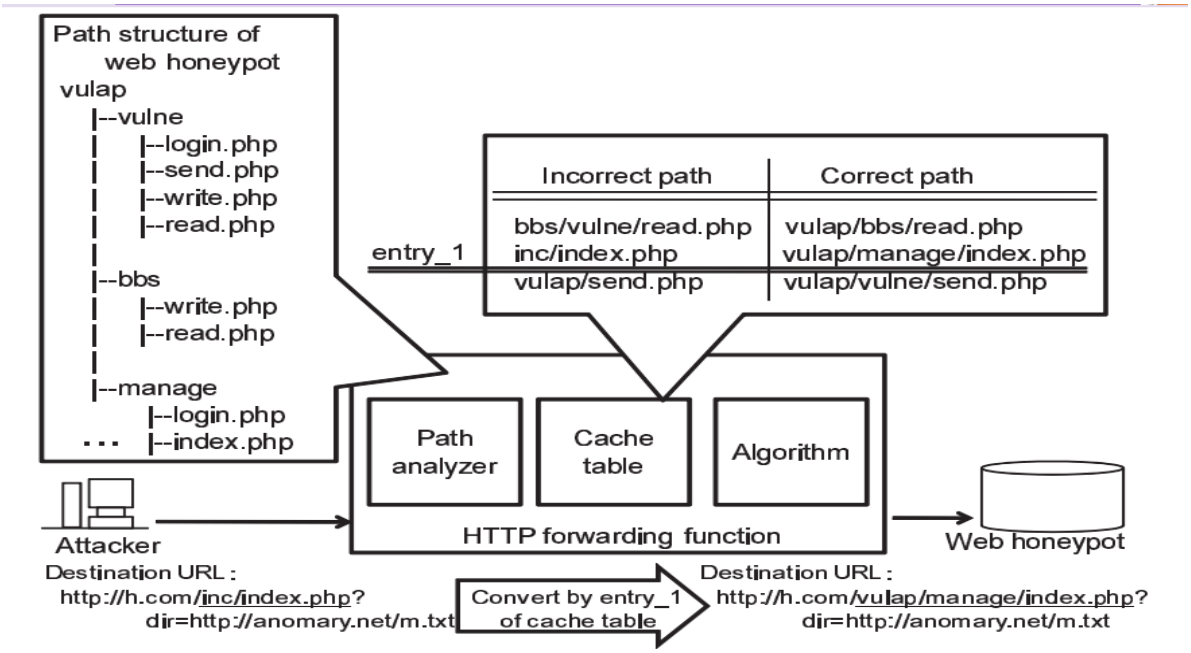


Figure 3 – Schematic diagram of the URL conversion function

They collected data about the attacks on web honeypots from the between January 30, 2009 and July 22, 2009. Using this scheme Takeshi Yagi and other researcher were

able to increase the number of successful attacks by almost 50%. The result is presented below:

	Total attacks	Naturally successful attacks	Successful attacks with proposed scheme
Number of attacks	1035	33	526
Different types of malware	63	14	45

Figure 4: Statistics about attacks and malware samples collected (Yagi et. al, 2010)

As it was seen that the number of successful attacks and the number of malware types that were collected after the scheme was used is very high in comparison to a number of attacks and number of malware types collected when conventional high interaction web-application honeypot was used.

Most of the web-application honeypots and honeypot deployment schemes focus on protecting e-commerce based web-application systems. Only few papers focus on honeypots emulating web-applications that are common in consumer electronic devices and industrial Supervisory Control and Data Acquisition (SCADA) systems. Jacob Farneth and other researches proposed a way to analyze and thus create a low interaction web-application honeypot that mimics the behaviour of a web-application running on an embedded system. They analyzed web server requests and responses and used it to develop emulation programs for low interaction honeypot that is running on an embedded system (Farneth, Dhanju & Jeremy, 2012).

The process of creating a web-application honeypot consist of three phase. The first phase involves using a web crawler to spider the web-application running on the embedded system to enumerate a comprehensive a list of resources on a web server. The open source Web scarab HTTP proxy tool is used to spider the web server resources and store the HTTP responses for both authenticated and unauthenticated sessions. In a second phase, a shifting program analyzes multiple responses from the web-server and constructs a list of command URL and query string parameters. The shifter program then performs a byte-wise comparison to determine which resources

are dynamic or static. The aim of this step is to keep the problem size as simple as possible. The third phase involves a fuzzer, which sends different input to the server for the same request and records the same. All the request and response are saved in a document that can be used to sketch a request to a response. Finally in phase four, the output of the fuzzer is analyzed through an enumeration to check the request and response pair. Finally, the request and response pairs are used to construct an emulation program. With these steps, the authors created an emulation program (honeypot) for the Novatech OrionLX substation automation platform which is a device that has a web server that is used for remote access and control of a range of SCADA equipment installed in electric substations. The resulting emulation program was able to mimic the original web-application by 99.5% (Farneth, Dhanju & Jeremy, 2012).

The paper entitled “Intrusion Attack Pattern Analysis and Signature Extraction for web Services Using Honeypots” by Nirmal Dagdee and Urjita Thakar propose a novel idea whereby attack signatures that are used by an Intrusion detection and prevention system (IDPS) are extracted by the help of web-application honeypots. These signatures are attack patterns that protect against malicious attacks against web services (Dagdee & Thakar, 2008).

For attack pattern analysis and automated signature creation the authors propose a system which has three components (Dagdee & Thakar, 2008):

1. Data Logging Component: This component in the architecture is responsible for logging attacks against honeypots which are running web-applications. For logging, the native honeyd server running as the honeypot web-server logs all the activity of the attacker at the application layer. Furthermore, the TCPDUMP packet capturing utility also runs on the honeypot system that captures the entire packet destined from an attacker to the honeypot. Generally the web service requests are in standard SOPA message format which can use HTTP, SMTP and FTP as underlying transport mechanism. The HTTP, SMTP and FTP messages are analyzed and filtered to extract the relevant SOAP messages. Furthermore, a Java based tool called Utilsnoop is run on the honeypot system that can intercept and log SOAP requests to the web service simulated by the honeypot system. It is configured so at the log all the SOAP requests to a file.

2. Data Analysis Component:

   The data analysis component analyzes the log file, SOAP requests and tcpdump pcap files to generate a list of URLs that accessed most frequently. The analysis system has a web-based interface which can list the IP addresses and target URLs that accessed frequently. The heart of this component is the shell script which analyzes the logs, SOAP requests and Tcpdump pcap files and populates a central database.

3. Signature Extraction Component:

   The signature extraction component works with the database that stores the logs of various attacks. A parser has written in Java or any other language that has strong string tokenization features used for parsing the database and creating a signature that can work with the Intrusion detection and prevention system (IDPS).

The final signatures that generated then can be feed into an Intrusion detection and prevention system (IDPS). The whole process can be automated so that the IDPS system has the signatures of the latest attack (Dagdee & Thakar, 2008).

One of the biggest drawbacks of this system is that different IDPS use different signatures. This raises interoperability issues between IDPS systems. The authors have generated signatures for the open source Snort intrusion detection system. It will take considerable time and effort to port those signatures to other IDPS systems.

In the paper entitled "Method for Two Dimensional Honeypot in a Web Application", the authors describe the use of web-application honeypots to protect an e-commerce system against automated bot attacks. A bot is a program that repeatedly requests resources from a web-site, sometimes resulting in denial of service attacks. In scenarios, when an e-commerce site propose hot inventory items where several traders are battle to get a limited supply items, bots can be employed by a trader to request multiple items and process unfairly to buy all the items. To protect against such attacks, different security mechanisms have employed in the past (Nassar & Miller, 2012).

[1] CAPTCHA :

Completely Automated Turing test to tell Computer from Human Apart [CAPTCHA] is a well known technique to make sure that a human and not a bot, is submitting a request to the web-appication. It basically challenges a user to enter a token (number or text) when submitting a request. However, the token is presented to the user in the form of a distorted image. When the challenge token matches the actual token presented in the form of the distorted image, then only a server processes the request. The figure below shows some CAPTCHA challenge images (Nassar & Miller, 2012) :

Figure 5: Captcha

[2] Form Honeypot :

This is a very old method of tracking an automated bot attack. In this technique, a HTML form contains additional form elements that are invisible to a user but visible to an automated bot script. Such fields are populated and submitted by the bot which allows a web server to determine that it was filled by a bot and not a human. The web server then automatically bands the IP from which the bot is operating. This method is not sophisticated and can be detected by making some changes to the bot script. The figure below shows the usage of form elements that are placed to lure a bot script.
(Nassar & Miller, 2012)

The authors present a modified version of the form honeypot concept to distinguish between humans and bots. This system also makes the detection of honeypot forms very difficult. Instead of placing invisible forms in a predictable manner and on the same

HTML form the authors recommend that the form fields (that are only filled by spiders and bots) to be generated randomly. Also, the authors recommend adding an extra from element in the page so that now there are forms that can only visible to bots and scripts and not human user. A governance module is created on the web-server that list and map which random field names are valid, and what real fields they mapped to (Nassar & Miller, 2012).

The proposed architecture is given below (Nassar & Miller, 2012):

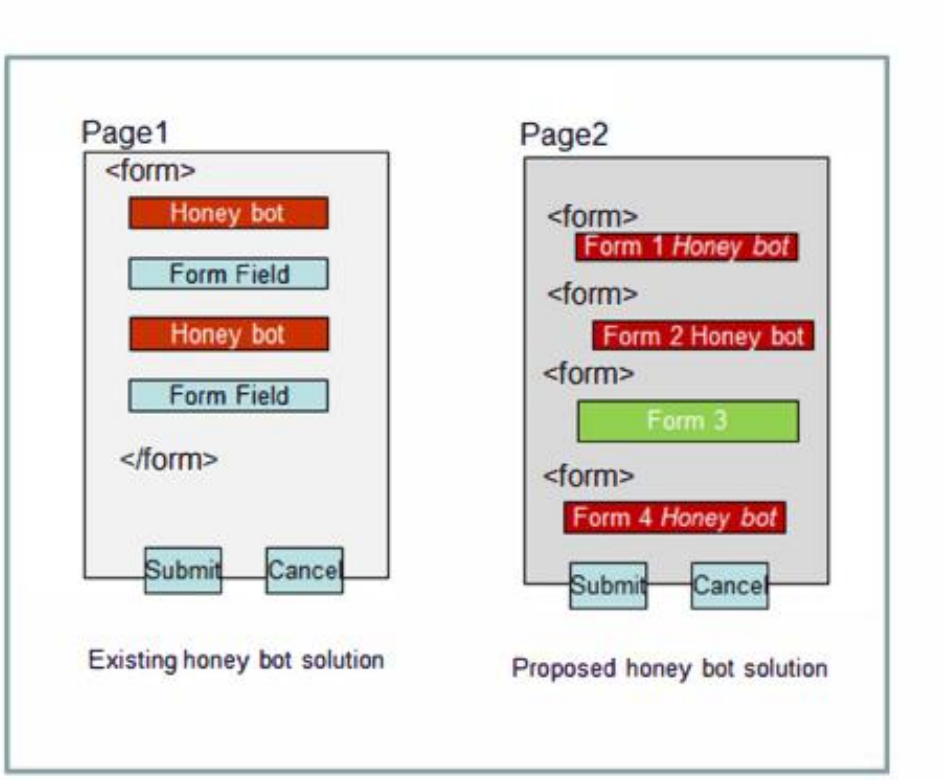


Figure 6: Honey pot solutions

It should be noted that only “Form 3” is filled by humans, whereas a bot fills all the forms or any of the existing forms marked in red. This technique can be very helpful in determining the bots and humans apart, using a form level honey pots components that enables an e-commerce site to protect against automate form submission or spidering attacks.

