Tracing Cyber Threats

With Honey Systems

By

ROHIT SEHGAL

COMPUTER SCIENCE & ENGINEERING DEPARTMENT
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

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

THESIS SUPERVISOR
Dr. Sandeep Shukla

MAY 2017

THESIS CERTIFICATE

This is to certify that the thesis titled "Tracing Cyber Threats with Honey Systems", submitted by Rohit Sehgal (15111038), 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:

ACKNOWLEDGEMENT

I would like to thank Dr Sandeep Shukla, for his motivation, inspiration, support and knowledge that helped me to achieve the target of my M.Tech Thesis.

I would also like to thank Mr Anil Yadav Samsung Research And Development Institute Delhi for offering an opportunity to be a Research Intern that helped us to deliver the model like Clientpot.

This acknowledge will be incomplete without the mention of two other person who worked in the field of Honeypot systems, my lab mates. Nishit Majithia who helped me like an elder brother and helped me to get into positive and fruitful direction. Without him, it would have been difficult for me to attain my target on time. Also great thanks to Krishnaprasad P, for listening to my doubts, problems and dedicating his time to solve them. My lab mates at RM506 and RM505 have been a continous support for my thesis. It is their motivation and inspiration that enabled me to land at a place where I am now. Special thanks to Nazia mam for reviewing this thesis report.

I also respect my friend Rakshit Sharma for making my stay at IIT Kanpur more pleasant. Hearty thanks to Lieutenant commander Amit Singh and his wife Mrs Anubha Singh for offering love like a family. Their presence was the most memorable thing that happened to me at this place.

My family is the one whose vision I am accomplishing. I would like to dedicate my thesis to them. My mother, father , younger and elder brother, I love you all.

Preface

The research work presented in this thesis report is part of a close collaboration, led by **Prof. Sandeep Shukla** at the **Indian Institute of Technology Kanpur**, with Nishit Majithia. This thesis presents three major research systems viz Clientpot, Honeytoken based solution for Malware like Gooligan and several generic Honeypots for various services. The first two systems or solutions presented in chapter **2 and 3** are the part of actual collaborative work, whereas the work presented in the design of generic Honeypots for various services is my independent work. The literature review has been made independently and has been outlined in the chapter **1**.

The Clientpot model was developed by Nishit Majithia and I as research intern at the **Samsung Research & Development Institute, 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 the monitor applications in Tizen based TV systems.

Analysed the logs and binaries captured by the Honeypots be it windows or Linux were done together by Nishit and I as it requires much time and analysis skills.

An independent study has been made into exploit to imitate the behaviour of Gooligan malware using CVE 2016-5195 vulnerability for Android. The conceptuality of about how Android sandbox can be broken, is part of a collaborative work by Nishit Majithia and I.

ABSTRACT

Name - **Rohit Sehgal**
Roll Number - **15111038**
Degree for which submitted - **Masters of Technology**
Department - **Computer Science & Engineering**
Thesis title - **Tracing Cyber Threats with Honeysystems**
Thesis submitted To - **Dr Sandeep Shukla**

Honeypots, Honeynets, and Honeysystems are entrapment devices to capture the continuous barrage of cyber-attacks to the IT infrastructure of companies, and institutions. Threat intelligence is important component of defense against cyber-attacks. Threat intelligence includes the malware, scripts and other artifacts that the attackers use for launching cyber attacks as well as the mode of attack, the attacker capabilities, and behavioral aspects. Once a honey-system is deployed, the attackers leave their signature and various tools into the honey-system and their behavior is logged. This data then can be further analyzed to generate threat intelligence which can then be used to decide the defensive techniques. This thesis presents simple and easily deployable Honey-pot systems ranging from client side Honey-pots, Honey-tokens to various server side Honey-pots. Our deployments have been able capture various tools used by the attackers to target weakly defended machines. Besides understanding of tools used by the attackers, we can glean the knowledge about zero-day attack vectors. If poorly designed, honey-systems might compromise the systems being defended. In view of that, we have designed, implemented and deployed honey-systems that offer a good level of security. From the post-mortem data analysis, we have been able to obtain a good amount of intelligence on what kind of attacks and attackers are targeting our systems at IIT Kanpur, and also outside. The major contributions in this thesis includes:

a) An approach to create a malware like Gooligan and Honey-token based post exploitation techniques to identify breached devices. **b)** Lightweight models of Honey-pots with live monitoring web interfaces. **c)** A Honeyclient for identifying malicious web servers that captures drive-by-download based attacks targeting Linux based machines.

Table of Contents

List of Tables

Table **Page**

List of Figures

Figure **Page**

CHAPTER 1

INTRODUCTION

Since the encroachment of the Internet into the day-to-day lives of common people, cyberattacks have become inevitable. This inevitability of getting attacked, has evolved research in the field of system security. Researchers have proposed ideas and models to prevent user identities from being robbed. There has been reduction in attacks and identity thefts with the development of these tools and tactics.

Moreover with the idea of *Cloudification*, in which enterprises keep their valuable data and resources in a common place i.e *Cloud* infrastructure, further incentive for attackers to do some mischief has been enhanced. The mischief is to gain some benefits out of them, either in terms of money or fun but most of the times the intent is ambiguous. Evidence from an attack DDoS attack on DYN Domain Name Server on October 21, 2016 [1] involved a targeted attack which created a bot network by identifying weak IoT devices with default username and password, and asking all those bots to attacks the server at once.

From the history of attacks that occurred in past two three decades, it is easy to cluster them among several types such as botnets for DDoS, malware, drive-by-downloads, ransomware etc. Research community has proposed different tactics to capture them in order to identify the intent of the attacks. And this was the major reason behind the evolution of Honeypots. Even after the existence of such research systems it still requires good amount of domain knowledge to identify attacks and their intents. Similar to other forms of crime, in cyber crime, very little or no knowledge is available about the attacker, the tools they use, their hacking expertise and their motive for attack.

This thesis proposes solutions to some of the problems mentioned above. Also it proposes a number of different systems that may perform better for some of the mentioned attacks. Different aspects of Honey system, from their implementation to their applications have been covered and using the technology of honeypots a model to capture traces of attackers is presented.

1.1 Honeysystems

The word Honeysystem is a collective term that is used to describe different varieties of Honeypots, which include client-side Honeypots, server-side Honeypots and Honeytokens. Each of these Honeypots offer different functionalities. Apart from offering different applications, their main objective is the same i.e to deflect and deceive the attacker or to

gain as much information about the attack vector as possible. There has been previous instances of deployment of Honeypots that led to capture of scripts used by attacker community to execute attacks like DYN DDoS attack [2].

Lance Sptizner have proposed the definition of Honeypots and according to him - *"The Honeypot is a security resource whose value lies in being probed, attacked, or compromised "* [3]. The more ease it offers in getting exploited, the more useful the Honeypot is. Offensive strategy they offer are different from traditional security mechanisms. They are built in a way such that they are isolated from the network and stores a copy of entire production server in the DMZ [1] without any real sensitive data. Such systems offer path of very least or no resistance to the attacker. After successful deployment of Honeypot and on analysing the main motive, action, behaviour, exploits and their scripts, the system administrator may take steps necessary for the protection of their main production systems i.e what kind of patch need to be applied to their code-base in order to protect them from getting exploited in future.

The Honeysystems as explained above can cover a range of systems, *server-side*, *client-side* etc. The Server-side ones emulate the behaviour of a server by exposing the services of a server exploitation by an attacker. The client-side ones, instead of deploying the services offered by server, emulate an actual client side system i.e an operating system running several services interacting with the malicious web server. On the basis of the interaction offered by the Honeypots, they are classified into two major categories, High & Low interaction. High interaction ones try to fetch maximum amount of the information of the attacker, analysing their every single move. On the other hand low interaction systems capture their attack vectors for later analysis.

1.1.1 Types of Honeypots

The level of interaction offered by a Honeypot is a metric to compare different types of Honeypot. The following are the traditional categories based on interaction capacity :

1. **Low Interaction Honeypots** - They are easy to deploy, configure and maintain because of their simple design and implements very basic functionality [3]. Such type of Honeypots simulates the front end of various services. A very simple example of it can be a login screen service that captures the user name and password combination but the actual login system to which attacker tried logging into is absent. Such systems offer very minimal risk to the system administrator as they only simulate services and hence there is very little chances of system getting abused e.g these systems cannot be used to attack the other systems. In turn these type of Honeypots are not very research friendly and they extract less information about the attacks. The sort of information that could be extracted from such system is very much limited.

2. **High Interaction Honeypots** - They extract much larger amount of information about attackers but they are extremely time consuming to build and maintain

[1] De-Militarised Zone

and they come with the highest level of risk [3]. Such Honeypots allow us to discover the tools, attack patterns used by the attacker and also allows to gain maximum information. If deployed properly, they may prove to be a good asset to the organization. This is the reason that they are also known as research systems. In order to have those systems deployed, a good care needs to be taken as they can be used by an attacker to attack other systems. This thesis presents detailed model of these high interaction Honeypots. It will also discuss some instances when Honeypots were compromised and used to abuse other machines.

Other than the above mentioned categories, there exists other kinds of Honeypots. Categorised on the basis of the services they offer or the basis of service that they emulate or simulate, they can be further classified into various classes :

1. **Server Side** - Honeypots offering the interface to services that are typically offered by a server. as example a sever emulating or running `telnetd` service open for attacks. They may expose vulnerable services for exploitation by attackers. The information extracted from such attacks allows to tighten the security control for an organization's security. A major part of this thesis is dedicated to such models, their implementation, design and attacks they capture.

2. **Client Side** - A Client side Honeypot allows to analyse the attacks that are targeted by a malicious web server on a client that connect to those servers. Client Honeypots crawl the network, and classify the server on the basis of their malicious activities. The major difference between a client side Honeypot and tradition Honeypot are [4]:

   - Client side: simulates client-side software and without exposing server based services.
   - Active: They cannot attract attacks rather they actively interact with remote servers to identify remote malicious or compromised servers.
   - Identify: Where all accesses to the traditional Honeypots are malicious, the client-side Honeypot must discern which server is malicious and which is benign.

3. **Honeytokens** - They are not computer systems or services but rather they are tokens whose values lie in their abuse. The major concern with them was to identify the password cracking, proposed by Ari Juels & Ronal L. Rivest [5]. In their proposed model they try to identify the cases of password cracking. This thesis proposes an idea using Honeytokens to tackle a problem that appeared with the spread of a malware like *Gooligan*.

1.2 Honeypots vs Traditional Security Control

1.2.1 Traditional Security Control

The focus of Computer security researchers prior to Honeypot was on passive defence strategies, primarily with the use of firewall or Intrusion Detection System(IDS). An IDS works by comparing the behaviour of standard users to a user with anomalous activity. On the basis on of the rules written in the IDS, it identifies the anomaly. A Firewall on the other hand, specifies allow/deny rules to system access, some firewall methods provides flexibility to system administrator to write the logs to the system. Both of these method execute on the basis of predefined rules, and these rules are identified on the basis of the history of attacks faced by the service. These mechanism are passive in nature in the sense that they are not designed to identify unknown attacks. A rule based IDS fails in scenario when it identify the attack or pattern with unknown signature.

1.2.2 Honeypot Model

In contrast to the traditional techniques, Honeypots are active in nature. Active here means that they are able to change the state with type of attack faced. They are able to deal with the new types of attacks, also known as zero day attacks or exploits. They are deployed so that the attacker who discovers it may choose to compromise it. After the attack, system administrator can investigate the attack in detail, identify the tools and techniques used by them. The analysis of the attack can provide valuable information to the organization e.g about some vulnerability which needs to patched, or backdoor that needs to be fixed. Honeypots also offer an advantage to keep intruder away from the actual production system. Creating and deploying such Honeypot system is fix-build incremental process. With time system administrator may keep on enhancing the security of the Honeypot with the analysis of attacks.

1.3 History and Overview of existing Honeypots

In the adversarial game of attacking community and researchers, every time when a defensive model is prepared by the researcher, attacking community tries to find a counter to that particular defensive move. This game has led to the development of various Honeypot, with certainly different applications. This section present a brief review of the design aspects of existing Honeypots solutions and how they evolved.

1. **Simple Logger** : The simple logging facility available to us by *syslog*, *sebek* and *auditd*, are examples of low interaction Honeypots. This model requires a system that one need to put on public network to capture more attack vector and in addition by deploying it on an easily compromisable machine. All the interaction are logged by the system logging facility. The main disadvantage of the design is if compromised badly, then the attacker may wipe all the logs from the system. So an alternative was to log in a remote location.

Sebek is a data capture tool, which allowed us to recreate the attacks on a Honeypot. But soon attacker developed a tool called as *Kebes*, an anti - Sebek toolkit for the detection and disabling of Honeypot [6].

2. **BOF** & **Specter** : *Backofficer Friendly* and *Specter* were one of the earliest Honeypot solution for Windows easily deployable and maintainable. They emulated various services with *Specter* emulating more service than *BOF*. They offer low level of interaction with the service they emulates. The model BOF was proposed to identify the trojan named *Back Orifice* for the Windows 98 platform. The main motive of the BOF was to open a dummy service on the port 1337, which was the port used by *Back Orifice* to create backdoor for remote access [3].

3. **Honeyd** : Honeyd is a low interactive open source Honeypot for Linux based platform whose development started in the year 2002. This lightweight model was capable of simulating different Honeypots on one system. The core idea of this model was to create system of dummy machine with the unused IPs in the LAN. When attacker tries to connect to IP address which is not allocated to any system, honeyd communicates on its behalf by sending the spoofed ARP packets. This system was capable of simulating behaviour of different OS even for windows. This model is suited best to analyse the attacks in case of Local network, but does not seems effective for the system that allocate static IP in public network [3].