In November 2010, researcher Lukas Rist published a paper entitled “Glastopf A dynamic, low-interaction web application honeypot” and released a low interaction honeypot called glastopf that is now regarded as the most mature low interaction web-application honeypot. The honeypot is written in Python scripting language. According

to the author the older and known honey pots like HiHAT, DShield and google hack honeypot are all template based and doesn't provide multistage interaction to an attacker. Template based honey pots are those honey pots that server response to an attack using a predefined template. For example when an attacker scans existing honey pots for SQL injection attacks using a scanner, the older honey pots will behave in such a manner the scanner will report the honeypot to be vulnerable to SQL injection. However, when all the later stages of SQL injection attacks like dumping the password using SQL injection or writing arbitrary files using SQL injection is tried out against the honeypot, no relevant result will be handed out to the attacker. The inability of such older honey pots to further interact with attackers and emulate multistage vulnerabilities mean that the attacker will not indulge themselves in the attack for long and move on to the next target. To overcome this problem the glastopf honeypot is designed to respond to attacks in effective and efficient manner, such that multi-stage attack against it is possible. Furthermore, the honeypot is dynamic in nature and thus can emulate new vulnerabilities that are not known in public. To do this glastopf honeypot follows a simple principle, the honeypot doesn't care about specific vulnerabilities but instead focuses on what an attacker would see if that vulnerability was successful. If the attacker's systematic approach and his expectations are known then it is possible to emulate even a complex multistage attack (Rist, 2010).

For example when glastopf honeypot is attacked using a "Remote File Inclusion" script the URL of the attack may look like the following:

```
GET http://example.com/vulnerable.php?color=http://evil.com/shell.php
```

Here, the variable color is used to define a malicious file. When such RFI attack is seen it is then handled by a specific handler build to handle RFI attacks (the python handler is shown below) (Rist,2010) :

```
if '=http://' in request:
        handle_rfi_request()
```

After the type of attack has been identified the glastopf honeypot generates a response to simulate a successful attack. The glastopf honeypot uses PHP sandbox called pKaji PHP Sandbox to execute malicious PHP scripts. The honeypot parses the injected PHP files (in the above example it is shell.php) and runs the script in the PHP sandbox. The result of the execution is then sent back to the attacker. This makes sure that the attacker sees exactly what he would have seen when the code was executed by the real server (Rist, 2010)

On , July 2012 glastopf became the first honeypot to emulate real life SQL injections, it can now emulate all kinds of SQL injection and has a separate SQL injection emulator built into it (Ragan, 2012). This capability alone separates it from the rest of the honeypots.

In conclusion, it could be say that the glastopf honeypot is a unique low interaction honeypot that can mimic web-applications at very detailed level and produce real responses to attacks. This makes it an attractive choice for the security professional and researches.

Summary

This chapter discussed about the work done by other researchers. Next chapter reflects the design and system implementation of Glastopf and Dshield honeypots.

CHAPTER 3: DESIGN/FRAMEWORK OF SYSTEM

In this design, honey pots will be configured and deployed on the Linux environments which will be hosted on the virtual environments. These honey pots would then be attacked by simulating different exploitations by using open source scanning tools and then analyze the experimental results.

3.1 Design of DShield web application honey pot

DShield is one of the low interaction web application honey pot which is deployed and configured on the CentOS 6.4 Linux Server. Honey pot machine would contain the Apache web server running with PHP scripting engine hooked with the Logging engine. This honey pot deployment would have "Test" pages and would not contain any actual web site implementation. This way this system will be a zombie application to test the honey pots capabilities. This honey pot system will then be attacked using a third party software and tools (using BackTrack Linux) that would act as end user and simulate different actions on this honey pot system.

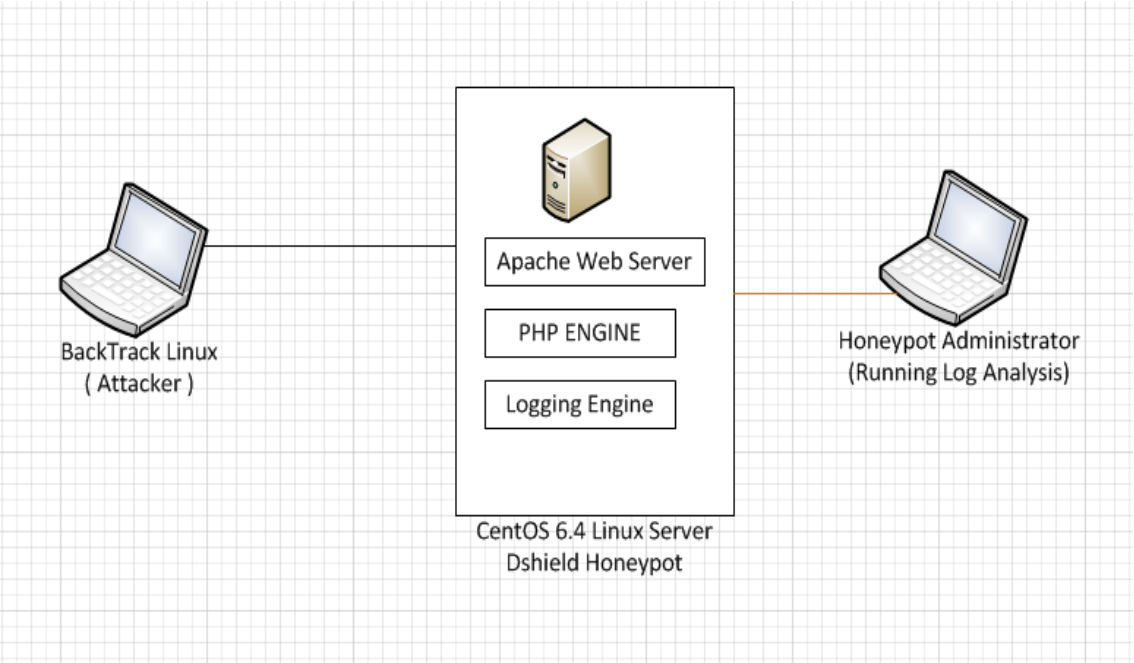


Figure-7: Test system design for DShield honey pot

Figure-7 shows the test demonstration for the Dshield system that would be tested. This system would also grab the logs that would be generated during the process by the logging engine that would be then analysed by the System administrator. This is the high level approach that would be followed to create a "Test System" for DShield.

3.2. Design of Glastopf web application honey pot

Glastopf is a low interaction web application honey pot that can effectively emulate a variety of web-application attacks including SQL injection. This honey pot is deployed and configured on a Linux server running Ubuntu 12.04. The Honey pot runs on top of python scripting engine and works closely with a PHP Sandbox which is also installed ion the Linux server. Glastopf has extensive logging capabilities and can log all aspects of interaction with the attacker. Since, it is a low interaction honey pot, compromising it is difficult, and hence it offers great security. The web-site served by the glastopf is rendered using Python code but is presented as a PHP application to the attacker.

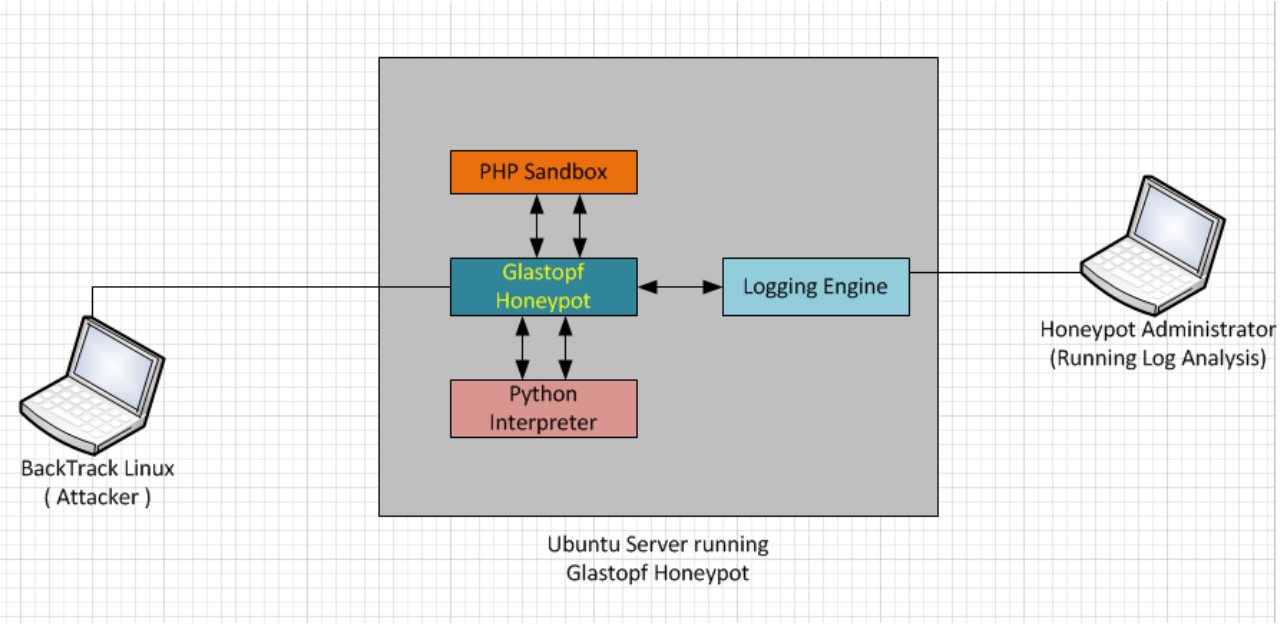


Figure-8: Test system design for Glastopf honey pot

Figure-8 shows the test demonstration for the Dshield system that would be tested. The honey pot system will then be probed and attacked using open source web-security tools that come pre-installed with BackTrack Linux or can be downloaded freely from the internet. Malicious, PHP code that will be injected by the attacker will be run inside

the PHP sandbox by the honeypot and the result will be shown to the attacker. This ensure that attackers sees what he wish to see i.e. the real output which is obtained when an attack is successful. These security tools will allow the attacker to simulate different attacks on the honey pot system. The logs that are generated during the attack would be then analysed by the System administrator. This is the high level approach that would be followed to create a "Test System" for Glastopf.

3.3. Attacks simulations on web application honey pot systems.

There are various types of attacks that will be simulated on the web application honey pots. All the attacks would be replicated by using a third party tools and software's and if required manual testing would also be performed to test the honey pot systems. These attacks simulation would be performed during the experiment phase of the project.

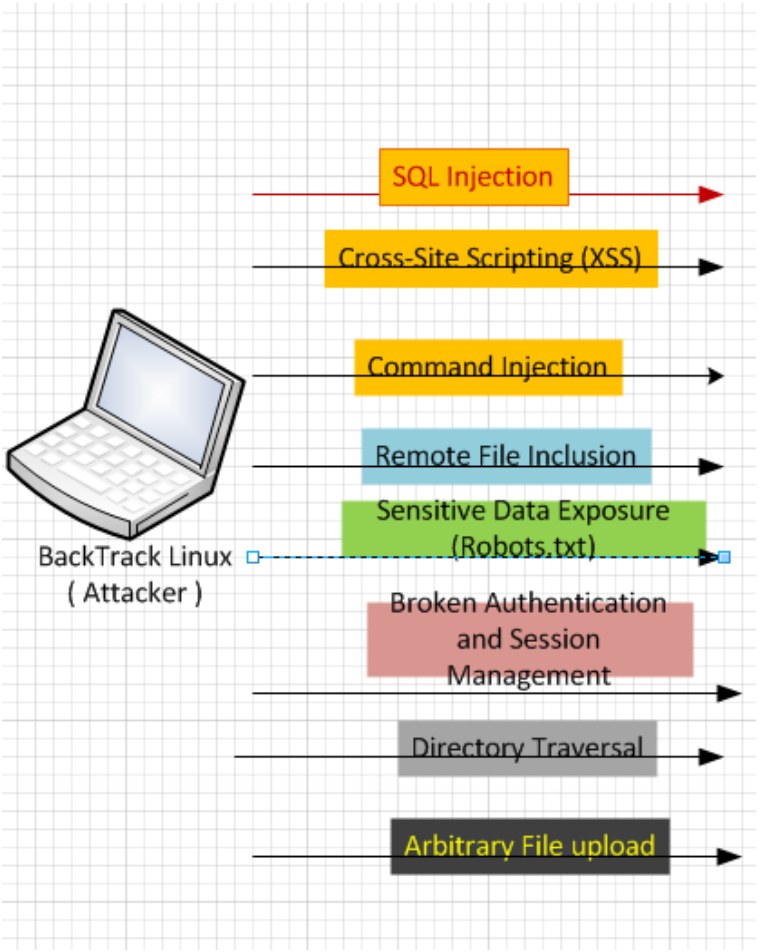


Figure-9: Security tests breakdown

Figure-9 shows the visual breakdown of the different simulations that would be performed on the web application honey pots to analyse their impacts during the process.

3.4 Design for attack simulation on Dshield honey pot system.

Figure-9 shows about the high level approach of the Dshield honeypot system under attack.

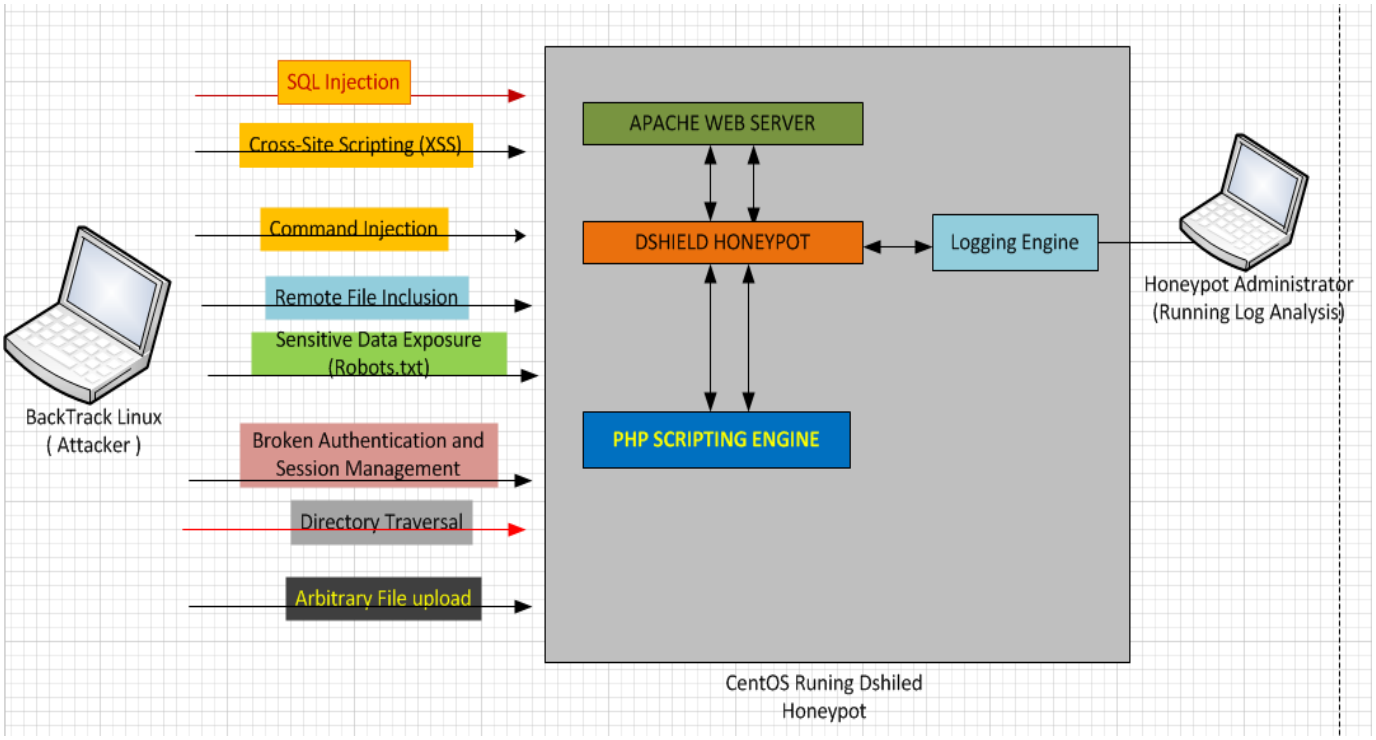


Figure-9: DShield honeypot under test

This system in run time would capture different logs as per the attacks that would be simulated using different penetration testing tools and software's. It would be expected from the DShield honey pot to respond back to the attacking machine (user) and all this data would then be captured by the logging system which is hooked up with the DShield honey pot system. These logs will then later on be analyzed using Log analysis tool like 'Apache Scalp' to verify the attack patterns were indeed correct.

3.5 Design for attack simulation on Glastopf honey pot system.

Figure-10 shows about the high level approach of the Glastopf honeypot system under attack.

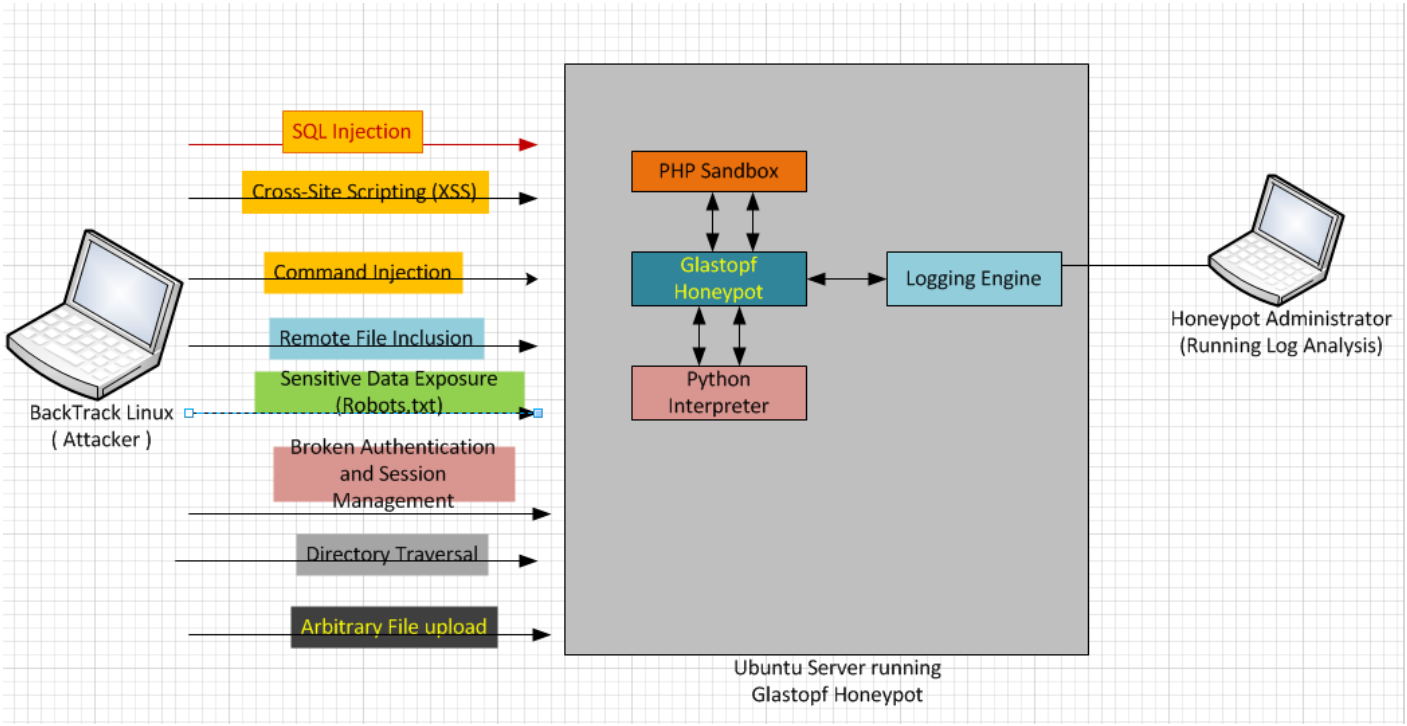


Figure-10: Glastopf honeypot under test

This system in run time would capture different logs as per the attacks that would be simulated using different penetration testing tools and software's. It would be expected from the Glastopf honey pot to respond back to the attacking machine (user) and all this data would then be captured by the logging system which is hooked up with the Glastopf honey pot system. These logs will then later on be analyzed to verify the attack patterns were indeed correct and capture the results after experiment.

3.6 Summary

This chapter explained about the design of the system that used to deploy the honey pots for Glastopf and Dshield. Next chapter would reflect the implementation of the system that used for deployment of these systems.

CHAPTER 4: SYSTEM IMPLEMENTATION

For the purpose of experiment, honey pots deployed as a web application on the Linux servers. All the deployments and configuration done on the virtual machines, and these honey pots accessed as a web application running on the servers which is a similar replica as will be considered during the actual attacks by the attackers where attackers will see the web application and then attack it finding it vulnerable. These experiments were based on the designed approach of the systems, as mentioned in the chapter-3 above.

4.1 Implementation of Dshield web application honey pot

This section displays the implementation of the Dshield web application honey pot:

4.1.1 Installation and configuration process of Dshield web application honeypot

1. Download and install the minimal version of Centos 6.4 from the following location :
2. Once the OS installed, the next step is to install the Apache web server with PHP support in the centos. This can be done using the following commands:

```
# yum install -y httpd php
# chkconfig httpd on
```

3. Download the latest version of DShield honeypot in the following location :
   # mkdir /webdir/
   # cd /webdir/
   # wget –c
   'https://sites.google.com/site/webhoneypotsite/downloads/webhoneypot-alpha.tgz?attredirects=0&d=1'

4. Untar the source code using the following commands :
   # tar -zxvf webhoneypot-alpha.tgz

This creates a new directory called web-honeypot which has the following directory structure:

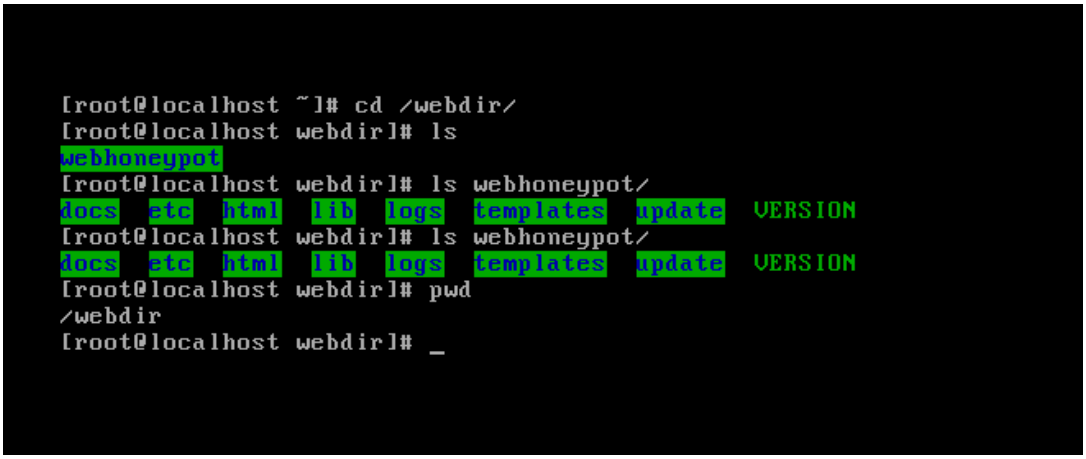


Figure-11: Dshield Honey pot installation

5. Once, the source code has been installed in the proper directory location. The next step is to configure apache for virtual hosting. Since, the web-honeypot can be only be accessed using an apache virtual domain, a virtual domain called www.example1.com is added to apache configuration. The following lines are added to the apache configuration file at the end:

```
NameVirtualHost *:80
#
# NOTE: NameVirtualHost cannot be used without a port specifier
# (e.g. :80) if mod_ssl is being used, due to the nature of the
# SSL protocol.
#

#
# VirtualHost example:
# Almost any Apache directive may go into a VirtualHost container.
# The first VirtualHost section is used for requests without a known
# server name.
#
<VirtualHost *:80>
    ServerAdmin webmaster@example1.com
    DocumentRoot /webdir/webhoneypot/html
    <Directory "/webdir/webhoneypot/html">
        RewriteEngine on
        RewriteRule ^(?!index.php)(.*) index.php/$1
   </Directory>
    ServerName www.example1.com
    ErrorLog logs/dummy-host.example1.com-error_log
    CustomLog logs/dummy-host.example1.com-access_log common
</VirtualHost>
[root@localhost ~]# _
```

Figure-12: Dshield Honey pot configuration

The most important lines in the configuration are the following lines:

```
<Directory "/webdir/webhoneypot/html">
    RewriteEngine on
    RewriteRule ^(?!index.php)(.*) index.php/$1
</Directory>
```

Using the mod_rewrite engine the apache web server configuration redirects all requests to the index.php file located in the /webdir/webhoneypot/index.php script.