4. **Capture-HPC** High interaction client side Honeypot solution for emulating Windows based client. A client Honeypot or Honeyclient is a security technology that allows one to find malicious servers in a network. Capture identifies malicious servers by interacting with servers using a dedicated virtual machine and observing its system state changes. If any suspicious system state changes are detected on interaction of Capture with server, Capture classify it as 'malicious' [7].

All these systems were just fundamental Honeypot solutions that have been proposed for analysing attacks. They do not offer any platform for research. They were not highly interactive and could not capture detailed evidences of attacks. Although the design and configuration of the these Honeypots was very much specific to target. At times it is required to tweak these simple Honeypots, to get some advancement out of them.

1.4 Overview of This Thesis Report

Rest of the chapters in this thesis report have been organised in the following manner. Chapter 2 covers the idea and design behind the **Clientpot** - a medium interactive client honeypot model developed at Samsung Research & Development Institute, Delhi during our internship period. Chapter 3 presents post exploitation method for the identification of devices breached by **Gooligan** like malware using **Honeytokens**. It also discusses an approach of this malware on the top of one of Linux's privilege escalation bug. Chapter 4 presents the design and implementation of the model of different Honeypot solution

for different Linux based services. The main idea presented in this chapter that is of a **HoneyFarm**. This is a model to capture ssh brute-force attacks. This chapter also presents two light weight models to capture attacks on FTP and HTTP service, thereby creating a Honeynet. Also a web interface built on the top of Elasticsearch, Logstash & Kibana has been implemented for the live analysis of attack on different protocols. Chapter 5 presents in detail analysis of logs captured by Honeynet and the knowledge extracted from them. Some analysis is also compared with the darknet traffic that IIT Kanpur network faced [2]. The thesis report then concludes with chapter 6 covering conclusion and future work.

[2]The darknet traffic have been captured by two students Devashish Kumar Yadav and Nikhil Vanjani as their UGP project under Dr Sandeep Shukla, IITK

CHAPTER 2

CLIENTPOT - A CLIENT HONEYPOT

Client side attacks are getting attention with the intent of the attacker to defraud victims. Attackers aim to put a malware or a malicious script on victims machine, to steal identity specific sensitive informations that could be either victim's username/password, credit card number etc. In the if victim's machine is behind a NAT, then attackers may also be interested in getting the information about the Local network of the victim. Every single point of carelessness of a victim is an invitation to a number of attackers. Most of the time it has been observed that, victim is unaware of such things that happen without his conscious. Such attacks are invited by the user itself, who is interacting with a malicious web service without any security. Such malicious web servers always wait for victims to come and browse them, so that they may inject malware or unwanted scripts in the victim's machine. These Scripts may perform port scanning of the network, infect victim's machine with a worm, trojan etc.

The researcher's community has devised a model called *Client side Honeypot or Honeyclient* to identify the client side threats involved in the web browsing, on interacting with any malicious web server. There has been a series of different Honeyclients, for tackling the issues related with different client side threats. E.g *Capture-HPC* allows to find malicious servers on a network. It does so by driving a vulnerable browser to interact with a list of potentially malicious servers [7]. This model turned out to be good, but can only simulate victim client as windows. Although there exists a similar model for Linux environment as well i.e. *Capture-HPC Linux*, but that uses the concept of Virtual machine management and state change analysis, therefore it is not well suited for light weight environments such as smart devices, TVs, mobiles etc.

Motivated from the same model, the **Clientpot** was developed in Samusung Research Institute Delhi, as a research intern team project with Nishit Majithia. The model is light weight in a way, that it uses light APIs to mimic Linux based browser victim and system. Samsung Research Institute has also proposed a technique to integrate it with their Tizen based systems.

2.1 Design

Studies have examined severe attacks to intrude into client systems by poisoning web servers [4]. There exists various static and dynamic analysis methods to identify these client side attacks. But they do not behave similar to how an actual client would have be-

haved. Client side attacks are usually categorized as *Web-Browsing based* like XSS(Cross Site Scripting), CSRF(Cross Site Request Forgery) etc or *Drive-by-Download based* in which malicious web servers make clients to download malicious binaries into their systems. Clientpot system is able to recognize the attacks initiated by web servers that may inject malware into user's system.

Clientpot design is based on client server model. Client issues web URL crawling request to the server and server in turn crawls the domain specified by URL. During crawling, if the browser triggers some attacks, like *drive-by-downloads* then server saves the URL, creates on the fly Linux Container, issues the URL to the container, and container runs that particular downloaded file in sandboxed environment by monitoring the state changes in the system. On the basis of monitored system state changes, user can analyze the logs and mark the server as benign or malicious. After each interaction, the changes in the file system structure and process tree are recorded. Client pot does not use any kind of predefined signature to classify the web server, so is open to zero day attacks.

2.1.1 Framework

Linux system was used to host the Virtual Machine Manager (VMM). The VMM model used was Virtual Box, with Ubuntu 14.04 LTS. VMM provides the hosting for Linux virtual operating systems, that are created on the go on each user request. Linux OS Virtualization technique was used to create on the fly sandboxed environment. For the same, LXC was used which is free open-source and lightweight software Virtualization technology, rather Virtual Machine based Hardware one. Figure 2.1 describes the model of the entire system view of Clientpot.

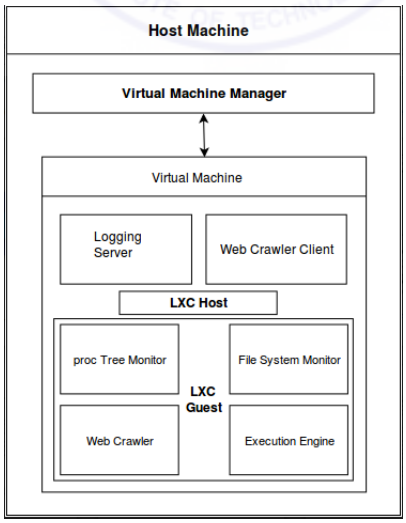

Figure 2.1: Clientpot system

Clientpot consists of various services viz *web-crawler client, logger, File monitor, process monitor etc*. All these services together bring the system in action. The system is made to run all these process in user space. As Compared to *Capture-HPC*, these core services were run in the kernel space. Kernel space processes are quite heavy to recreate and offer a good level of risk when dealing with malwares and other malicious scripts.

2.1.2 Service Interactions

The client initiates a series of invocations when user initiates a request for a particular website or web server. A web crawler client invokes a request to the web crawler server. This service then performs the crawling and execution engine executes the attack vector received by the web server. Two independent services viz *process monitor daemon* & *file system monitor daemon* examine the state changes of the system and report them to the logging server. Figure 2.2 enumerates the complete sequence of the interaction.

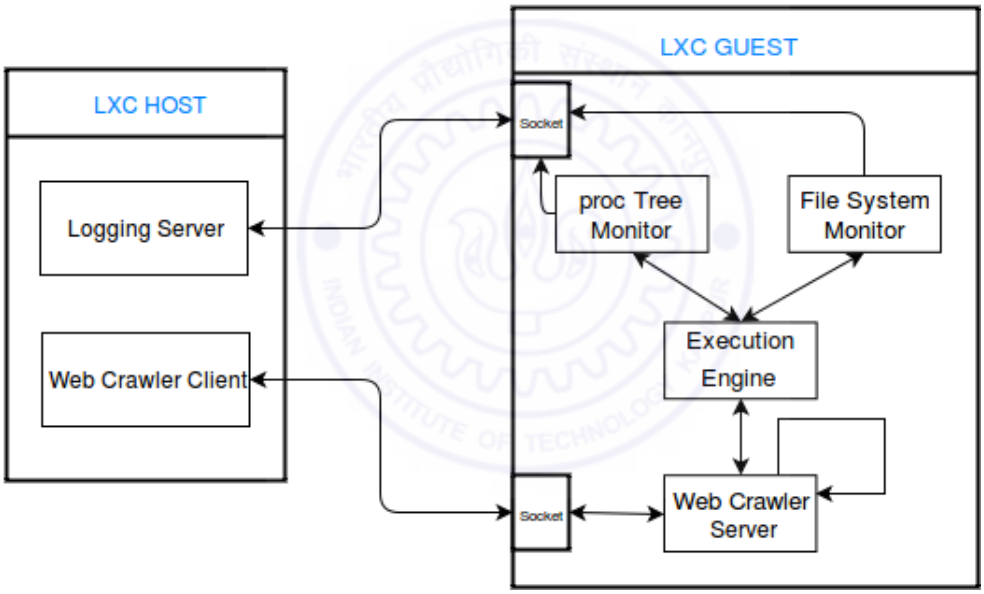

Figure 2.2: Clientpot process invocation overview

2.1.3 Services

Following is the set of processes that run across multiple virtual systems to provide desired functionality of a Clientpot :

1. **Web Crawler Client**: Process that invokes entire system. This Process takes an input from the user which is a URL to crawl. Then sends address to Web Crawler server, which crawls the domain for various href links, followed by execution engine which handles each link.

2. **Web Crawler Server**: Receives request from the web_crawler_client for the domain to explore. If during the domain exploration it finds any suspicious link within the same origin, then the link is handed over to the execution engine and is checked for further traces of the downloaded file, if any.

3. **Execution Engine**: This daemon accepts all link requests from the web_crawler_server, filters them on the basis of extension and runs them. E.g if the link contains .html extension then it creates a request to the **lynx** browser to open it. If the link offers a pdf file, the file is downloaded and analyzed using **pdftk** tool, and if the binary is executable, it runs the binary in the sandboxed environment.

4. **File System Monitor**: This daemon listens for the file structure changes to the specific directories, i.e. unauthorized file system changes. If upon execution of link by **Execution_engine** results in file system changes, those changes are reported back to Logging server. This daemon is written in C language and uses the Linux open source **Inotify** API to perform snooping of the file system. This module is capable of reporting various file system state changes like file creation, deletion, modification, access, directory creation etc.

5. **Process Tree Monitor**: This daemon performs action similar to that of File System Monitor service, but reports the state changes in the OS processes' tree. This is achieved by intercepting various system calls executed while running a file by execution engine. It deploys very simple user space, system call interception to monitor the changes. The information about how the system calls are intercepted using this technique is depicted in the Figure 2.3

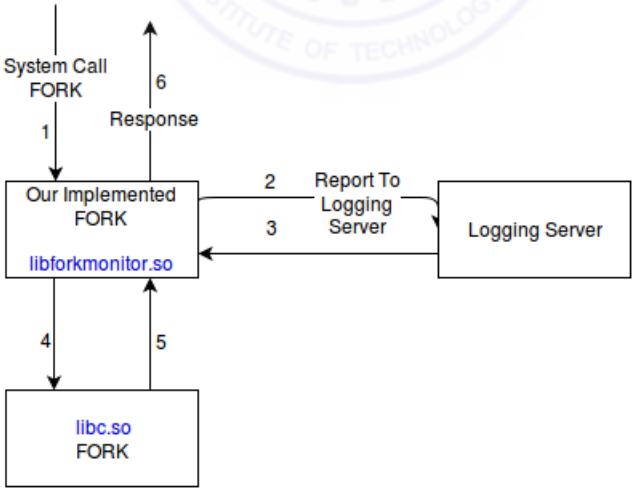

Figure 2.3: Intercepting fork call in Clientpot

Figure 2.4 suffice the interaction sequence of different process. The entire system is run in Virtual Machine, running Ubuntu 14.04 LTS. Within the virtual machine, there

hosts a client server LXC model to run each request in OS level isolation. The virtual machine is also host of LXC client. The Web crawling request is issued to the LXC guest machine. This machine is created on the fly per LXC host request and wipes away once the request is served. This makes each request to be served in isolation. Once the request is issued, the web crawler server running in LXC guest, crawls for that particular domain. While crawling, if any suspicious link is identified, then it is downloaded and executed separately in Execution engine. This Execution engine servers each downloaded file by matching the pre-defined rules corresponding to their extension. E.g a pdf file downloaded is made to run and analysed in tool called *pdftk*, a binary is simply run in a sandboxed environment and for compressed files, the files are first un-compressed and each file within them is executed in same fashion recursively.

When the binaries are run, they are run in sandboxed environment. The sandboxed environment is prepared by augmenting user space to intercept all the system calls and monitoring all the file system access. This sandboxed system is socketed with LXC host system. Host runs a logger daemon and this logger logs all the activities and changes made reported by LXC guest.