6. Once the honeypot configured the apache server restarted using the following command :

   # service httpd restart

   Now the web application honeypot can be accessed from Windows Machine by adding the following lines to the hosts file, where 192.168.79.129 is the IP address of the honeypot machine:

   192.168.79.129 www.example1.com

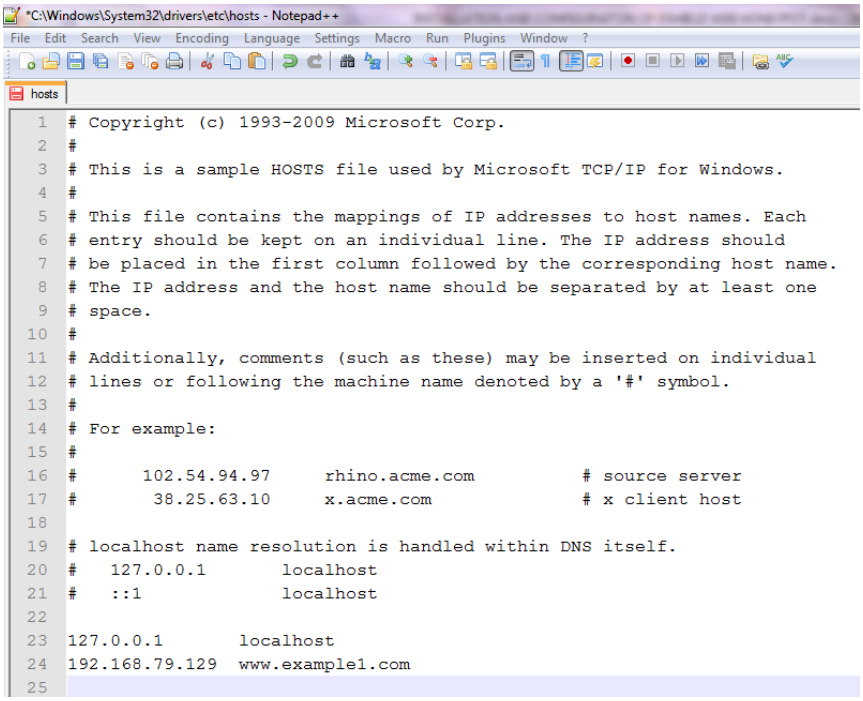


Figure-13: Setting Apache configuration *(Pinda, 2012)*

4.1.2 Configuring the logging process in DShield honey pot

The apache server was configured to redirect to logs to a separate file using the following directives:

</Directory>

ServerName www.example1.com

ErrorLog logs/dummy-host.example1.com-error_log

CustomLog logs/dummy-host.example1.com-access_log common

Logs for the honey pot will now be directed into two separate files:

a. **The error logs** will be placed in the following location:
/var/log/http/dummy-host.example1.com-error_log

b. **The access logs** (HTTP request) by the clients will be placed in the following location: /var/log/http//dummy-host.example1.com-access_log common *(Pinda, 2012)*

Log Analyzer:

Apache-Scalp utility which can analyze the Apache logs for attack pattern using IDS signatures from PHP-IDS project is used to analyze the logs against hacking attack. Download the python script from https://code.google.com/p/apache-scalp/downloads/detail?name=scalp-0.4.py and filter xml file from https://code.google.com/p/apache-scalp/ (please also make sure to install the "python.lxml" package).

4.2 Implementation of Glastopf web application honey pot

This section displays the implementation of the Glastopf web application honey pot:

4.2.1 Installation and configuration process of Glastopf web application honeypot

1. Install the latest version of ubuntu server edition from
http://www.ubuntu.com/download

2. Install OpenSSH server using the following command:

aptitude install -y openssh-server

3. Install the required dependencies needed by glastopf honeypot which includes python openssl library, php5 development libraries, C and C++ compilers:

```
sudo aptitude update
sudo aptitude install python2.7 python-openssl python-gevent libevent-dev python2.7-dev build-essential make python-chardet python-requests python-sqlalchemy python-lxml python-beautifulsoup mongodb python-pip python-dev python-numpy python-setuptools python-numpy-dev python-scipy libatlas-dev g++ git php5 php5-dev
sudo pip install --upgrade distribute
```

4. Once the dependencies are installed the next step is to install the PHP sandbox which can run PHP applications inside a sandbox. The generated Zend :

```
cd /opt
git clone git://github.com/glastopf/BFR.git
cd BFR
phpize
./configure --enable-bfr
make && make install
```

The last command generates the following zend extension and installs it in the following location :

```
/usr/lib/php5/20090626+lfs/bfr.so
```

The php.ini file needs to be updated so that php engine (zend) uses the generated extension:

```
# vim /etc/php5/apache2/php.ini
zend_extension = /usr/lib/php5/20090626+lfs/bfr.so
```

4. The next step is to download and install the latest version of glastopf honeypot from its GIT repository:

```
cd /opt
git clone https://github.com/glastopf/glastopf.git
cd glastopf
python setup.py install
```

This installs the glastopf honeypot in the following location :

/usr/local/bin/glastopf-runner

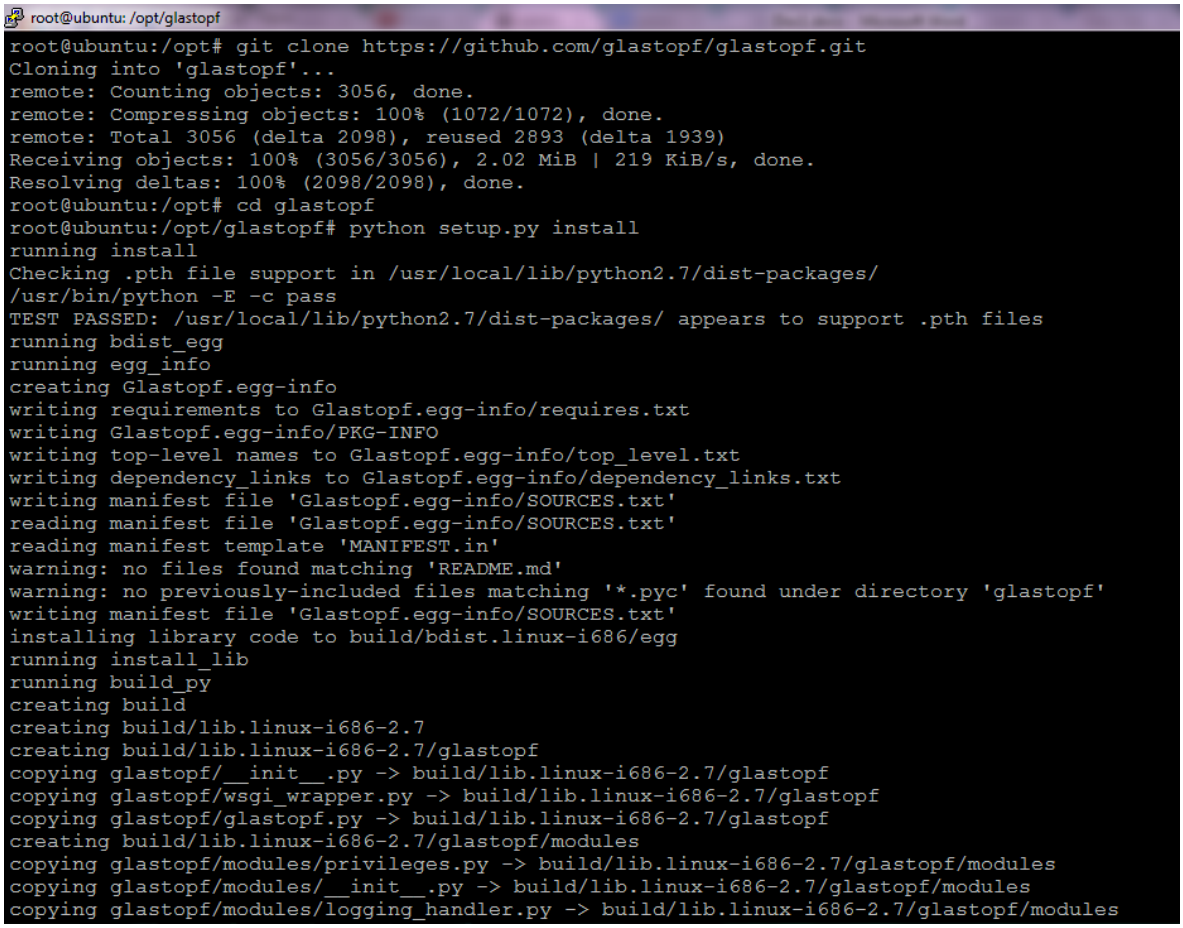


Figure-14: Installation of Glastopf

6. The final step in the installation is to generate the glastopf configuration file and glastopf environment :

```
cd /opt
mkdir myhoneypot
cd myhoneypot
glastopf-runner.py
```

Once, the environment is setup the next step is to start the glastopf honeypot using the following commands :

```
cd /opt
mkdir myhoneypot
cd myhoneypot
glastopf-runner
```

By default glastopf honeypot listens on tcp port 80 on all interfaces :

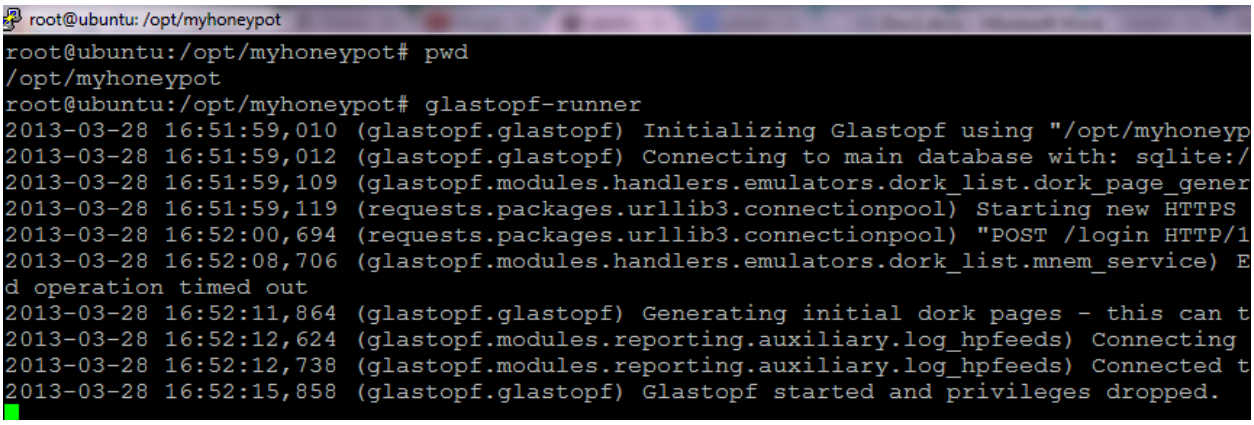


Figure-15: Glastopf runner (Rist, 2012)

4.2.2 Configuring the logging process in Glastopf honey pot

The Glastopf server was configured to redirect to logs to a separate file using the following directives in /<install-directory>/glastopf.cfg :

[logging]

consolelog_enabled = True

filelog_enabled = True

logfile = log/glastopf.log

The Glastopf server was also configured to redirect to logs to a separate sqlite database using the following directives in /<install-directory>/glastopf.cfg :

[main-database]

#If disabled a sqlite database will be created (db/glastopf.db) to be used as dork #storage.

enabled = True

connection_string = sqlite:///db/glastopf.db

Logs for the honey pot will now be directed into two separate files :

a. **The error logs** will placed in the following location :

/<install-directory>/log/glastopf.log

b. **The database for the logs** would also be logged in the sqlite database file that will be placed in the following location: /<install-directory>/db/glastopf.db

(Enisa, 2012)

Log Analyzer:

The above generated log file and db file can be analysed by either manually analyzing the log file or querying the sqlite db file using SQLite3 utility.

(please also make sure to install the "sudo apt-get install sqlite3 libsqlite3-dev" package).

4.3 Implementation of Attacking process

For simulating the attacking process, a back track image was used which contains lot of penetration testing tools to simulate different actions and attacks over the web application. Following are the list of tools that would be used during the experiment,

4.3.1 W3AF

W3af stands for Web Application Attack and Audit Framework, it is an open source web application vulnerability and exploitation tools written in python. It can scan a web-site for vulnerabilities like SQL injection, local file inclusion , cross site scripting etc. and exploits them to give access to the underlying database or operating system.

W3af follows three steps to audit and exploit a web target.

The first step is scanning, using scanning plug-ins like “crawl-plugin" and “web-spider” w3af will firstly try to identify all the pages and URLs that are present in the web-application. Also, for each URL, the various HTML forms and HTML query string parameters in the URLs are identified. W3af recursively follows all links extracts URLs those links. At the end of this process W3AF will have an accurate application map with various links in it.