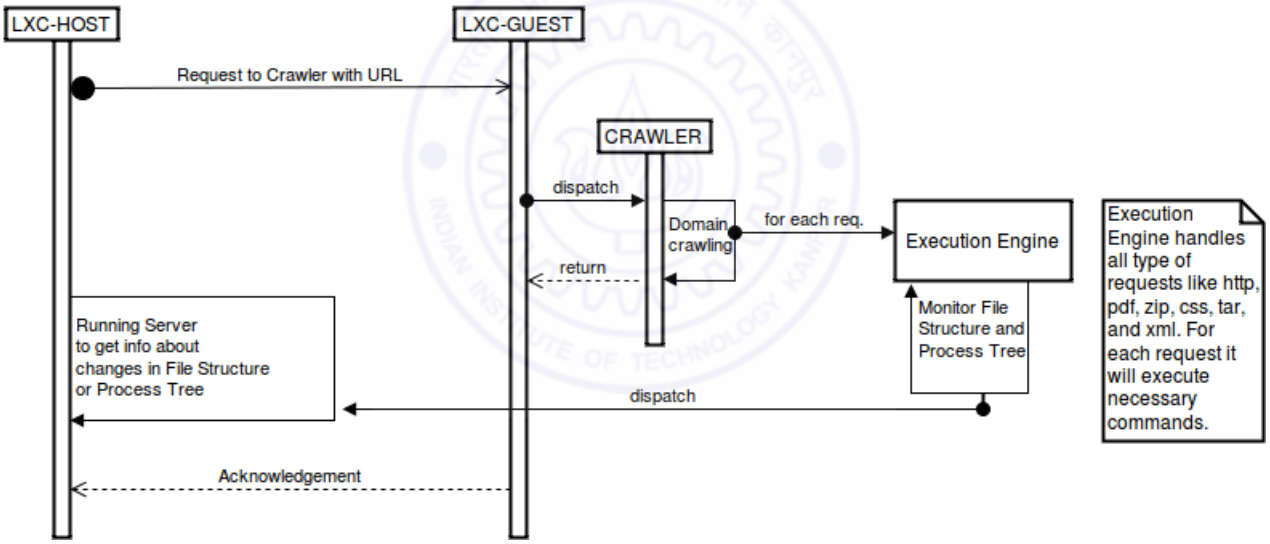

Figure 2.4: Sequence diagram of processes of Clientpot

2.2 Implementation

The major services in Clientpot are *File monitor* & *Proc Monitor*. These services are built using the Linux open source *inotify* & *dlfcn* API. They allow us to perform monitoring using user space processes. Though the file monitor was loaded at boot time, runs with root user id. This behaviour is acceptable as it allows to snoop root directory as well.

2.2.1 File Monitor Service

This Service written in C++, snoops for the specific directories by default. During the boot time the following directories are snooped for the following events: `/tmp/,access,modify, create /,create, modify, delete`.

The syntax is specified in the format `<directory_path>, {access | modify | create | delete }+`. E.g `/root, access, modify, delete` will allow file monitor to snoop in the directory `root`, and in the same directory if any file or directory is created, modified, accessed, or deleted, then file monitor reports the entire event description to the logger service running in LXC host system. For obvious reasons, this service can snoop to those directories, to whom the owner of service running have access to view. This is the major reason that, service is initiated by the root process during boot up phase.

Inotify provides an event driven monitoring based on inodes, therefore any activity on that link to which inotify file descriptor is added, an event is generated for that link. And in case of directory the event is generated for the all the links in that particular directory [8].

The pseudo code for the file monitor service has the following structure :

```
inotify_fd = inotify_init()
inotify_add_watch(inotify_fd,file_path,IN_ACCESS | IN_MODIFY);
read(inotify_fd,buff, BUF_LEN);
inotify_event *event = (inotify_event *) buff;

analyse_event_and_log(event);
```

`inotify` provides several mask value that allows to do different monitoring with generated events. These could be `IN_ACCESS, IN_MODIFY, IN_CREATE, IN_DELETE, IN_OPEN, IN_MOVED_TO, IN_MOVED_FROM` etc. The event object have the following structure :

```
struct inotify_event {
int      wd;       /* Watch descriptor */
uint32_t mask;     /* Mask of events */
uint32_t cookie;   /* Unique cookie associating related events) */
uint32_t len;      /* Size of name field */
char     name[];   /* Optional null-terminated name */
};
```

In file monitor service, when the event is handled by the event handler, then the event is compared against default events mentioned above using bitwise 'OR' operation. In case if the event is one of those registered by the File monitoring service for the specific file or directory, a call to socket is triggered that sends the wrapped data to the logger service for the later analysis. This monitoring service, was also integrated with a `send_file` method, so when the file was created, or modified, the file is sent to the server running

`recv_file` method alongside logger service. This completes the functionality of the core file monitoring service.

2.2.2 Process Monitor Service

This is also written in C++ and runs in user space that intercepts some of the libc system calls. Figure 2.3 explains the complete process how a system call is being intercepted by Clientpot. The interception is performed in user space. With the concept that allows the preloading of userdefined `.so` files before `libc.so`. After the interception, `dlfun` API calls the actual libc function. In order to make system call interception, Clientpot executes the idea of LD_PRELOAD trick. LD_PRELOAD environment variable specifies the list of user-specified shared library `.so` files that are loaded before any other `.so` file [9], even before `libc.so`. Using the same technique Clientpot creates a wrapper around the system calls, saves the actual system call, executes the wrapper and at last calls, the actual system call and report the event to the logging server.

When a user creates an environment with the name of LD_PRELOAD & sets its value to the address of some `.so`, if any, ELF binary is loaded. That particular binary loads the `.so` file specified by the LD_PRELOAD variable prior to other `.so`. If that `.so` file contains a function with name like `fork`, then that function is called in place of libc's `fork` using LD_PRELOAD. This concept is LD_PRELOAD trick. Clientpot model deploys this technique to make system call interception. Clientpot currently intercepts `fork, vfork, exec, exit, abort`. The call to the actual libc's function was made using `dlfun` API's `dlsym` function, by using RTLD_NEXT as argument.

The pseudo code for the file monitoring service has following structure:

```
int fork(){
/* Saving the address of
   original fork call */
   pid_t (*actual_for)(void) = dlsym(RTLD_NEXT,"fork");
   handle_event_and_loggin();
   actual_fork();
}
```

The main role in this is played by the function `dlsym` and `RTLD_NEXT`, when this `dlsym` function is called with the argument of `RTLD_NEXT`, then function `dlsym` looks for the next declaration of the function, in this case `fork` i.e. where the next `fork` is saved. Since no other declaration is made at this point about the `fork`, in this case, actual or libc's `fork's` pointer will be saved in `actual_fork`. Clientpot uses this type of system call interception technique to create a sandboxed environment.

Apart from these two above mentioned services, there is another module written in python, that runs the actual binaries or pdf in this sandboxed environment, composed of `file monitor` and `process monitor` services.

2.3 Results

Clientpot has been tested with several domains. The testing was done during internship period. It was tested on malicious and non malicious websites. The Clientpot is able to download suspicious files and analyse them. As discussed, suspicious files were run against the corresponding utilities. The analysis of three different attack vectors have been presented here. The one is in which the malicious domain was crawled and in the other, two attacks of two different malware files which were intentionally fed to the Clientpot system.

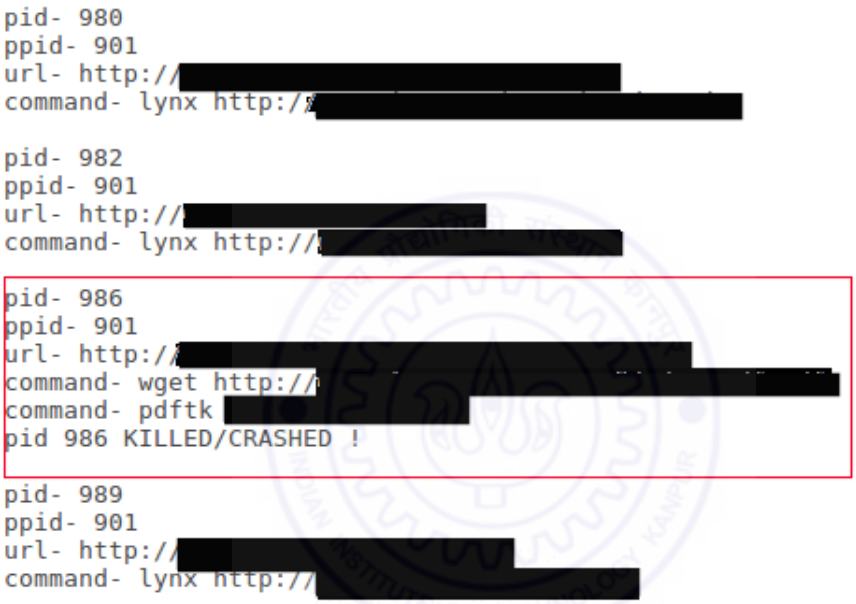

FIGURE 2.5. Log snippet when the Malicious Web was crawled and malicious pdf was fetched from it

The log snippet in Figure 2.5 has been extracted from the logger service. When the URL was crawled, then there was a pdf file that upon downloading and running with `pdftk` application, crashes the system. The file was malicious, that could cause temporary state Permanent Denial of Service, it was marked with the **CVE-2010-0188** category, that could also allow to execute arbitrary code with unknown attack vectors. The logs clearly suffices malicious nature of the pdf file.

Clientpot model can also be used to analyse malicious files individually. Instead of passing URL, Clientpot can be given a binary to analyze. In that case, it behaves like a Linux sandboxed. To test this, Clientpot was inputted with two malicious files, that were **fork-bomb** DoS malware and a ransomware. They both were taken from the malware database of Samsung Research Institute Delhi. The snippet of these attacks is also presented in Figure 2.6

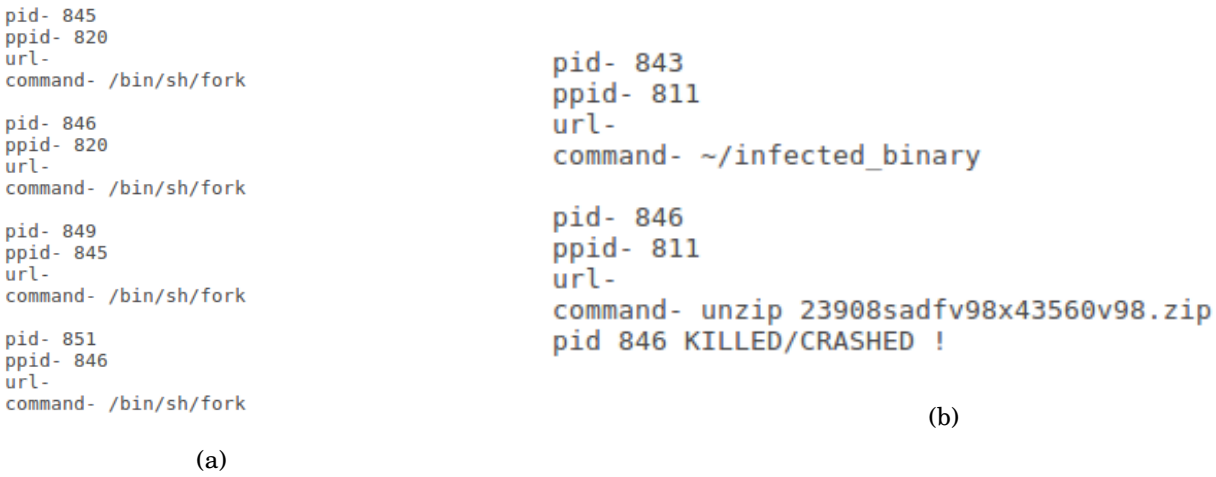

FIGURE 2.6. (a) Logs snippet of Fork Bomb (b)Logs snippet of Malicious Binary

In Figure 2.6(a), it was observed that various consecutive id process, tried to call fork system call. The logger was filled with these types of logs and LXC guest system was frozen and was in the state of temporary DoS. The main point to note about this is that, malware like fork bomb and similar signatures, are difficult to be identified on the basis of signatures. Also during static analysis, they are filtered as benign, because making a system call do not make file suspicious.

In Figure 2.6(b) when the ransomware was executed in the machine, a new file was created which zipped the content of home directory. When the zipped file was tried to unzipped, the system crashed and returned into the state of DoS. This is the most tentative behaviour of ransomware, where a simple binary encrypts the folder or data, and unzipping the data crashes the system. The Log snippet signifies the same.

Clientpot is still in very early stage of development. The main aim of Clientpot was to be lightweight and simple, so that it could run on the mobile devices which do not bear good computation power as modern computers. There are still scopes of developments that can enhance the beauty of the design. Clientpot can be integrated with machine learning based malware analysis so that it may identify the suspicion on the basis of signatures. Clientpot is currently a medium interaction honeypot, and adding signature based functionalities, support for different system call interception and network packet monitoring may enhance the capabilities of the model.

CHAPTER 3

HONEYTOKENS & CVE-2016-5195

Mobile devices have always been an integral part of our daily lives, be it personal or professional. With an increase in connectivity of these devices on the Internet, we have also witnessed a tremendous rise in the number of reported vulnerabilities, which might affect our Mobile devices adversely. Attackers try to identify the weak points to exploit these vulnerabilities. This has given an opportunity to attackers to write malware specific for mobile devices and subsequently exploit these vulnerability to gain unauthorized access or steal sensitive information. The lack of awareness amongst regular users further eases the task for the attackers. To mitigate such threats it is not feasible to run security models such as IDS or IPS since these devices donot offer good computation power. Mobile devides with good networking capabilities once compromised may create to serious issues for the user. such as smart phones or tablets of today's age offers good computational & networking resources which are comparable to that of computing machines, thereby presenting itself as an easy platform to target.

On Nov 15, **Checkpoint** researchers revealed a critical malware, called **Gooligan**. The malware APK [1]was supposed to have 12 different root exploits. However, it was mentioned in [10], that only 2 of them have known CVEs.

In this chapter, we have proposed an idea about how this particular APK malware is capable of exploiting Linux based CVE-2016-5195 vulnerability. We have also presented a very simple yet effective method based on Honey-Token technique to detect breaching of account by malwares like **Gooligan**. This approach as mentioned in[5] claims to capture device breaches with an accuracy of 50% at least.

3.1 CVE 2016-5195

CVE is common Vulnerabilites and Exposure, a Standard for information Security Vulnerabiltiy names maintained by MITRE. CVE 2016-5195 is a `Dirty Copy On Write` based privilege escalation vulneribility in Linux kernel [11]. It exploits race conidition bug in Linux kernel (range from `2.0` to `4.0.3`) to write a read only file. This exploit executes two threads for long enough, to create a situation in which the executions trace would allow a non-privileged user to write a file.

[1]File Extension for android installable, stands for Android Package Kit

3.1.1 Working Of Exploit

The bug is exploited by the by creating a copy of section of file in memory which is called map of the file [2]. This **map** does not get original copy until process makes changes to the memory pointed by map. There are two different threads runs in the main process, that work closely with one another. Both the threads runs in infinite loop and perform opposite actions. The two threads performs the follwing work:

- **Thread1** : keep on executing `madvise(map,100,MADV_DONTNEED)` that tells the kernel that process does not need the address space pointed by `map`.
- **Thread2** : keep on executing `lseek(f, (uintptr_t) map,SEEK_SET); write(f,str, strlen(str));` where **f** is file whose map is created by the thread one. This thread informs the kernel that it wants to write to some specific position in the **map**. Since the map was just a reference to file until now, initiation of write request allocates the content to the map copying the data from the file **f** and then writing to it. Since this map is created by the same user,Äôs process, it is marked with a writeable flag.

The bug was in the code which performs the copying function. The two threads are working closely with each other. In this case, one thread can take the advantage of writeable flag of another thread to execute write on the original copy of the file rather on the **map**.

3.1.2 Consequences Of Exploit

CVE 2016-5195 is a serious Linux privilege escalation vulnerability, with un-patched system leading to serious repercussions. The following are few of the impacts :-

1. A fundamental impact of the exploit could be to write to a read-only file and then behave like a root user of the system.
2. An attacker can modify the important binaries of the system and can bypass the permission mechanism. That may even lead to PDoS [3] and DoS based attacks. Once a machine is compromised, it can be further used to compromise other machines.
3. This exploit can allow us to write data to the read-only mount of a volume of a docker container [12].
4. A non-root user of container or name-spaced user may get root privileges in the container, even if its user is id mapped to the non-root uid in the host machine. It will allow arbitrary writes into the read-only mount system of the host [12].

[2] The file is read only for the user who have opened it
[3] Permanent Denial of Service

Apart from the above-mentioned consequences, a different exploitation technique has also been proposed in this report, keeping in mind that Android systems are no different from the Linux systems. The approach presented in our report allows someone to break the Android application sandbox using the same exploit. And for proof of concept, various logs have been added to the report after filtering sensitive data out of it. A mapping of the work presented has been made with the Android malware called *Gooligan*.

Though Google has released a patch for this bug, but it is only for Android Lollipop. The bug still exists in Android kit-kat model which is not supported by Google anymore. Though the bug has been patched in the latest version of Android, it has been observed in a country like India, many people do not keep their devices updated. Hence, the application of exploitation of CVE 2016-5195 in Android explains the need for keeping our devices updated. Lack of knowledge on the subject which makes us an easy target, further emphasizes the need to understand the security aspects of the system.

3.2 Gooligan Malware

here had been ample history of root based attack for mobile device devices running Android systems. These malware exploits one or the other vulnerabilities to access temporary or permanent root privilege. On June 4th, 2015, Checkpoint discovered multiple instances of a malware that was running 12 different exploits to get the root access. The malware installs itself in the devices, with the help of malicious Windows application. So when the device is physically connected to the system, then the malicious Windows application named SnapPea *SnapPea* [13] injects *SnapPea* malware into mobile devices. The Check point team was also able to identify the count of 12 exploits with the help of Mobile Threat Emulation. While the malware utilizes 12 different exploits only 2 of them have official CVEs [13] **a)** `VROOT - CVE-2013-6282` & **b)** `Towelroot - CVE-2014-3153`.

Checkpoint researchers have mentioned in their report that the Gooligan is the new variant of SnapPea Malware [10]. This malware is assumed to have compromise around 1 million Google Accounts. Still the researchers have not been able to identify correctly how the Gooligan achieved the task of stealing particular G-suite related information. Figure 3.1 hows the detailed description of how the attack happened using this malware and its consequences.

An Android device can get infected when a user installs third party applications from untrusted sources. A device may also get infected by going to links mentioned in phishing messages/emails. After compromising the device, the malware attempts to send the control data to Command & Control Server, which is the central server that stores the information about the compromised devices. Then malware tries to fetch the root-kits from the central server. After deploying the root-kits, the malware mimics user behaviour in the G-suite apps [10].

The augmented module with gooligan also allows to :

1. Steal a user's Google Suite apps authentication tokens.

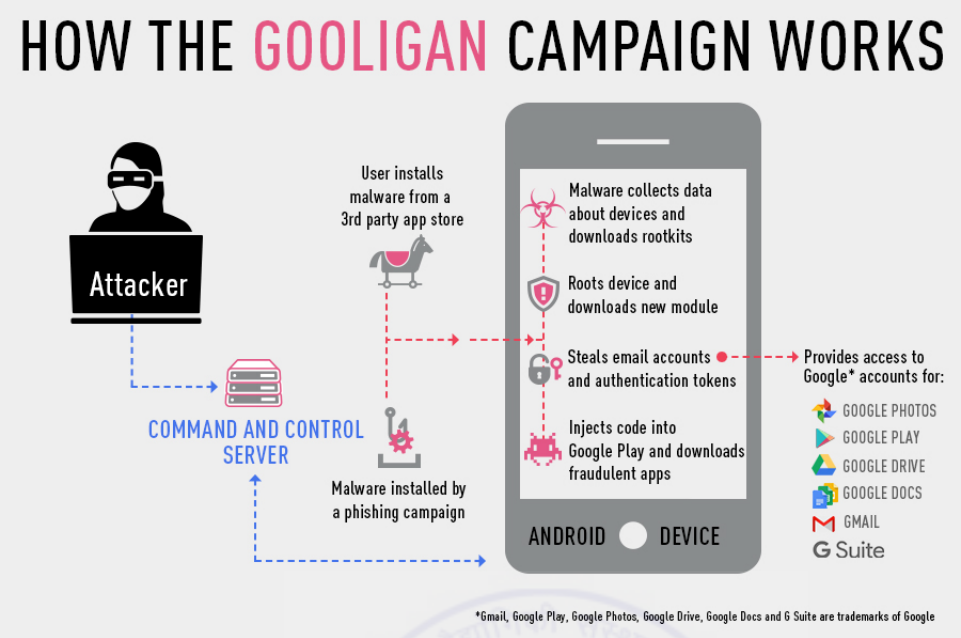

FIGURE 3.1. Attacking scenario of Gooligan Malware. Picture Credits: http://blog.checkpoint.com/2016/11/30/1-million-google-accounts-breached-gooligan/

2. Altering the data with the other applications from play store.

3. Generate revenue by injecting advertisements.

Google authentication token provides an easy access to Google based apps on behalf of the users. Google based service do not store the password into the device, rather it stores just an authentication token, that is generated once when the user enters the password for the first time in the application. Subsequently, to access login based services client application use previously generated tokens to authenticate the user. Once the authentication token is compromised by the attacker, then this token can be used to get access to Google,Äôs login based service on behalf of the user. This allows the attacker to bypass the login process, in case if the user is already logged in.

3.3 Honeytoken

Honeytokens or Honeywords implements a technique that makes password cracking identifiable. This terminology was explained by Ari Juels and Ronal L Rives in [5]. They called it as the fake password that is associated with each user,Äôs account. The Honeytokens are honeypots which prove their worth when they are actually compromised and misused. The approach presented in [5] states to associate a fake password or Honeytoken to each user,Äôs account in `/etc/passwd` with a corresponding entry in

`/etc/shadow`. The attacker, who manages to get access to this file, will try to invert the hashes and using the inverted hashes, he will try to login to the system, and with this login using the Honeytoken, an alarm is raised. This saves the system from being exploited and creates a trap for an adversary. It's intuitive to see with an one-user system, probability of an adversary getting into the system is 50%, whereas, it has been proved in [5] that if an adversary have compromised the hashed file with Honeytokens, with 20 Honey users, there is only 25% chances of adversary getting away while using honeytokens.

The system which implements Honeytoken needs to be integrated with Honeychecker which assists in the working of Honey-tokens. The interaction with the Honeytoken and Honeychecker, in all probability, represents an activity that is malicious and unauthorized. By offering a good level of interaction Honeywords can become an educational research tool.

3.4 CVE 2016-5195 for Android

Section 3.1 presents the overview of what the privilege escalation bug is and how this bug can be exploited using the race condition vulnerability that exists in Linux Kernel version `>=2.x` & `<=4.0`. Since the kernel of Android is no different from the kernel of Linux, this bug can be exploited in some specific versions of Android operating systems as well. The overall idea about how the bug can be exploited is detailed in this section of the report. The consequences of this exploit for Android device are also outlined in this section.

3.4.1 Differnces between Android & Linux

Android system though based on Linux kernel, runs Android runtime Virtual Machine called *Dalvik*. This runtime environment implements application sandboxing. This application sandboxing isolates data and code execution of an application from another application. In Android `su` bianry or superuser binary is not present.There is a detailed process of rooting the Android devices phones using the super-user binary. Once rooted the application sandbox can be easily broken. The main purpose of Application sandboxing is to allow data and code execution isolation between processes but as soon as the device is rooted, the concept behind sandboxing is shattered. Though there is a setuid system call available in `bionic` [4] but it is not available to all binaries. This introduces the concept of Android Security Policies. Due to the enhancements that were offered by Android version >= 4.3 `setuid` is a privilege call that only those binaries can execute who have setuid bit set.

1. **Reinforced SELinux Sandbox** : This allows Android sandbox to ensure SELinux mandatory access control system (MAC) in Linux Kernel. To ensure the concept of

[4] Bionic is Libc replacement for Android

backward compatibility the release of 4.3 allows the use of SELinux permissive mode [14]. The SELinux policies are not enforcing, they can be in permissive mode.

2. **No setuid/setgig programs** : The programs that use setuid/setgid were removed and the concept of file-based capabilities were introduced [14].

The exploitation technique described in the report uses the weakness of both of the above-mentioned concepts. This report will also describe how the task was accomplished and what could be its possible consequences. The similarities of exploit with Gooligan will also be discussed in subsequent sections.

Vulnerabilities that can be used to serve the basis for the exploitation of CVE 2016-5195 in Android versions are listed below.

1. The versions prior to the `4.3` uses `AOSP Kernel 3.4.39`, there were few binaries in `/system/bin & /system/sbin` that were given `setuid` bit set. These binaries were `run-as, wpa_supplicant, fsckmsdos` etc.hese binaries used to execute with elevated privileges and were allowed to make calls to `setuid` [15].

2. From version Android `4.3` and higher, running `AOSP Kernel 3.10, 3.16` or higher, the concept `setuid` bit was changed to SELinux capabilities. Even in this case `run-as` binary present in `/system/bin` was given the capability identified by `CAP_SETUID & CAP_SETGID` [15].

3. The prime reason why `run-as` binary is given such capability is because this binary used by the Android program debugger to run some package via `adb shell`. Upon inspection, it may be realised, this binary have permissions of
   `-rwxr-x-- root shell 9768 2016-04-11 09:51 run-as`.
   Only `shell` group is allowed to run this binary. The `shell` user id belongs to the shell that is started via `adb shell`.

3.4.2 Exploitation

The core concept of CVE 2016-5195 is to allow a user to write a file which is read-only. In this section, an exploit method is explained which uses the weakness of Android Security Model mentioned in section 3.4.1. The Android device Yu Yureka Running Android Version `5.1.1`, Kernel version `3.16.1` and Cynogen OS version `12.1-YOG4PAS8A4` was used for testing purpose. The exploitation was also tested on Hp Slat 7 voice tablet, running Android version `4.4`, Kernel version `3.10`. On both the devices the test was successful.

Figure 3.2 explains the basic pipeline used to fetch the sensitive data out of the mobile device using this exploit.

The following is the detailed explanation of the above :

1. Copying the malicious binary file to the device. This binary is the one that runs the malicious code. In this case, it is the code that actually exploits the vulnerability.

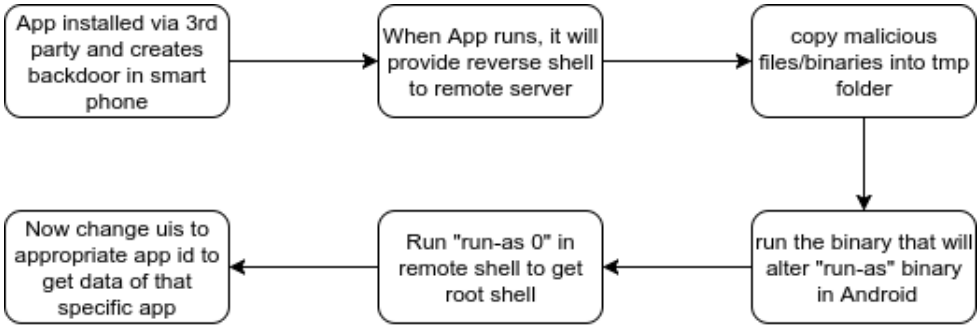

Figure 3.2: The Pipeline to exploit

Along with this code, another code is also copied to the device that creates a backdoor in the device to steal sensitive information.

2. Using the exploit to copy the malicious `run-as` binary to the one present in `/system/bin`, as explained in the previous section that only `run-as` binary have a capability to do `setuid`. This `run-as` binary has been tailored in such a way that it can achieve any id specified in its command line argument.