The second step is application auditing, this is done using audit_plugins. The audit plugins will used the input from the first phase and then fuzz all the URLs, query string and forms to find various vulnerabilities like SQL injection, Cross Site Scripting etc. When a bug is found it is logged and can be reported to the user. One of the most sophisticated audit plugin is sqli which can find error-based SQL injections.

Finally, the identified vulnerabilities, error messages and all other logs that are generated is formatted and stored in HTML , XML and text files for users using plugins called output plugins (Riancho , 2012)

4.3.2 Nikto

Nikto is a web –application security scanner written by Chris Sullo of the Open security foundation, it is written using perl programming language. Nikto is an Open Source GPL licensed web server and web-application scanner that performs multiple tests against a web servers and scans for multiple vulnerabilities. It also scans for vendor specific web server vulnerabilities for known and outdated versions of web-servers. It can also check for server mis-configuration and open configuration items such as the presence of multiple index files, HTTP server options, and will attempt to identify installed web servers and software. Some of the features of Nikto are as follows :

1. SSL support using OpenSSL on Linux.
2. Mutation techniques to check and bypass IDS and IPS.
3. Supports HTTP authentication (Basic and NTLM)
4. Checks for outdated server version and information disclosure vulnerabilities.
5. Can log to metasploit which can then be used to exploit the discovered vulnerabilities.

(Sullo, 2010)

4.4 Summary

This chapter explained about the implementation of the system that was used to deploy the honey pots for Glastopf and Dshield. Next chapter would reflect the different experiments that were performed on the system.

CHAPTER 5: EXPERIMENTATION

In this chapter, the implemented web application honey pots will now be tested and experimented based on different inputs which would be simulated using automated scanning tools to replicate the attacks on the honey pots and then analyse them from the logs to verify what does these honey pots log and how are their configurations used.

5.1 Supported Attacks by Dshield and Glastopf

Dshield and Glastopf were designed and implemented for a purpose. As these are low interaction honey pots they support limited functionalities and are extended based on the development done by their respective communities. Following is the list of attacks that Dshield and Glastopf support,

S.No	Attack Type	Supported by Dshield	Supported by Glastopf
1	SQL Injection	No	Yes
2	Cross-Site Scripting (XSS)	Yes [limited]	Yes
3	Command Injection	Unknown	Yes
4	Remote File Inclusion	Yes	Yes
5	Local File Inclusion	Yes	Yes
6	Sensitive Data Exposure (Robots.txt)	Yes	Yes
7	Directory Traversal	Yes	Yes

* Unknown- due to limited documentation on DShield it is not specified that this honey pot supports the corresponding attacks.

Table-4: Supported attacks by DShield and Glastopf (Rist, 2010)

Table-4 demonstrates about the supported attack simulations that DShield and Glastopf web applications are designed for. These are the list of common attacks that an attacker would be able to simulate on the web application honey pots which will be recorded/logged by these honey pot systems.

5.2. Experiment and evaluation test performed on DShield and Glastopf

To perform the experiments on the Dshield and Glastopf environments an attacker was required which is the reason automated security scanners were used against the deployed honey pots to replicate the different scenarios and analyse the logs that were generated as a result of simulating the attacks. Use of the automated scanners eased the process of simulating different attacks on the honey pots. This process was much more efficient and has a close relation to the attacker where he/she can use different tools to replicate the attacks and try to find the vulnerabilities in the web applications.

Test 1:

- Sensitive Data Exposure (Robots.txt)

Tools used & their results:

- Execute the command "wget <IP ADDR OF HONEYPOT>/robot.txt"

Result to end user:

a. Glastopf: Attack was successful and the file robots.txt was returned with the following data:

User-agent: *

---

b. Dshield: Attack was successful and the file robots.txt was returned with the following data:

User-agent: *

Disallow: /admin

Disapply: /phpadmin_secret

Reason for selection of this tool for emulation from attackers prospective:

- wget is one of the most commonly used HTTP client in Linux platform. This attack prints sensitive information about the web-server and is the first attack used before other attacks are launched. Attacker now has further web directories to scan and/or probe and find weaknesses in it.

Reflection of results in web application honey pots:

a. Dshield: It was observed from the logs that the attacker attempted to get robots.txt file on Dshield web application honey pot. In this case attackers host machine is shown with IP address 192.168.50.155.

```
[root@localhost httpd]# grep robots.txt dummy-host.example1.com-access_log
192.168.50.155 - - [07/Apr/2013:19:09:35 +0000] "GET /robots.txt HTTP/1.1" 200 6
0
192.168.50.155 - - [27/Apr/2013:22:02:09 +0000] "GET /robots.txt HTTP/1.0" 200 6
0
192.168.50.155 - - [27/Apr/2013:22:02:22 +0000] "GET /robots.txt.1 HTTP/1.0" 404
 13695
```

Figure-16: Dshield log analysis

b. Glastopf: sqlite logs all the attacks in the sqlite.db which can be accessed using the sqlite3 command line utility. The attack class (pattern) is robots and is extracted using the following sqlite query :

[sqlite3> select * from events where pattern='robots';]

```
sqlite> select * from events where pattern = 'robots';
14943|2013-04-07 13:16:13|192.168.50.155:37019|GET|/robots.txt||HTTP/1.1|{"Connection": "Keep-Alive", "Host": "192.168.50.162", "User-Agent": "Mozill
a/5.00 (Nikto/2.1.5) (Evasions:None) (Test:robots)"}||robots||HTTP/1.1 200 OK
Connection: close
Content-Length: 23
Content-Type: text/html; charset=UTF-8

User-agent: *
Disallow:
15313|2013-04-27 15:07:29|192.168.50.155:53918|GET|/robots.txt||HTTP/1.0|{"Connection": "Keep-Alive", "Host": "192.168.50.162", "Accept": "*/*", "Use
r-Agent": "Wget/1.12 (linux-gnu)"}||robots||HTTP/1.1 200 OK
Connection: close
Content-Length: 23
Content-Type: text/html; charset=UTF-8
```

Figure-17: Glastopf log analysis

Test 2:

- SQL Injection

Tools used & their results:

- SQLMap: Execute the command

./sqlmap.py --url 'http://<web-application honeypot>/index.html' --data='pma_username=test&pma_password=test&server=1&lang=en-iso-8859-1&convcharset=iso-8859-1' -p 'pma_username' --level 5

Result to end user:

a. **Dshield:** This web application honey pot failed to emulate against sql injection attack however was able to capture the continuous activities from the attacker.

b. Glastopf: This is the running instance of SQLMap:

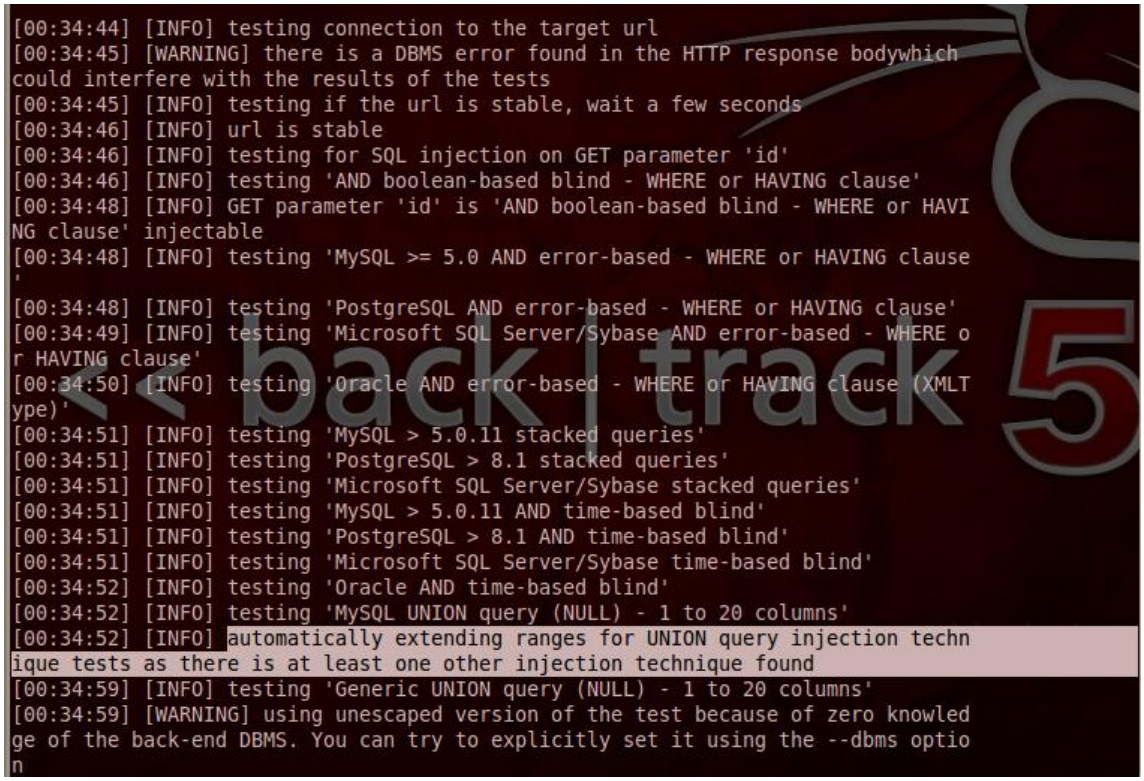


Figure-18: SQLMap scanning

Reason for selection of this tool for emulation from attackers prospective:

- sqlmap is one of the most sophisticated sql injection tool available on the market. It support multiple databases including oracle, mysql, mssql and postgresql and support multiple sql injection techniques including boolean based, time based, union and blind sql injection.

Reflection of results in web application honey pots:

a. Dshield: Dshield can't emulate sql injection vulnerability at all. This means that it can't interact with the attacker when he/she is using sql injection tools/techniques.

```
[root@localhost httpd]# grep POST dummy-host.example1.com-access_log
7559
192.168.50.155 - - [27/Apr/2013:23:01:17 +0000] "POST /index.html HTTP/1.1" 200
7559
192.168.50.155 - - [27/Apr/2013:23:01:17 +0000] "POST /index.html HTTP/1.1" 200
7559
192.168.50.155 - - [27/Apr/2013:23:01:17 +0000] "POST /index.html HTTP/1.1" 200
7559
192.168.50.155 - - [27/Apr/2013:23:01:17 +0000] "POST /index.html HTTP/1.1" 200
7559
192.168.50.155 - - [27/Apr/2013:23:01:17 +0000] "POST /index.html HTTP/1.1" 200
7559
192.168.50.155 - - [27/Apr/2013:23:01:17 +0000] "POST /index.html HTTP/1.1" 200
7559
192.168.50.155 - - [27/Apr/2013:23:01:17 +0000] "POST /index.html HTTP/1.1" 200
7559
192.168.50.155 - - [27/Apr/2013:23:01:17 +0000] "POST /index.html HTTP/1.1" 200
7559
192.168.50.155 - - [27/Apr/2013:23:01:17 +0000] "POST /index.html HTTP/1.1" 200
7559
```