3. After over-writing if `/system/bin/run-as` is called with specific `id` number. The `shell` promotes to user id as `id`. After getting the id of `system` [5], explore the `/data/systems/packages.xml` file, and look for the necessary application's user-id to spoof with that of shell.
   Each entry in the file have the following structure
   `com.android.launcher 10013 0 /data/data/com.android.launcher`
   The second entry is user application's user-id.

4. Running `run-as 10013` now change the user-id of the shell to the application's user-id. And after getting the application user-id, the sandbox of the application is broken. Once the sandbox is broken, sensitive data from the application can be sent to the server using the backdoor created as mentioned in the first step.

3.5 Demos : Fetching sensitive data from two Applications

This Section will demonstrate the procedure to break Application Sandbox and stealing the sensitive data from two different applications. First one is peer to peer chat messenger application **Whatsapp** and another one is **Gsuite** application that uses OAuth token for authenticating a user. The following are the steps involved:

[5] Android user **system** have numeric uid of 1000

1. Cross compiling the `exploit.c` file and pushing it to the Android system using `adb push`. After this step, read-only write exploit file named as `CVE-2016-5195` is copied into the location `/data/local/CVE-2016-5195`. After this, file modified `run-as` is copied into same location. This `run-as` is tailored to change the uid of shell to other application's user-id.

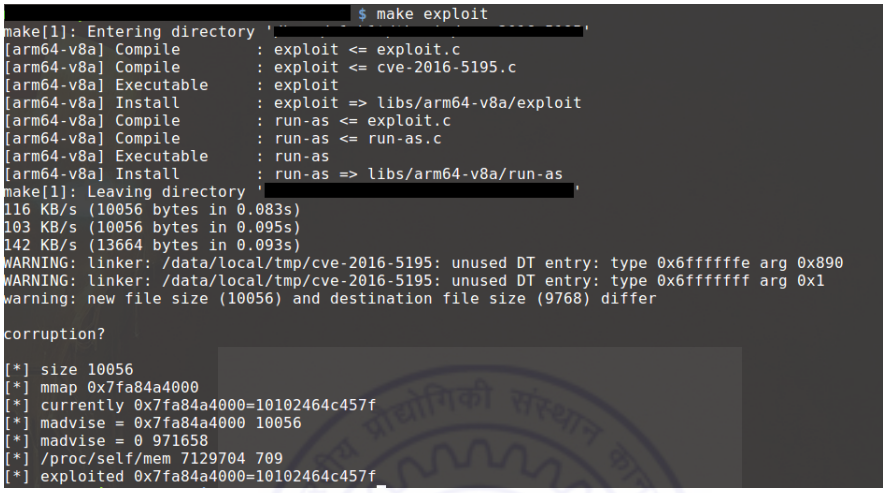

Figure 3.3: Cross Compiling and pushing the binaries into the Android Device

After the binaries are copied then the system checks if the copy operation on run-as is successful or not. If it is successful then shell prompts **exploited**, can be seen in the last line of figure 3.3

2. On pushing and copying the necessary files to device, it is required to check if `run-as` has been copied successfully, and if it copied then whether it is able to change the user-id of the process or not. On getting into `adb shell`, it could be checked via executing `run-as 0` and checking the id of the shell process. As shown in figure 3.4.

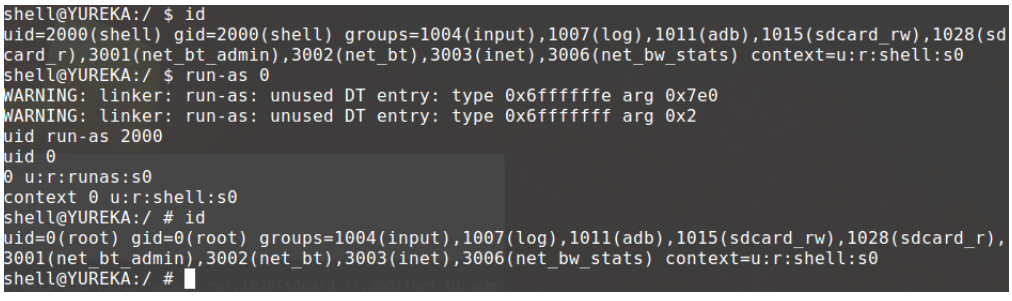

Figure 3.4: Executing run-as 0 to see if shell user achieves temporal root or not

Executing `run-as 0` reveals that shell user fetches temporal root. Device have been compromised, and is ready to break out the sandboxing of `Dalvik`.