Figure-18: Dshield results on SQL injection

b. Glastopf: On the other hand glastopf can partially emulate sql injection vulnerability and can be useful for luring attackers using sql injection tools like sqlmap. At the same time glastopf provide full log including the URL of attack.

```
2013-04-27 16:40:52,594 (glastopf.glastopf) 192.168.50.155 requested GET /wp-content/themes/presstige/wp-content/themes/presstige/timthumb.php?id=23%
27%20UNION%20ALL%20SELECT%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL--%20 o
n 192.168.50.162
2013-04-27 16:40:52,868 (glastopf.glastopf) 192.168.50.155 requested GET /wp-content/themes/presstige/wp-content/themes/presstige/timthumb.php?id=23%
27%20UNION%20ALL%20SELECT%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20N
ULL--%20 on 192.168.50.162
2013-04-27 16:40:53,216 (glastopf.glastopf) 192.168.50.155 requested GET /wp-content/themes/presstige/wp-content/themes/presstige/timthumb.php?id=23%
27%20UNION%20ALL%20SELECT%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20N
ULL%2C%20NULL--%20 on 192.168.50.162
2013-04-27 16:40:53,598 (glastopf.glastopf) 192.168.50.155 requested GET /wp-content/themes/presstige/wp-content/themes/presstige/timthumb.php?id=23%
27%20UNION%20ALL%20SELECT%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20N
ULL%2C%20NULL%2C%20NULL--%20 on 192.168.50.162
2013-04-27 16:40:53,905 (glastopf.glastopf) 192.168.50.155 requested GET /wp-content/themes/presstige/wp-content/themes/presstige/timthumb.php?id=23%
27%20UNION%20ALL%20SELECT%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20N
ULL%2C%20NULL%2C%20NULL%2C%20NULL--%20 on 192.168.50.162
2013-04-27 16:40:54,270 (glastopf.glastopf) 192.168.50.155 requested GET /wp-content/themes/presstige/wp-content/themes/presstige/timthumb.php?id=23%
27%20UNION%20ALL%20SELECT%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20N
ULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL--%20 on 192.168.50.162
2013-04-27 16:40:54,753 (glastopf.glastopf) 192.168.50.155 requested GET /wp-content/themes/presstige/wp-content/themes/presstige/timthumb.php?id=23%
27%20UNION%20ALL%20SELECT%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20N
ULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL--%20 on 192.168.50.162
2013-04-27 16:40:55,237 (glastopf.glastopf) 192.168.50.155 requested GET /wp-content/themes/presstige/wp-content/themes/presstige/timthumb.php?id=23%
27%20UNION%20ALL%20SELECT%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20N
ULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL--%20 on 192.168.50.162
2013-04-27 16:40:55,683 (glastopf.glastopf) 192.168.50.155 requested GET /wp-content/themes/presstige/wp-content/themes/presstige/timthumb.php?id=23%
27%20UNION%20ALL%20SELECT%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20N
ULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL%2C%20NULL--%20 on 192.168.50.162
2013-04-27 16:40:56,226 (glastopf.glastopf) 192.168.50.155 requested GET /wp-content/themes/presstige/wp-content/themes/presstige/timthumb.php?id=23%
27%20AND%20%2887%2B74%29=161%20AND%20%27ktVS%27=%27ktVS on 192.168.50.162
2013-04-27 16:40:56,505 (glastopf.glastopf) 192.168.50.155 requested GET /wp-content/themes/presstige/wp-content/themes/presstige/timthumb.php?id=23%
27%20AND%2074%3E%2867%2B87%29%20AND%20%27zPJH%27=%27zPJH on 192.168.50.162
```

Figure-20: Glastopf results on SQL injection

Test 3:

- Cross-Site Scripting (XSS)

Tools used & their results:

- W3AF with XSS injection plugin enabled.

Result to end user: - Dshield/Glastopf: With the current setups of the virtual environments user was unable to view any cross site scripting potential within the website due to the limited availability of the features in these packages.

Reason for selection of this tool for emulation from attackers prospective:

- XSS or cross site scripting is one of the most common web-application attack. The XSS attack focuses on the application layer and attacks the user of the web-application. XSS attack is generally used to steal cookies (session hijacking) or for delivering browser based exploits. The W3AF (web application attack and audit framework) can detect and exploit XSS vulnerabilities in any web application. While others tools can only detect XSS, W3AF can detect XSS and leverage it to deliver browser based exploits using the metasploit framework. Also, it has database of around 45,000 XSS attack patterns, including XSS filter bypass database which can bypass WAF (web application firewalls). This makes this tool the ideal choice for testing and exploiting XSS vulnerabilities (Riancho, 2012).

Reflection of results in web application honey pots:

a. Dshield: Dshield was able to capture the cross site scripting inputs that were used by the attacker.

Figure-21: Dshield analysis on Cross site scripting

b. Glastopf: Glastopf was also able to capture the events that shows the use of W3AF against the honey pot deployment and these events also capture the attack methods used.

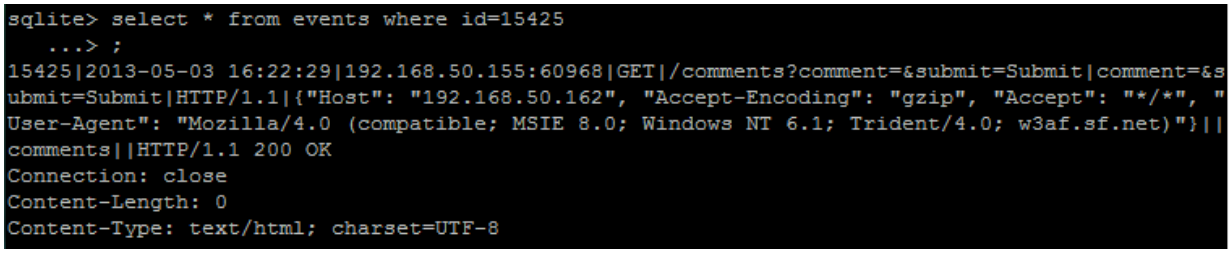


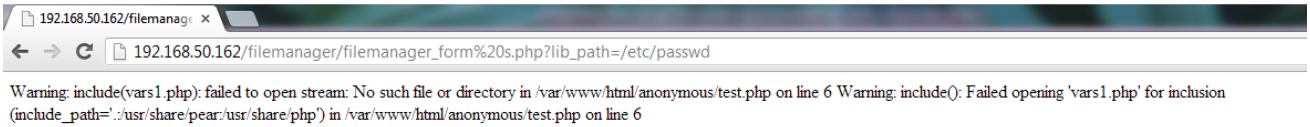
Figure-22: Glastopf analysis on Cross site scripting

Test 4:

- Remote/Local File Inclusion

Result to end user:

- The user was able to perform remote/local file inclusion(s) by inserting the URLs on the browser and was able to see the results.



Reason for selection of this tool for emulation from attackers prospective:

- Remote File Inclusion (RFI) vulnerability allows an attacker to include(inject) a remote file and read or execute the same by the web-application. This vulnerability occurs due to poor input validation. This is a very high risk vulnerability as it allows attacker to read sensitive files or execute arbitrary code on the web-server (Stuttard, 2011).

Local File Inclusion (LFI) vulnerability allows an attacker to include (inject) a local file and read or execute the same by the web-application. This vulnerability occurs due to poor input validation and allows an attacker to read sensitive files on the local system. If the attacker can also write a file on the system then he can combine this weakness with LFI to create a local web shell (backdoor) on the web-server. This is a very high risk vulnerability and can be avoided by ensuring proper filtering of user input (Stuttard, 2011)

Reflection of results in web application honey pots:

a. Dshield:

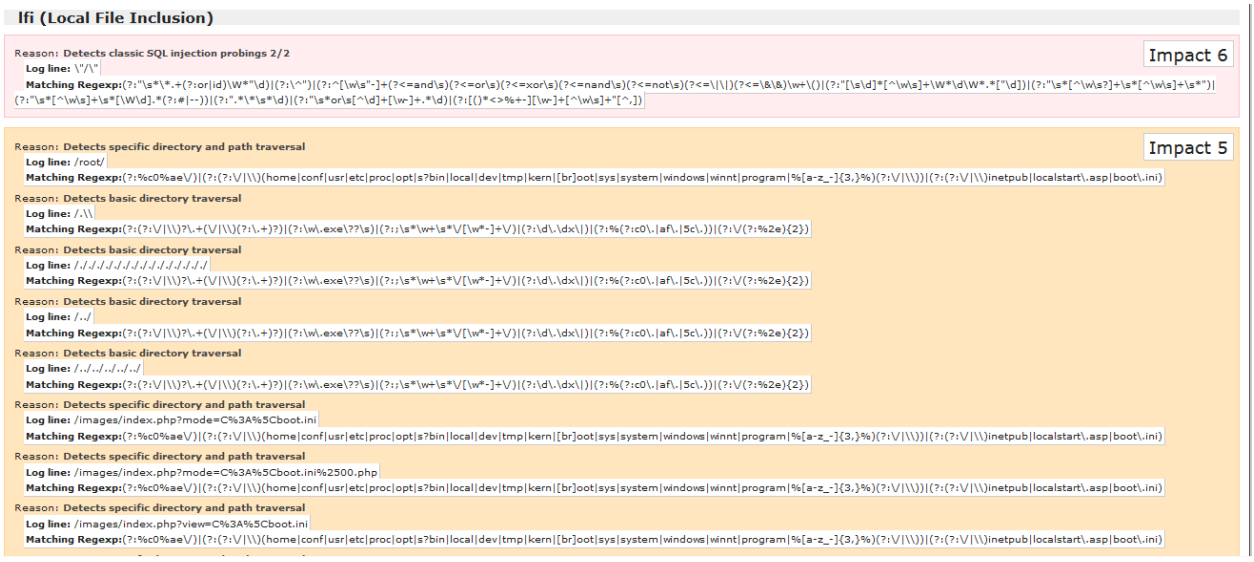


Figure-23: Dshield analysis on Local file inclusion

b. Glastopf:

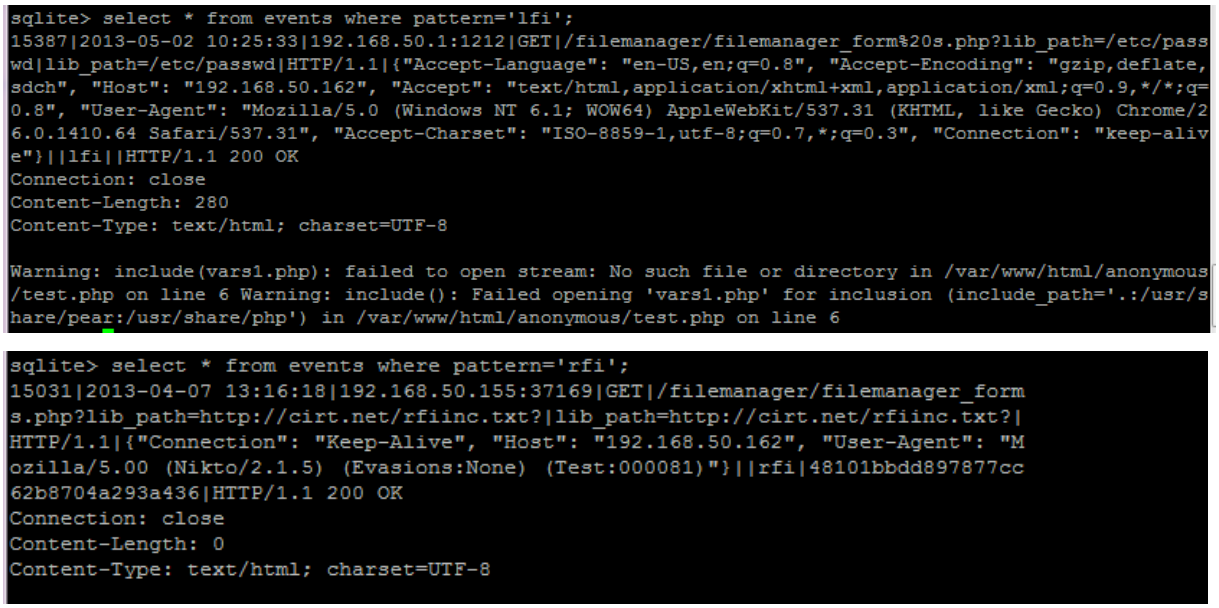


Figure-24: Glastopf analysis on Local file inclusion

Test 5:

- Command Injection

Result to end user: The user was able to do the command injection on the DShield/Glastopf interface and was not able to retrieve any valuable information and in case of DShield command injection was accepted but not processed, the attacker/user was redirected to the home page. This behaviour can fool automated scanners like nikto.

Information disclosed by the servers to end user about Glastopf and Dshield. Both of the web application honey pots display the website banners which could be helpful for the attacker to collect information about the server:

Glastopf:

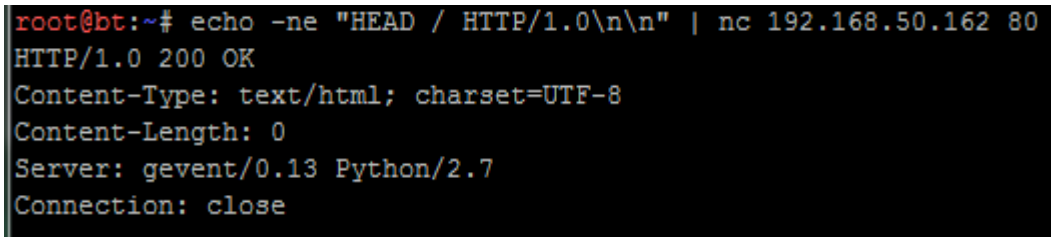


Figure-26: Glastopf analysis on Command injection

Dshield:

```
root@bt:~# echo -ne "HEAD / HTTP/1.0\n\n" | nc 192.168.50.171 80
HTTP/1.0 404 Not Found
Date: Sat, 04 May 2013 13:36:37 GMT
Server: Apache/2.2.15 (CentOS)
X-Powered-By: PHP/5.3.3
Last-Modified: Fri, 17 Oct 2008 17:03:05 GMT
Content-Length: 13695
Connection: close
Content-Type: text/html; charset=UTF-8
```

Figure-27: Dshield analysis on Command injection

Reason for selection of this tool for emulation from attackers prospective:

- Standard web browser was used to carry out the command injection attack. The command chosen were also targeted towards Linux platform because both the honey pots are running on top of Linux platform.