`1000` is the user-id that is allocated to the `system` user. This user has all the previliges that executes all the task that are required to keep the system running. `System` user keep track of installed application and allocates user-id to each of them.

```
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 3.5: Executing run-as 1000 shell user escalated to `system`

3. The next step is to identify the user id of the process from which sensitive data is to be stolen. In this case, it‚Äôs the `whatsapp` application, and there is a file in `/data/systems/packages.xml`which is accessible to the system user, which stores the user-id mappings of application. After obtaining id of the system, while inspecting the file, it was observed that in the current device, `whatsapp` holds the user-id of `10096`.

4. he next step will be to execute `run-as 10096` to get the shell with the rights of `whatsapp` user-id. Most important point is to observe here is that the `run-as` can be executed from a shell or root user prompts. If the shell is running as system, it first needs to come to shell user. Pressing `Ctrl + d` will drop the shell from the system user to shell user. Once the `whatsapp` userid is obtained, data from the whatsapp application can be sent via a backdoor to other systems. And shell gets the access to the content of `whatsapp` directory at `/data/data/com.whatspp`. Figure 3.6 corresponds to the same.

5. Cross-Compiling & Installing the binary into the system named `sendf` that creates socket connection to the server and send the data from android to server. Figure 3.7 corresponds to the same.

   Starting the corresponding `recv_file` server in host machine at port `12345`. This server will receive the file.

6. A user can choose what kind of sensitive data that he/she wants to extract from the system. This demo will fetch `msgstore.db`. This database store the information of entire conversation of user with other users and their contact information. The

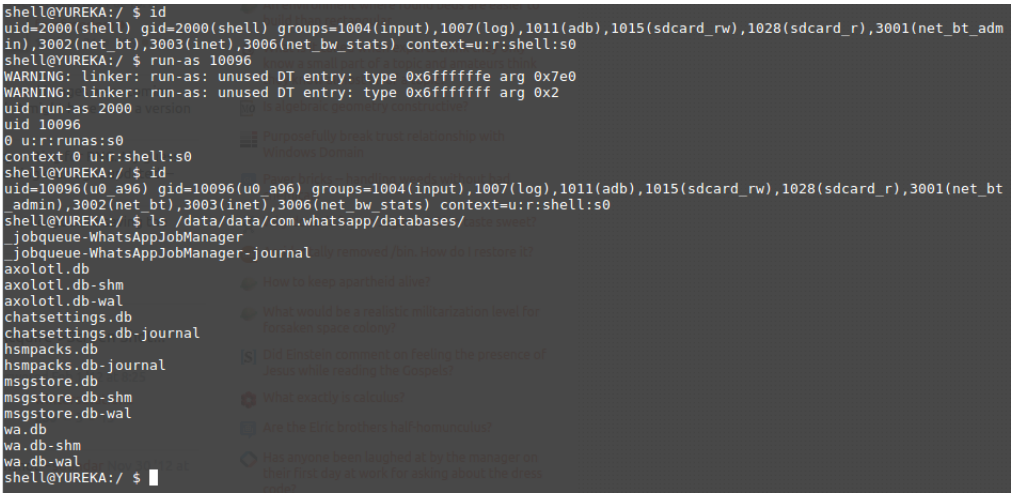

Figure 3.6: Getting `whatsapp` user-id and getting into application's directory.

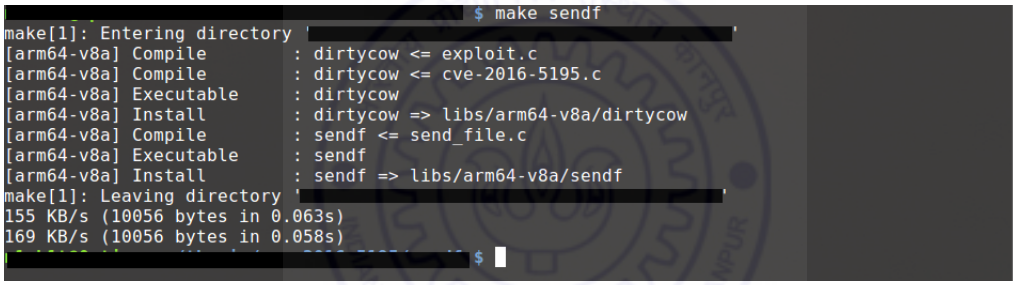

Figure 3.7: Installing backdoor in using `sendf`.

`sendf` binary will send the data to the server.
`/data/local/tmp/sendf <FILE_NAME> <SERVER_IP> <SERVER_PORT>`

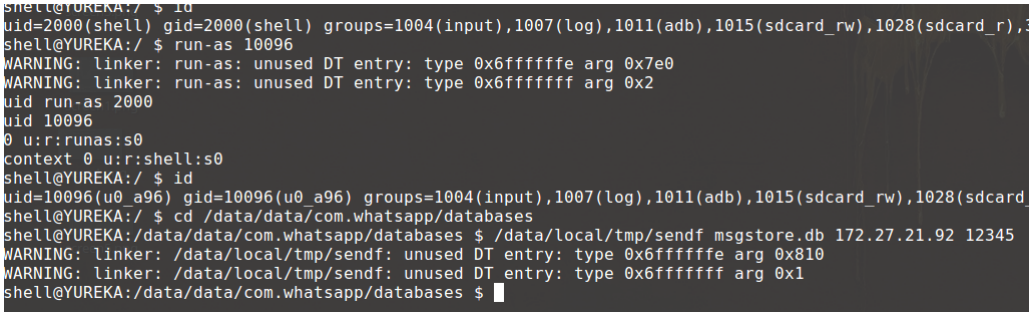

Figure 3.8: Sending `msgstore.db` to remote server

The corresponding file is received at server end.

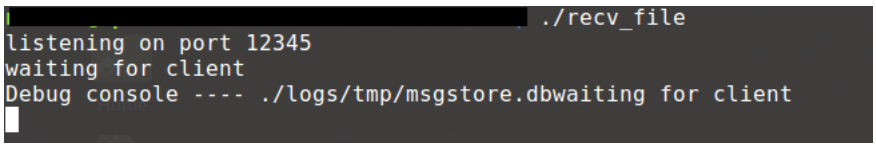

Figure 3.9: File received by the server

7. To confirm the theft, the database was opened using the `sqlite3` dbms. And the first few result are shown in the Figure 3.10.

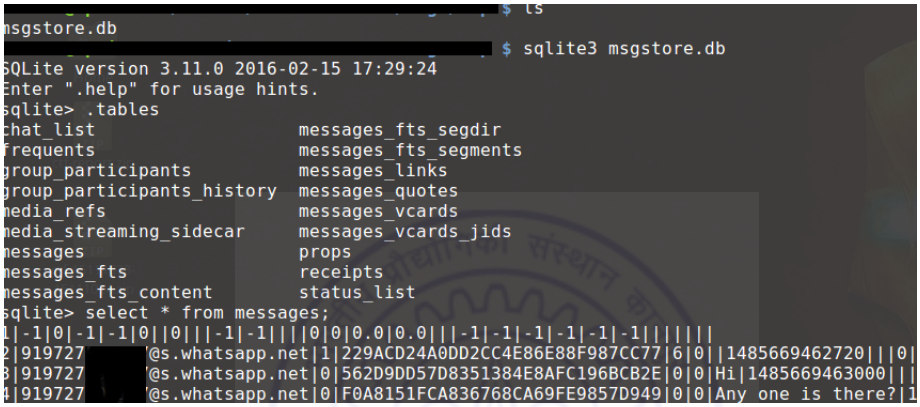

Figure 3.10: Decoding received file with `sqlite3`.

8. Next the System Google Auth Tokens will be sent to the server. The Google OAuth tokens are used by the Google Applications to authenticate a user on a certain application, instead of prompting for password again and again. These tokens are accessible to the `system` user and are stored in directory located at `/data/system/user/0` in a file named `accounts.db`. For each Google,Äôs application there exists a separate OAuth token. Again the same sendf will be used to send the data to the remote machine.

9. Processing the accounts.db file in sqlite3 database, reveals all tokens associated with each application.

The steps from 1 to 9 depict the use of an exploit to fetch necessary data from the devices using a malicious binary that are pushed during the step one. During the entire process, there has been a change of user ids from one user to another. The brief description has been visualized in figure 3.13.

3.6 Similarities of Gooligan & Exploit CVE 2016-5195

As it can observe that in step 9 figure 3.12, the exploitation technique was able to get the OAuth tokens from the Google Installed application. The exploitation is only valid

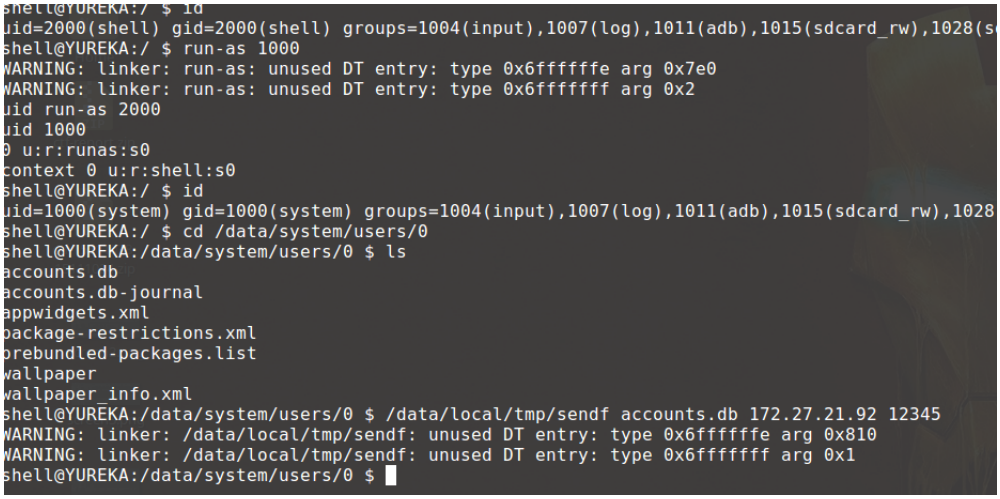

Figure 3.11: Sending data in `/data/system/user/0` to remote.

Figure 3.12: OAuth tokens processed using `sqlite3` database.

in Android Kernel version from `3.4 to 3.10`. Considering the above two points and the point mentioned in [10]. Its not difficult to realize that Gooligan may have used the described `dirty_cow` exploit to Steal Google OAuth tokens and uses these OAuth tokens to generate the Access tokens for accessing the application data specific to user. Check point researchers have identified that Gooligan Malware would have used two other exploits viz `towel root` and `Vroot`. But these rootkits are the one that make root to the system permanent as they installs `superuser su` binary to the system. However traces of such `superuser su` binaries were not identified in the Gooligan compromised system. It was also observed that Gooligan may have used different exploits other than `towel root & v-root` [10].

From the above points, it can be concluded that Gooligan may have used temporal root technique that can be exploited by the exploitation technique discussed in previous sections of this report to get the temporal root. Instead of exploiting the device via `shell` user, Gooligan exploits the same technique with the application. To exploit the same technique via application is not an easy task. The binary `run-as` is only readable to `shell` group and root user. And the shell user is only accessible to developers.

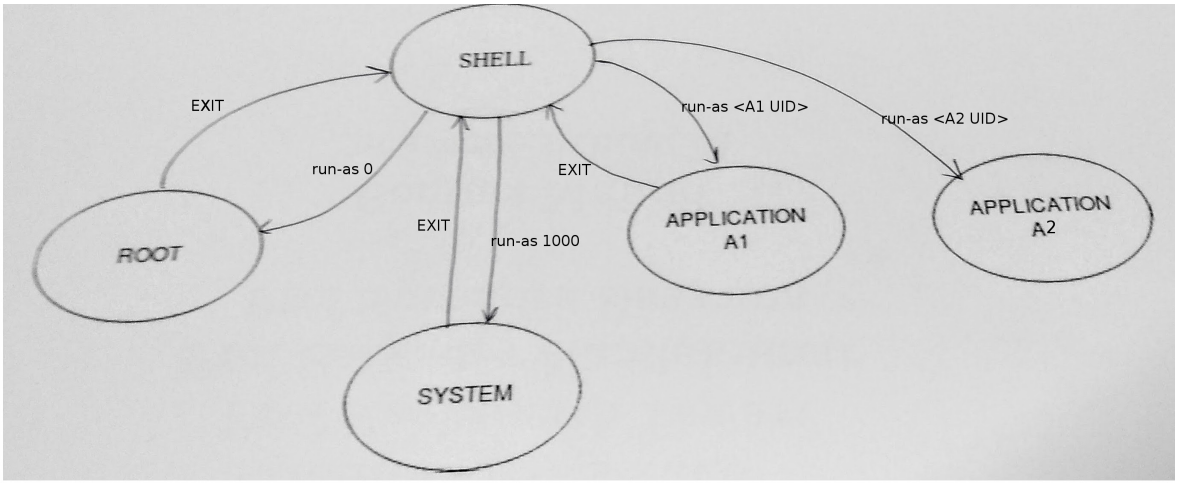

FIGURE 3.13. How the user-ids can be changed from one user to another

Nishit Majithia and I have identified the weak point to exploit the bug from within the application. To exploit the bug from within the application, the core idea is to patch `libc` in the memory file of the current application using the exploit, in such a way that it calls patched `run-as`. Once this is done, A Gooligan like malware can be created. This idea will also complete the picture at figure 3.2.

3.7 Honeytokens based Identification of breached devices

In this section, a theoretical approach for capturing un-authorised access of OAuth tokens is presented. As mentioned in section 3.3. Honeywords or Honeytokens can be used to identify the accounts that are breached using Gooligan. The idea is quite similar to that of identifying the password cracking by adding a fake entry in `/etc/passwd` file. In this case, as it is mentioned that Gooligan targets only Google OAuth tokens. To counter the same, i.e. to identify the account breaching, fake Honeytokens similar to tokens that are present in accounts.db can be added in the directory `/data/system/users/0/`. In simple words, a fake OAuth token for a nonexisting application can be installed in the device directory. And once the device is compromised and tokens are accessed, the alarm can be raised. Usually, the attackers do not check if the application is installed in the system or not. They just try to use the tokens to steal every bit of information. The Access to the fake token can be captured, and the alarm can signify the account breaching. So there are two main things to observe :

1. If a new entry of OAuth token is created for the non-installed application in the system, there are `100%` chances of exploitation of Honeytoken to entrap attacker.

2. If a fake entry for the corresponding installed application is created, as discussed in section 3.3 and in [5], there are 50% chances of identification of breaching.

The corresponding Honeychecker OAuth token access service can be written, which can store the parameters required to identify the account breaching. These parameters can be a list of installed apps or user's Google app suite information etc. And using this way the identification can be made for the device.

CHAPTER 4

HONEYPOT MODELS

Honeypots are the tool or systems used to capture the tactics of attackers. These systems are designed in such a way that attacker tries to attack them. Even if they are compromised badly, they do not affect the production servers. They are basically deployed to capture the attacks and the scripts used by attacker. Honeypot has served industry to capture a different types of malware. Apart from collecting malware, they also provide information about the security loopholes in the system. Even after such awareness of Honeypots in industry, implementation and deployment part of the Honeypot is still lagging. There exist different models for Honeypot, but they are very much constrained to the environment on which they are deployed also level of interaction they offer.

The study presented in the report uses centralised log management system built on the top of open Elasticsearch Logstash & Kibana stack. The system is designed to monitor live traffic. The complex query handling ability of Kibana, to query Elasticsearch Database, is used to generate the attacking pattern for several day of inputs. The sample has been collected after one month of deployment of different Honeypots on various different geographical location other than India. These locations were New York, London, Singapore, Toronto & San Francisco.

Following technologies have been used to develop range full of Honeypots:

1. **Linux Containers** : Containers offers operating system level software virtualisation that allows one to run different Linux system on same physical machine. These systems are isolated in terms of namespace and control groups.

2. **LXD** : They offer Linux containers hypervisation which allows server to communicate with other containers via rest api. It also offers good management console for the different containers.

3. **Docker & Docker Compose** : As container offers the level of virtualisation for the operating system, Dockers offers virtualisation to the layer of application only. They allow to run each application in isolation. Instead of separating all namespace and control groups, it separates few of them. Docker-compose on the other hand offers network of Dockers. It allows to create network amongst the applications so that they appears to exist in the same virtual environment.

4. **ELK stack** : Built on the top of `Docker compose` environment. It allows to manage different logs collected from different Honeypots. The kibana interface can be used to generate the analytics about the attacks as well.

4.1 Objective

Honeypots have been used extensively to capture the attackers in order to identify their the tools and tactics. The main intent of honeypot is to keep the attacker in the system for as much as possible. The main issue with this is, as soon attacker identifies the presence of any monitoring tool, then either it tries to remove those monitoring tools out of the system or simply leaves the system by damaging system. There have been sequence of attacks in which the honeypots have been deployed to capture variety of scripts but attacker damages the system after identifying the presence of honeypots [3].

Apart from this concept, deployment of Honeypot has been a challenging task and no such generic honeypots exits that are easily deployable and widely applicable. Honeypot offers exploitable service, if not exploitable then compromise-able. Identification of level of how much it can be exploited and compromised is a challenging task for the type of service that are offered by Honeypots.

The following are the major scenarios need to be targeted while the deployment of Honeypots :

1. To keep the attacker busy for the longest time possible in the system.
2. To keep the system secure enough so that attacker never try to compromise other machine using this machine.
3. To keep system different from the available exploitable technique i.e other than the CVE exploits database that are associated with the each version of different services.