Reflection of results in web application honey pots:

a. Dshield: No results were traced by the DShield in case of command injections.

b. Glastopf: Results of the operations performed on the Glastopf honey pot were tracked in the events table of sqlite database which shows the trail of activities and attempts performed by the attacker.

```
sqlite> select * from events where id=15595;
15595|2013-05-03 17:00:30|192.168.50.1:7165|GET|/head.php?shell_exec('ls%20-al')|shell_exec('l
s%20-al')|HTTP/1.1|{"Accept-Language": "en-US,en;q=0.8", "Accept-Encoding": "gzip,deflate,sdch
", "Host": "192.168.50.162", "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,
*/*;q=0.8", "User-Agent": "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.31 (KHTML, like
 Gecko) Chrome/26.0.1410.64 Safari/537.31", "Accept-Charset": "ISO-8859-1,utf-8;q=0.7,*;q=0.3"
, "Connection": "keep-alive"}||unknown||HTTP/1.1 200 OK
Connection: close
Content-Length: 9486
Content-Type: text/html; charset=UTF-8
```

Figure-28: Glastopf analysis using Sqlite on Command injection

Test 6:

- Directory Traversal

Tools used & their results:

For testing directory traversal attack the following attack vector was used:

GET /admin/../../../../../../robots.txt HTTP/1.0

Using the directory traversal attack it was possible to read the robots.txt file which is on the document root of the web server. It shows us that both the web application honey pots can mimic directory traversal attack.

Result to end user:

a. Dshield:

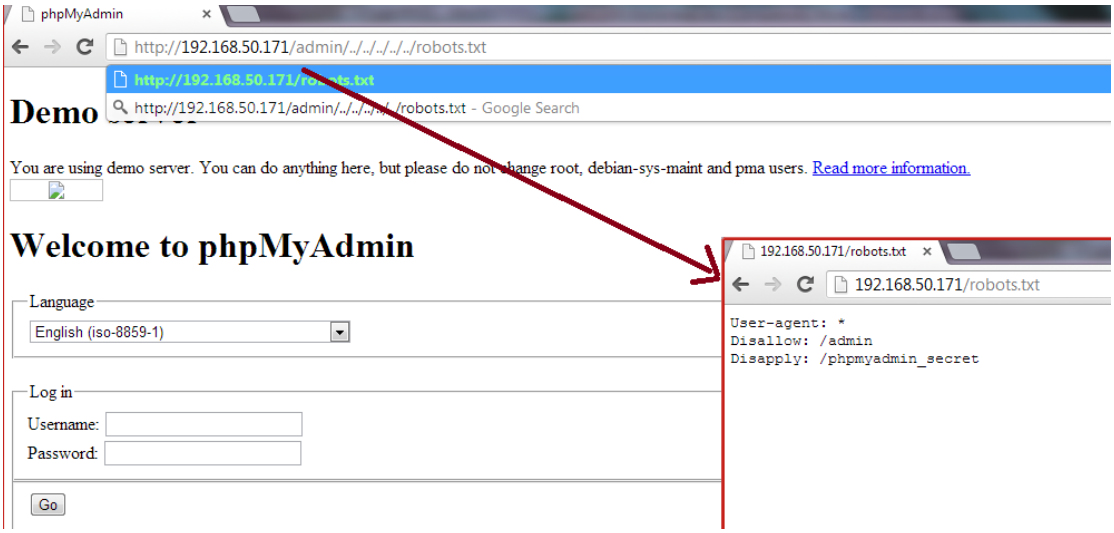


Figure-29: Dshield honey pot with directory traversal attack

b. Glastopf:

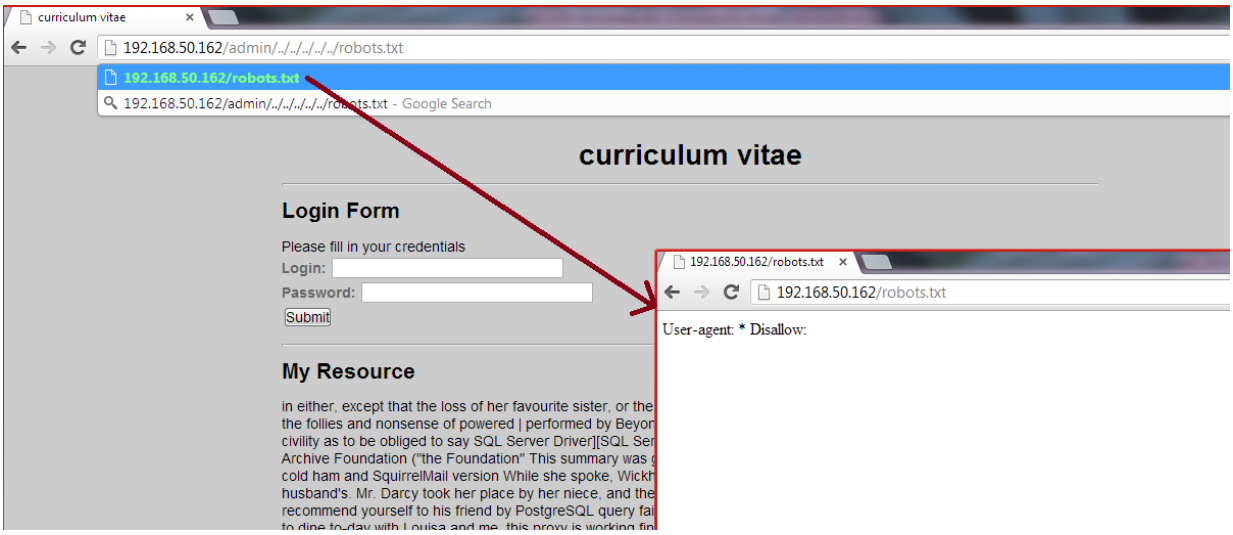


Figure-30: Glastopf honey pot with directory traversal attack

Reason for selection of this tool for emulation from attackers prospective:

Reflection of results in web application honey pots:

- Dshield and Glastopf both were able to capture the results from the directory traversal events fired by the attacker. Here is the results as shown by glastopf.

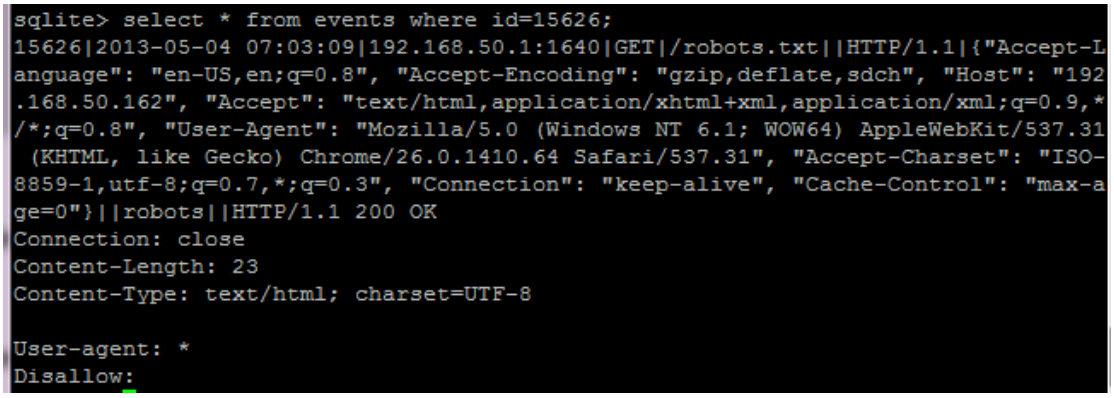


Figure-31: Glastopf honey pot accessing SQLite database for directory traversal attack

5.3 Summary

This chapter explained about the different experiments that were performed on the different honey pots i.e. Glastopf and Dshield. Next chapter would now reflect the analysis and evaluation of the experimental results.

CHAPTER 6: ANALYSIS AND EVALUATION OF EXPERIMENTAL RESULTS

In this chapter we would evaluate the experiments that were performed during the previous chapter on the different web application honey pots

Based on the tests performed on the honey pots following are the results that were extracted from the log files of the captures:

S.No	Attack Type	Results based on experiment performed on Dshield	Results based on experiment performed on Glastopf
1	SQL Injection	No	Yes
2	Cross-Site Scripting (XSS)	Yes	Yes
3	Command Injection	Yes [Limited]	Yes
4	Remote File Inclusion	Yes	Yes
5	Local File Inclusion	Yes	Yes
6	Sensitive Data Exposure (Robots.txt)	Yes	Yes
7	Directory Traversal	Yes	Yes

Table-5: Results from log files

Table-5 shows the output of the results that were extracted from the log files using log analyzer. This results elaborates that the attacker if simulate these type of attacks, then they would be captured by the honeypot. Moreover since Dshield and Glastopf are low interaction web application honey pots so they have a very limited set of web pages by default that is developed by the supporters of community to establish these honey pots for running and providing response to the attackers.

During the experiments on Dshield web application honey pot it was found that it has a very limited scope and is a very small application project that could be used as a honey pot. Deployment and configuration process of the DShield is very easy to do, however it is tedious to analyze the logging process of this web application honey pot as the default logs generated by this system is not user friendly. This is the reason a third part log analyzer has to be used at this stage for analysing the logs which makes it easy to read and understand the process/activities performed by the attacker.

Dshield appears as a normal website to the end user and performing tests on it is same as any other website. Dshield keeps its information masked and does not disclose it to the outside world. During the tests it was found that Dshield was able to capture the logs for the attacks. For every request Dshield will answer with HTTP 200 and redirect the attacker to the same web-page whereas glastopf generates random page for each request and for sql injection attacks produces mysql error message that is exactly similar to the error message produced by a real web-application which make it very difficult for an attacker to induce that the running system is not a real web-application but a honey pot. In terms of login capability, Dshield uses the logging capability of apache web server and stores the log as real text files.

During the experiments on Glastopf web application honey pot it was found that it has a very limited scope and is a very small application project that could be used. Glastopf can log the attack using a custom database. The database can be MySQL or sqlite database. Furthermore, the database has its own schema and can be analyzed using sqlite3 command line utility. It is hence also possible to write scripts that can parse glastopf logs in real time and insert firewall (iptables) rules on the fly to block an attacking IP. This effectively turns glastopf as a web application IDS (intrusion detection system).

Glastopf can log and detect command line injection attacks. However, it can't fully emulate command line injection vulnerability which makes its use quiet limited. Also, compared to high interaction web-application honeypot it offers limited interaction to the attacker when it comes to malwares. For example, it doesn't allow attacker to inject persistent XSS code which in-turn downloads malware into the client browser or loads a client side exploit. The ability to download malwares and exploits and save the same in

a separate directory is not present in glastopf. This feature should be included in the later version of glastopf. Since, glastopf runs PHP code(s) in sandbox, it is very secure. The best place to deploy glastopf in the network is alongside production web server(s) or in a separate DMZ protected by firewall(s) and IDPS (intrusion detection and prevention system). The logs generated by glastopf can help detect web application attacks against the web servers in the production network and help gather information (such as IP address, tools, techniques etc) related to those attacks. This information can be used to further secure the web application servers and can also detect co-ordinate attacks against web servers.

Summary

This chapter explained about the analysis and evaluation performed after the different experiments on the honey pots for Glastopf and Dshield. Next chapter would reflect the conclusions derived from the research.

CHAPTER 7: CONCLUSIONS AND FUTURE WORK

The best place to deploy glastopf in the network is alongside production web server(s) or in a separate DMZ protected by firewall(s) and IDPS (intrusion detection and prevention system). The logs generated by glastopf can help detect web application attacks against the web servers in the production network and help gather information (such as IP address, tools, techniques etc) related to those attacks. This information can be used to further secure the web application servers and can also detect co-ordinate attacks against web servers.