Honeypot deployment in an incremental process, whose capabilities keep on enhancing with the attacks it faces. With each of different honeypot model, every scenario is being targeted and incrementally enhanced. After the analysis of bag full of attacking scenarios few honeypot model with their design issues have been proposed.

After deploying a bunch of Honeypots in the public network lot of data have been collected and the analysis have been presented in the paper.

4.2 Proposed Honeypots

4.2.1 System Overview

In this report, three different Honeypot models have been proposed. Each one targeting a different scenario. Each honeypot has been defined by its design, implementation and enhancement over the existing ones. All of the presented honeypots have been integrated

with the log management system developed on the top of Docker compose framework. This report will also discuss the issues associated with each honeypot and their security threat model.

The entire system have been deployed on two different public IP with docker and lxc as the base of honeypots. The figure 4.1 describes the entire system snapshot, illustrating the model of different honeypots The main system runs ELK stack, docker server and LXD server. LXD server interacts with the LXC client which is HoneyFARM. Docker server is used to communicate with the docker client applications viz HoneyFTP and HoneyWEB. Docker Compose system running in physical machine is that serves ELK web server. Physical system was allocated two different IPs with in the same network. The Entire system consist of the firewall, Linux Ubuntu 16.04 LTS Server, Elasticsearch, Logstash, Kibana, one virtual machine running HoneyFARM, two virtual isolated docker client application running HoneyFTP and HoneyWEB.

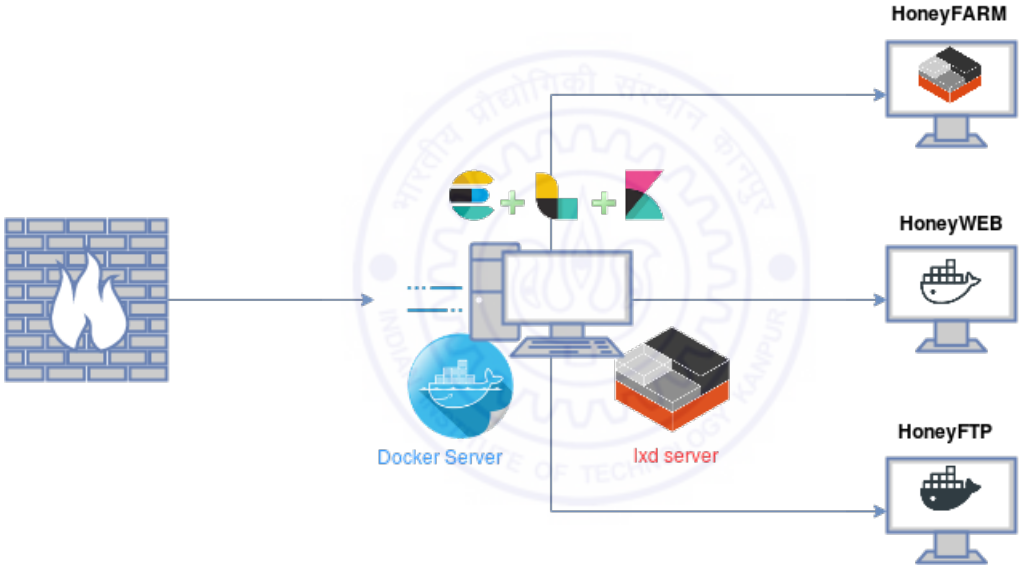

FIGURE 4.1. Overview of Entire system

Some Honeypots were deployed using system consist of Linux based software virtual environment, hosted on 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. Some of the honeypots were also deployed on cloud infrastructure of digital ocean in order to collect logs from the attacks on different location such a New York, San Francisco, Toronto.

4.2.2 HoneyFARM

HoneyFarm system traps the attacker in network of SSH service Honeypots. It has been implemented in such a way that attacker spends most of the time in exploring the internal network. This system creates 4 different software confinements that serves ssh

honeypots and then traps the attacker into one of them. When the attacker tries to try do manual brute forcing attack using this system, it is then redirected to the another honeypot. Study from the implementation of simple and restricted honeypot system deployed on the top of open source and most widely used ssh proxy `HonSSH`, extracts that most of manual attacks of the system are those in which attacker enter a system and tries to inject the worm in the network. Keeping the same idea in the mind, HoneyFARM model have been implemented. When the attacker explores the internal network, tries to do something malicious, everything is captured and can be used to train Rule based IDS system. All logs created by the attacks are managed parallel at real time system which is not approachable to attacker.

4.2.2.1 System Design

HoneyFarm model have built upon the three major components. They are open source SSH proxy, Open SSH source and open source virtualisation technique i.e Linux containers. This system creates a virtual network of lxc guest machines and these machines are connected to same bridged network. Apart from that, the system implements a ssh redirection. When anybody from the system lets say in `sys` tries to do ssh to any other machine across the network, then he is redirected to the machine named `router` that too in same bridged network. In this way the attacker is nurtured under the observation while he is spending good amount of time in the network.

In the present system virtual network constructed had 4 compute nodes. These nodes were the part of one bridged network. Figure 4.2 defines the same ssh redirection view of the network from the attackers view point.

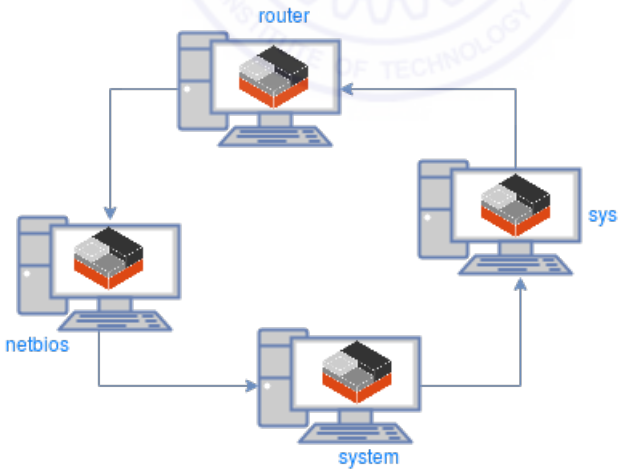

FIGURE 4.2. Compute Nodes in HoneyFarm and their virtual connectivity of target ssh system

In the model, there are 4 virtual system running in same physical machine. This physical machine runs ssh proxy. When the attacker gets in to the machine, the following execution happens:

1. Attacker targeting ssh attack on physical machine is captured by the proxy. Proxy had already configured with the target of virtual machine named `system`, as shown in Figure 4.3.

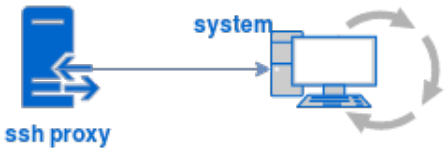

FIGURE 4.3. The targeted virtual system of ssh proxy

2. Proxy transfer the request to the `system` virtual machine.
3. When the attacker tries to do ssh from `system` then he is redirected to other replicated device named `sys`, from `sys` to `router` and so on. The complete ssh trace is depicted in figure 4.2.

This system makes the attacker to spend good amount of time in the system.

4.2.2.2 Implementation

The system have been implemented with three open source technologies.

1. **HonSSH** : Free and most widely used ssh proxy based on the idea of ssh honeypot `kippo`. This proxy site in between the honeypot and attacker. The proxy is implemented as transparent layer. This proxy runs in physical machine. And routes all requests honeypot named `system` see Figure 4.3.
2. **Customised Ssh Client** : The fork of open-ssh client, that is customised to redirect the ssh requests from the container to other. The ssh client have been tailored to redirect requests from one honeypot to other, thereby creating virtual network to create attacking surface. And deceiving attackers to spend good amount of time in system.
3. **Linux Containers** : They have been used to build virtual system. The container hypervisor `lxd` is used to control and manage container via command line interface.

4.2.3 HoneyWEB

Customised Web server honeypot, initially deployed using identify the attack timings between the attack on ssh and web. The main aim of a system was to enhance the capabilities offered by the existing web based honeypot which is `glastopf` which simulates a web server. And also to identify the attacker similarity in different honeypots together. This is lightweight system with simple implementation using exiting open source tools and technologies like `nodejs' express & http-proxy`. This first clones a website to

a volume that is attached to Docker and then static serve that directory by creating a proxy. This proxy monitors the connection and loggable data about the connection. This proxy can be customised to log what data that we want to log. The system is built using Docker Compose environment. For clonning a website, open source tools `httrack` is used and the data is then served using the `node js` based http-proxy and http-server.

4.2.3.1 System Design

Application is built on the top of docker compose environment. The system runs two different docker systems in isolation that are integrated with the help of docker compose build. This creates a nertwork of two different machine connected via `compose` isolated from the actual system on which they are running. The proxy open the port to the system, the interaction of the user takes place via this port and the request is then logged, manipulated and handed over to the another docker which serves the web site statically. The logs of the system are mounted to particular directory in the host of docker server. Figure 4.4 gives the entire description of the model. Also as shown in the figure, proxy sitting in between is responsible to logging and also extends the safer model for browsing. `http-server` docker access the directory mounted to it and servers the website using by looking for `index.html` in it.

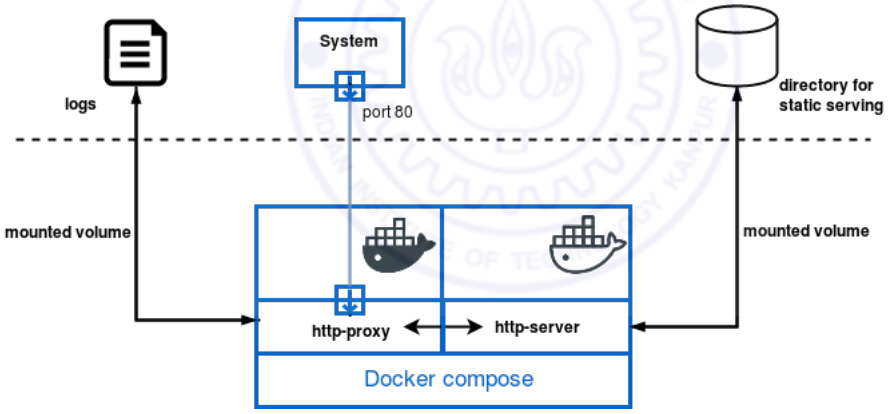

FIGURE 4.4. System Design of HoneyWEB

In this model, the port `80` of the docker host system is forwarded to the port http exposed port of the `http-proxy` docker. Then the port 80 of http-server docker is linked via `docker-compose` to the one. When the attacker targets the system for port `80`, in detail the following execution happens :

1. Attack sends the malicious link [1]. The request URL is forwarded to the server to `http-proxy`.

[1]Malicious Link is the URL of the website that attacker send to exploits some vulnerability of the web server. This link can also contain a re-direction to some other malicious website

2. Http-proxy logs the request and process it so that it may not harm the system running in docker of http-server.

3. Http-Server handles the request and sends the response, in return http-proxy checks if the response is `404 Not Found`, if so then, it initiates the request to the `index.html` to http-server.

4.2.4 Implementation

The HoneyWEB system is built using httrack, docker, docker images, docker compose, node.js and node.js express. These open source technologies have been integrated in such a way that they offer the entire functionality of the system depicted in Figure 4.4.

1. `Httrack` : Open source web site clonner, integrated with the service that clones the website in to the directory mounted to `http-server`. This is served as dummy website by `http-server` docker.

2. `http-proxy` : A `node.js` based service running in docker, logs the information in the docker host system. The logs are maintained in the volume mounted. The docker `alpine` image is used for the running this service in docker. This Linux based image is a security-oriented, lightweight Linux distribution based on musl libc and busybox. This decreases the running space of docker and also its image size from around 500MBs to 27MBs. This service implements the open source node.js module to run a web server based proxy. The functionalities offered by the proxy are quite limited to the application only.

3. `http-server` : Implemented using docker `alpine Linux` image that runs the `node.js` service and its open source `static-server` module to server the requests. The website clonned using `httrack` is saved into the directory which is mounted to volume in docker.

4.2.5 HoneyFTP

HoneyFTP offers a light weight honeypot platform to monitor and analyse various FTP based attacks and capture some malicious executables. It instead of simulating a FTP server like other honeypot solution do, creates a docker that actually runs an FTP server. This system upon docker startup creates a isolated environment and runs the service that creates two different demons running on different ports. The service have been specially created to to capture internal port scanning attacks which are called `ftp bounce attacks` [16]. This system serves the vulnerability in safe environment to allow attacker to do port scanning of the internal network [2] [3]. To get the idea about the tool and tactics of attacks compromising the FTP server, HoneyFTP was build and deployed.

[2]The network of the docker guests. The system running the ftp server is connected with the other docker in the system

[3]One of these docker were allowed to do ssh from any of the compromised docker in the Network

It capable of capturing the attacks of foreign host scanning by compromising the service and also it does not allows attacker to do port scanning for the network outside the range of classful private IPs.

4.2.5.1 System Design

The fundamental design of the system is depicted in the figure 4.5. The service for the FTP server with two different daemons was executed in the docker network within the docker default bridge. The system is made to connect with the two different docker guest machine as well. One of these docker machines allow to do brute forcing of ssh attacks as well. This invites the attacker to scan for the internal network after observing the opening of ssh port of some machine, to attack for ssh brute force on it.

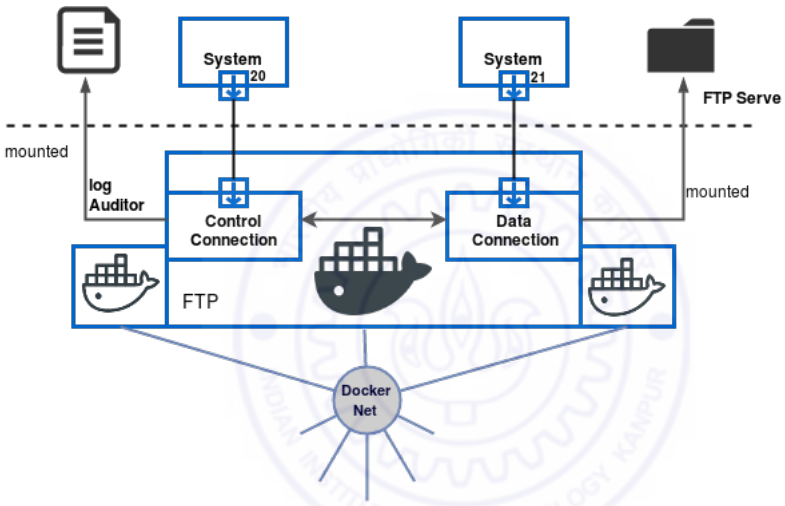