Dshield is a production level honey pot which is recommend to be installed in the DMZ along with existing web servers to detect and log scan and attacks against web application. Since, Dshield is only a PHP based application which offers limited interaction to attacker and can't emulate advanced vulnerabilities like SQL injection, it can't lure advanced attacker. According to the project home page: "***The gold of the Dshield web application honeypot is to collect quantitative data measuring the activity of automated or semi-automated probes against web applications. First of all, we will not just look for "attacks". We look for "probes". If they are malicious or not can only be determined in context.***"

Hence, it can be seen that Dshield honeypot is created to log information about common attackers and their IP addresses. This information will then be shared with other members of community who can then block these attackers in their perimeter firewall.

REFERENCES

1. Pinda, J. (2012). DShield Web Honeypot Project. Available at: https://sites.google.com/site/webhoneypotsite. (Last Accessed: 17 April 2013)

2. Rist, L. (2012). Glastopf Project. Available at: http://glastopf.org. (Last Accessed: 17 April 2013)

3. ENISA (European Network and Information Security Agency) (2012). CERT Exercises Toolset (Honeypots). Available at: http://www.enisa.europa.eu/activities/cert/support/exercise/files/Honeypotstoolset.pdf. (Last Accessed: 25 April 2013)

4. Lam, J. (2009). SANS Dshield Webhoneypot Project. Available at: https://www.owasp.org/images/3/30/SANS_Dshield_Webhoneypot-Jason_Lam.pdf . (Last Accessed: 25 April 2013)

5. Rist, L. (2010). Know your Tools: Glastopf, A dynamic, low-interaction web application honeypot. Available at: http://honeynet.org/files/KYT-Glastopf-Final_v1.pdf. (Last Accessed: 30 April 2013)

6. Riancho, A. (2012). W3AF USER GUIDE. Available at: http://w3af.sourceforge.net/documentation/user/w3afUsersGuide.pdf. (Last accessed: 27 April 2013).

7. Stuttard, D. (2011). The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws. 2 Edition. Wiley.

8. Yagi, T., Tanimoto, N., Hariu, T and Itoh, M. (2010). "Enhanced Attack Collection Scheme on High-Interaction Web Honeypots". IEEE Symposium on Computers and Communications (ISCC). Pages: 81-86.

9. Saat, S., Endut1, N.A. and Othman, A.H (2007). Capturing Web Application Threats Using virtual CMS Honeypot. Available at:

http://www.kaspersky.com/images/capturing_web_application_threats_(s.saat)-10-60018.pdf

10. Vetsch, S., Kobin, M. and Mauer, M. (2010). Glastopf. A dynamic, low-interaction web application honeypot.

11. Holz, T. (2007). Learning More About Attack Patterns with Honeypots. Available at: http://pi1.informatik.uni-mannheim.de/filepool/publications/learning-more-about-attack-patterns-with-honeypots-1.pdf

12. Provos, N. (2004). A Virtual Honeypot Framework. In Proceedings of 13th USENIX Security Symposium, pages 1–14, 2004.

13. The Honeynet Project. 2005. Know Your Enemy. Internet: http://www.honeynet. org, Accessed: 2005.

14. Joho. D. (2004). Active Honeypots, M.Sc. Thesis, Department of Information Technology, University of Zurich, Switzerland, http://www.ifi.unizh.ch/archive/ mastertheses/DA_Arbeiten_2004/Joho_Dieter.pdf.

15. Spitzner, L. (2002). Honeypots: Tracking Hackers. Pearson Education Inc.

16. Valli, C. "Honeypot technologies and their applicability as an internal countermeasure", *International Journal of Information and Computer Security, Inderscience Publishers, Geneva, Switzerland.* Pages: 430-436, 2007.

17. Mao, C. "Experiences in Security Testing for Web-based Applications". *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human, ACM.* Pages 326-330. 2009.

18. Huang, Y., Huang, S., Tsai, C. and Lin, T. "Web Application Security Assessment by Fault Injection and Behavior Monitoring", *Proceedings of the 12th international conference on World Wide Web, ACM.* Pages: 148-159. 2003.

19. William G.J., Halfond and Orso, A. "Preventing SQL Injection Attacks Using AMNESIA". *Proceedings of the 28th international conference on Software engineering, ACM.* Pages: 795-798. 2006.

20. Kieyzun, A., Guo, P.J., Jayaraman, K. and Ernst, M.D. "Automatic Creation of SQL Injection and Cross-Site Scripting Attacks". *Proceedings of the 2009 IEEE 31st International Conference on Software Engineering, IEEE Computer Society*. Pages: 199-209. 2009.

21. DShield Web Honeypot Project. Available at: https://sites.google.com/site/webhoneypotsite/

22. Honeypot through Web (Honeyd@WEB): The Emerging of Security Application Integration Nor Badrul Anuar, Omar Zakaria, and Chong Wei Yao University of Malaya, Kuala Lumpur MY.

23. Khattab, S., Melhem, R. Moss, D. and Znati, T. 2006. Honeypot Back-propagation for Mitigating Spoofing Distributed Denial-of-Service Attacks. Department of Information Science and Telecommunications,University of Pittsburgh.

24. Muter, M., Freiling, F., Holz, T. and Jeanna. 2007. A generic toolkit for converting web applications into high-interaction honeypots. Available at: http://people.clarkson.edu/~jnm/publications/

25. Yagi, T., Tanimoto, N., Hariu, T. and Itoh, M. 2011. Enhanced Attack Collection Scheme on High-Interaction Web Honeypots. NTT Information Sharing Platform Laboratories, NTT Corporation.

26. Farneth, J, Dhanju, A. and Blum, J.J. 2012. Analysis of Embedded Web Applications for Honeypot Emulation Programs.

27. Dagdee, N. and Thakar, U. 2008. Intrusion Attack Pattern Analysis and Signature Extraction for web Services Using Honeypots. First International Conference on Emerging Trends in Engineering and Technology.

28. Nassar, N. and Miller, G. 2012. Method for Two Dimensional Honeypot in a Web Application.

29. Rist, L. 2010. Glastopf: A dynamic, low-interaction web application honeypot. Available at : http://honeynet.org/files/KYT-Glastopf-Final_v1.pdf

30. Ragan, S. 2012. HoneyNet Project Releases SQL Injection Emulator.
Available at :http://www.securityweek.com/honeynet-project-releases-sql-injection-emulator.

31. Riancho, A. 2012. w3af User Guide. Available at:
http://w3af.sourceforge.net/documentation/user/w3afUsersGuide.pdf

32. Sullo, C. 2010.Nikto2 Web Scanner. Available at: http://www.cirt.net/nikto2

APPENDIX-A

POSTER

Study on Web application Honey pots

Kamaldeep Sehgal
1135553
MSc Computer Science
Supervisor: Dr. Ali Mansour

University of Bedfordshire

Introduction

Web application honey pots are a replica of the real application since it attracts the attackers to attack them. Identification of the attacks along with their patters is a classified information. However zero vulnerability attacks that are new born attacks are sometime exceptions. This set of information help in building the defending strategies against such attacks. However with the extensive range of attacks, attackers intrusion, variety of attacks and vulnerabilities combine together to make it complex to address all the problems. Moreover new vulnerabilities make these things more difficult [1].
Honey pots are classified based on the their level of interaction with the outside world like,
- High level interaction,
- Medium level interaction, and
- Low level interaction honey pots.

Low level interaction honey pots do only limited interaction with the attackers and are one of the easiest honey pots to deploy and configure. These honey pots are designed to interact with the attackers and log the information which is later on sent to the administrators for study and analysis (Zhou et.al, 2012).

Problem Statement

These days web application are being extensively used over the internet which also indirectly gives the invitation to the attackers to hack the websites and retrieve confidential information and data. The vulnerabilities of the web application let the attackers intrude to the servers or can even get inside the network environments. Attackers the can exploit the web sites with different attacks and can cause heavy damage to the owners of the website or the corporate holders. In this situation the owners of the website have to stay updated all the times so that they can fight against the vulnerabilities of the websites. However it also gets very difficult to protect the websites against zero-day exploits which are newly developed and invented.

Aim and Objectives

This project will focus on the critical study and evaluation of the web application honey pots which will show the different techniques and solutions used. These days web-application and client-side attacks are increasing that demands for a solution to detect them so that they can be prevented effectively [2]. This project will study about most popular web application honey pots like Glastopf [3] and Dshield web honey pot [4] and present the critical findings on them.

This projects aims to study about the different aspects of web application honey pots available and what is the current trend and solutions being implemented to mitigate the security risks around web applications.

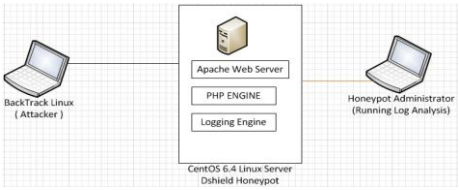
Objectives:

- Researching the subject.
- Study about the various Honey pot technologies available.
- Research about the web application honey pots.
- Research about the various available solution of web application honey pots and their applications.
- Perform deployments of two web application honey pots Dshield and Glastopf.
- Evaluate and conclude about the Honey pots.

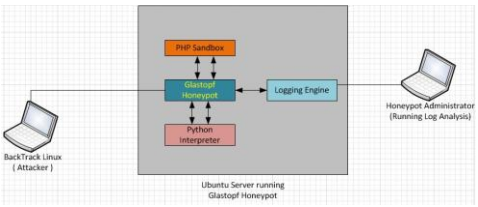
Design & Implementation

The system design outlines the design of both the honeypots. DShield is one of the low interaction web application honey pot which is deployed and configured on the CentOS 6.4 Linux Server. Honey pot machine would contain the Apache web server running with PHP scripting engine hooked with the Logging engine.

References:
[1] Costa, J.P., Freitas, E.P., David, B.M., Serrano, A.M., Amaral, D., and Júnior, R.T. 2012. Improved Blind Automatic Malicious Activity Detection in Honeypot Data. Available at: http://www.icofcs.org/2012/ICoFCS2012_08.pdf.
[2] A. Ghourabi, T. Abbes, A. Bouhoula , "Design and implementation of Web service honeypot," 19th International Conference on Software, Telecommunications and Computer Networks (SoftCOM), Pages: 1-5, 15-17 September 2011.
[3] Glastopf Web Application Honeypot http://glastopf.org/glastopf.php
[4] DShield.org Web Application Honeypot http://code.google.com/p/webhoneypot/



This glastopf honey pot is installed on a Linux server running Ubuntu 12.04. Glastopf honeypot runs on top of python scripting engine and works closely with a PHP Sandbox which is also installed on the Linux server. Glastopf has extensive logging capabilities and can log all aspects of interaction with the attacker. Malicious, PHP code that will be injected by the attacker will be run inside the PHP sandbox by glastopf and the final result of execution is shown to the attacker.



IMPLEMENTATION

The honey pot systems will then be probed and attacked using open source web-security tools that comes pre-installed with BackTrack Linux or can be downloaded freely from the internet. It would be expected from the both the honeypots to respond back to the attacking machine (user) and all the attack data would then be captured by the logging system.

TESTING AND EVALUATION

Penetration tests were performed on the honey pot deployments to simulate the attacks and verify if the information about the attacks done by the attackers are being logged by the honey pots are not. Dshield and Glastopf were able to trace many events as tried by the attacker.

S.No	Attack Type	Results based on experiment performed on Dshield	Results based on experiment performed on Glastopf
1	SQL Injection	No	Yes
2	Cross-Site Scripting (XSS)	Yes	Yes
3	Command Injection	Yes [Limited]	Yes
4	Remote File Inclusion	Yes	Yes
5	Local File Inclusion	Yes	Yes
6	Sensitive Data Exposure (Robots.txt)	Yes	Yes
7	Directory Traversal	Yes	Yes

CONCLUSION AND FUTURE WORK

The best place to deploy glastopf in the network is alongside production web server(s) or in a separate DMZ protected by firewall(s) and IDPS (intrusion detection and prevention system). The logs generated by glastopf can help detect web application attacks against the web servers in the production network and help gather information (such as IP address, tools, techniques etc) related to those attacks.

Dshield is a production level honey pot which is recommend to be installed in the DMZ along with existing web servers to detect and log scan and attacks against web application. Since, Dshield is only a PHP based application which offers limited interaction to attacker and can't emulate advanced vulnerabilities like SQL injection, it can't lure advanced attacker.