FIGURE 4.5. HoneyFTP running in docker network and offering surface for capturing FTP bounce attack.

The system deploys three docker machines. One of those three machines runs Docker python services that offers offers the similar to ftp service to the ftp client. The other two docker machine have been added to the HoneyFTP system to make few machines available in the docker internal network for attack to scan those machines. The machine that runs python service have been linked with the data port and connection control port of the actual machines. The other two machines are isolated with respect to the host machine but offers ssh login access from any of the docker machines in the network that is specified as `Docker net` in the figure 4.5.

4.2.5.2 Implementation

HoneyFTP model has been implemented using the docker, docker network, python open source `pyftpdlib` etc. The docker machine is mounted to a directory in host machine, that serves as the base to the FTP directory serve. In the same machine there are two

different service running, one saves the file in the docker host system and other logs the activity of the user in host. The detailed description of different service is given below.

1. `pyftpdlib` : This python ftp server library provides a high-level interface allow to create scalable ftp servers. The library have been tweaked so that it allows to make port access calls of foreign host machines to capture FTP bounce attack.

2. `file receiver and monitor service` : This service captures the malicious files `put` by the receiver on this machine. The ftp share library is continuously snooped with the file create event, as soon as the directory receives a file create event, the file is sent to the server[4] and server saves the file in docker host machine.

3. `Augmented Docker machine` : Two augmented docker machines have been added to the network of the docker machines to allow ssh brute forcing attack locally.

4.3 ELK Stack

ELK stack build on the top of docker compose environment visualises variety of data collected from different Honeypots. This consist of Elasticseach, Logstash & Kibana. The data is pushed to ELK stack by logstash pipeline. The complete picture of the system is show in the figure 4.6

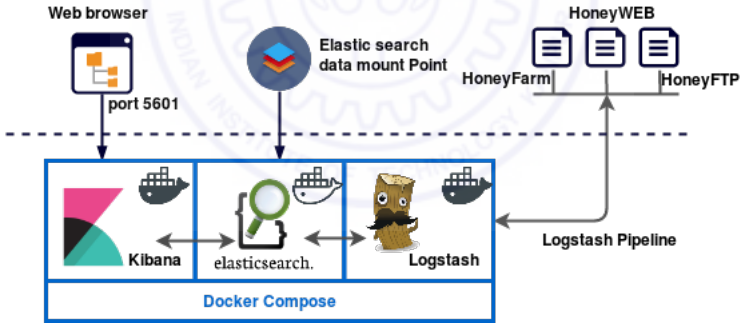

FIGURE 4.6. ELK stack pipeline for analysis of data captured from Honeypots.

1. **Elsticsearch** allows to store the data in form on indexes using `nosql` database. It offers good real time scalability. It provides powerful rest api to communicate with the database engine. The elasticsearch has been used as storage to index the data gathered from all Honeypots.

2. **Logstash** Log stash allows to push real time data to elasticsearch database. This service keeps track of the modification made to the log files. As soon as the monitored files are added with more data, the Logstash pipeline pushes that data

[4]File Monitor server, similar to one that of Clientpot mentioned in chapter 2.

onto the elastic search database automatically. Corresponding `logstash.conf` file needs to be written in to the logstash conf directory.

3. **Kibana** is an visualization engine that fetches the data from the Elasticsearch database and visualise it in different forms. It also allows the user to view logs in runtime, with the attack payloads and their locations. It can used to perform complex queries into elasticsearch's database. It is the only system whose port is exposed to the main docker host machine for the end user to view the statistics.

CHAPTER 5

ANALYSIS OF ATTACKS ON DIFFERENT HONEYPOTS

This chapter includes how attacker penetrates into the honeysystems comprising of Honeypot, HoneySSH, HoneyWEB and HoneyFTP which was deployed at geographically distributed locations upto 720 hours and how to organise the information that has been extracted from these attacks for analysis. Honeysystems had been deployed using NKN pubic network and cloud infrastructure. Initially, HoneySSH and Honeyweb was deployed in India using existing NKN public network for 480 hours. for next 240 hours, The HoneySSH was deployed using cloud servers at **India**, **San Francisco**, **Toronto** and **London**.Honeyweb was deployed using cloud servers at **London**, **Singapore**, **San Francisco**, **Toronto** and **France**. HoneyFTP was deployed for 240 hours only [1]at NKN network.

Tactics used by either novices or experts all are logged in runtime to a main server which further have been refined to extract knowledge that aims to identify most aggressive IPs [2], most common behaviour of those IPs and the pattern amongst them. Analysis also includes the access pattern across different geographical locations for different locations. Different analysis have been presented from Honeynets [3] point of view and Honeypot point of view.

5.0.1 Attacks : Wider View

Analysis has been done for last 240 hours which scales down the count of IP addresses. During last 240 hours, There are total **13741** unique IP addresses has been observed and distribution of these IP addresses has depicted in the Figure 5.2. Out of which only **6021** unique IP were performing attacks on deployed honeysystems i.e. HoneySSH, HoneyWEB and HoneyFTP [4] and distribution of the same has depicted in the Figure 5.1. . The attacks attempts were much greater than this count. This count only contains the attack attempts that were successful. It can be observed that SSH seems to be the most targeted protocol amongst FTP and HTTP. In case of SSH, successful attempts are those that allows the attacker to get into the system.

[1]HoneyFTP was generating lot of foreign address scan to other Networks including internal Network
[2]The IPs that make attempt to access Honeypots for longer durations
[3]When the logs are considered for all honeypots together
[4]HoneyFTP was deployed for 10 days only, so the other Honeypots were scaled down to 10 days

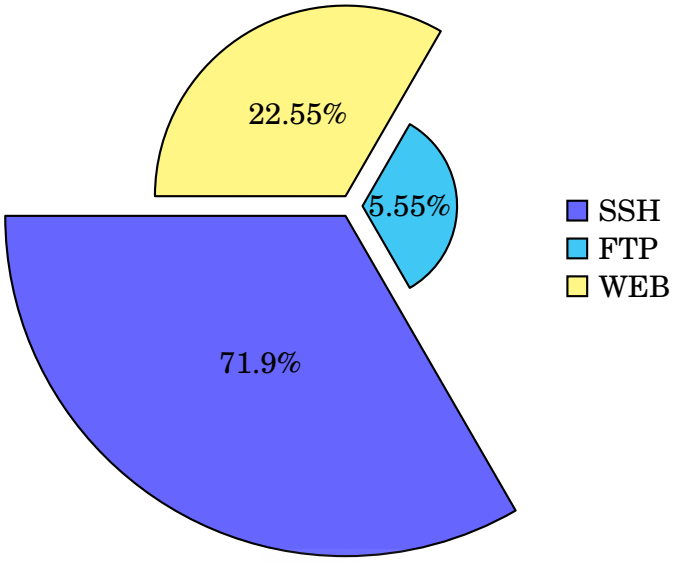

Figure 5.1: Average Number observed IPs during the duration of 10 days targeting ssh,web and ftp Honeypots

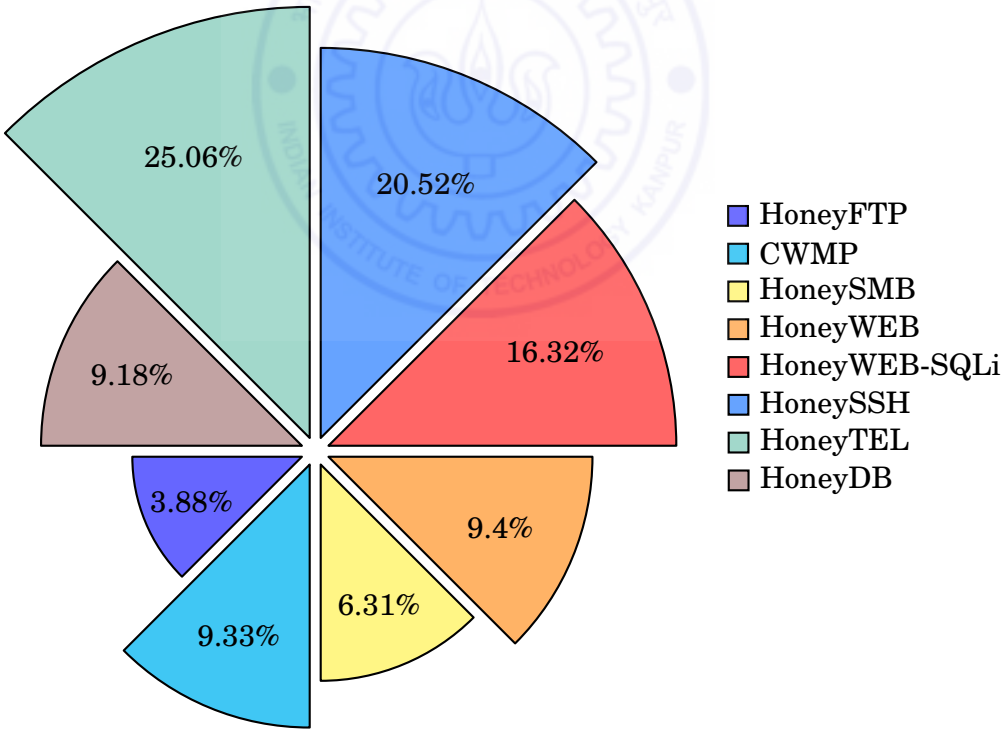

Figure 5.2: Average Number observed IPs during the duration of 10 days targeting all Honeypots

5.1 Attacks on HoneySSH

This section will present the detailed analysis of the attacks captured by HoneySSH.

1. The total count of the Unique IPs that have been recorded by the Honeypot HoneyFARM/SSH system was **1246** [5]. Out of these **1246** unique IP addresses, a few IPs aggressively tried to login to the machine with the maximum attempt of around **71158** and many IPs tried for fewer attempts. The detailed split is provided in the table 5.1. The number of attempts made by aggressive IPs have been listed in table 5.2.

Table 5.1: Attempts made by one IP to successfully identify SSH user's password

Number of Attempts	**IP count**
Attempts <1000	1215
1000 ≤ Attempts <10000	24
Attempts ≥ 10000	7

Table 5.2: Number of attempts made by the IP whose count was ≥ 10K

IP address	**Attempt Count**
61.177.172.24	12329
61.177.172.17	12680
61.177.172.55	15734
183.214.141.104	27170
116.31.116.10	38128
58.218.211.78	49315
58.218.204.223	71158

Few observations made by the this collected data i.e IP to attempt Count:

a) The IP address that targeted the SSH the most were 61.177.172.24, 61.177.172.17, 61.177.172.55, 58.218.211.78,58.218.204.223 belongs to **AS4134 CHINANET-BACKBONE** organization. They all belong to the same network that is present in China. Except 183.214.141.104 that was not from the same network but belonged to China.

b) The timing difference between the attacks from the IPs belong to network 61.177.172.* mentioned in Figure 5.2 was also small. This indicates that these machines may be compromised and tried to do brute force in parallel over this network. This may be either intentional or planned.

c) IPs that seems to belong to the same classful network, their attempt count was almost similar in number. There were cases that one IP did brute force

[5] These IPs include those who were able to log in successfully .

on the SSH protocol and saves the password and other IP belonging to the same `*.*.*.0/8` network did only one attempt to get in via SSH. The total count of such IPs was **221**.

2. Total attacks captured by HoneySSH belonged to one of the three categories. These categories are `Scripted`, `Terminal based`, `SFTP based`. SSH protocol allows to execute the command without fetching remote terminal, and these types of attacks are mostly **scripted**. There is log of attempts in which attacker fetched the terminal and executed commands in that terminal, such type of attacks are **terminal** based and lastly there are a few evidence that some of the IPs attempted to put malicious files via **SFTP**. The statistic about the count of such attacks is mentioned in the Table 5.3.

Table 5.3: Attack type and their Unique IP count

Attack Type	Count
Scripted	115
Terminal	67
SFTP	10
Only Login to the system	1054

   a) The maximum attempts made by any IP for SFTP protocol was **3**, for fetching the terminal was around **71** and scripted were **54**.

   b) This gives the idea that which protocol variant of SSH is most targeted and How and what IDS need to learn.

3. Except for the NKN network, HoneySSH was deployed for the duration of 10 days at 4 particular locations on a cloud. These locations were **San Francisco, London, India and Toronto**. The total unique IPs captured by all of them were **248** and total attacking IPs were **273**. Few IPs attack multiple servers. The distribution of these attacks on different locations is depicted in Figure 5.3.

   a) There is a large variation of attacks in Indian Server in Bangalore compared to the attacks in other servers at locations other than India. In 5.3 around 80% of the total attacks in 10 days were captured by the Indian Server only.

   b) **India** is $\mathbf{3}^{rd}$ most targeted country in terms of Cyber Attack [17], which in case of SSH protocol is deducible from this data portrayed Figure 5.3.

   c) In the year **2016** India was at $\mathbf{5}^{th}$ and there is increase in rate of attacks on Indian server after the period of Digitization [18].

   d) According to IP to Geo Location Mapping available at internet, the countries that originated malicious traffic to attack these cloud systems were 50 in count. Around 50% of the traffic for HoneySSH was from Vietnam. Figure 5.4 depicts the attacks distribution from all those countries.

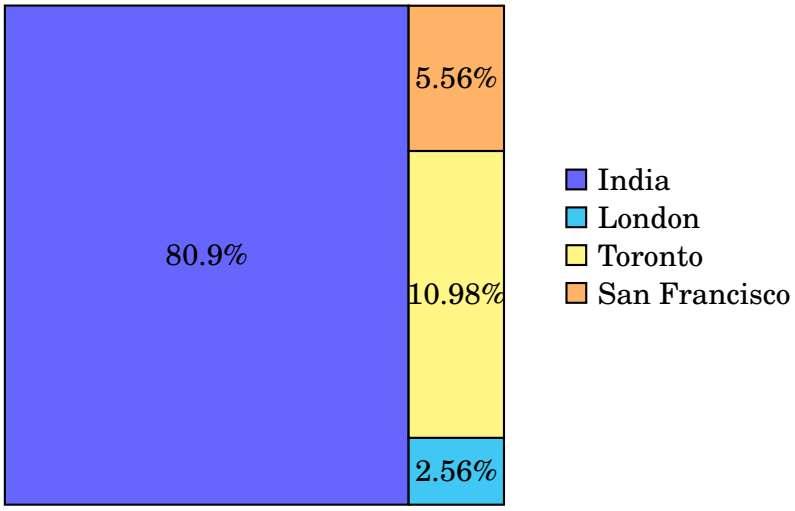

Figure 5.3: The count of attack captures at different location for same duration.

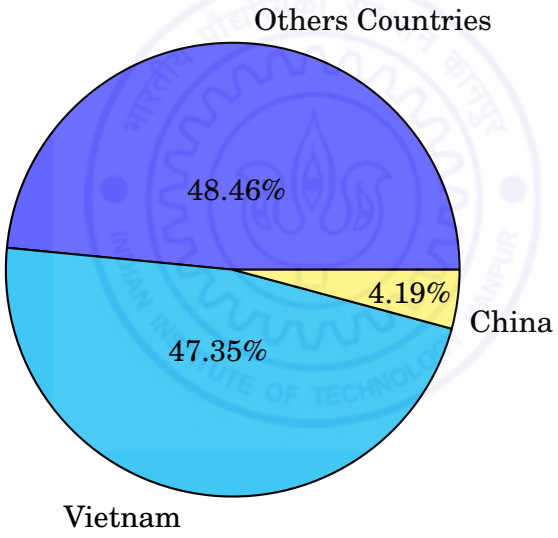

Figure 5.4: Attack count distribution across various countries for SSH protocol across All machines at different locations

e) According to the survey [19] most of the malicious traffic for this specific Honeypot originated from Vietnam & China [19]. The statistics of this was confirmed with the distribution given in Figure 5.4.

4. Amongst the attacks captured by all the HoneySSH deployments across different geographical locations, few IP address were common amongst them. The list of those IPs has been shown in Figure 5.5.

5. Different HoneySSH deployment scenarios:

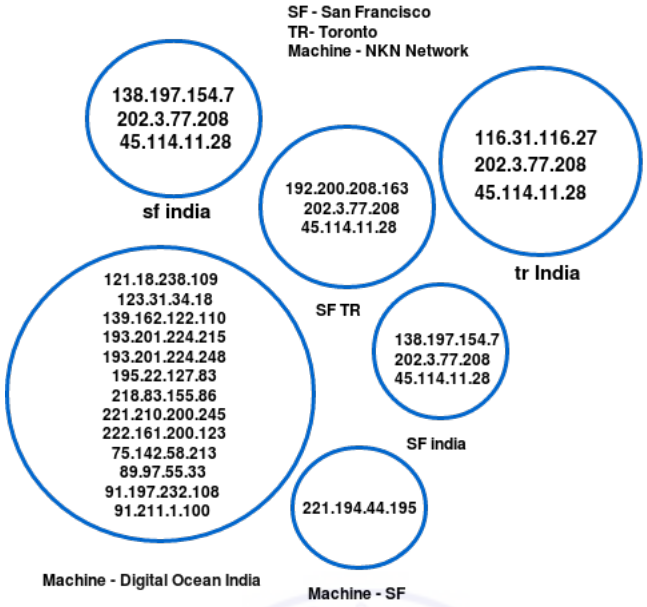

Figure 5.5: Common IPs amongst all deploys

a) **With 'admin' user & Unrestricted Firewall** : In this case, the HoneySSH was deployed with **admin** as a privileged user with no strict firewall policies. This scenarios invited good number of attacks and there were cases when honeypot was abused to do some harmful stuff to other servers. Most of the attacking scripts fetched in this case were similar.Total number of scripts that were fetched was around **6** unique scripts in the time span of 30+ days.

b) **With 'root' user & Restricted Firewall** : In this case, the outbound traffic of the firewall was restricted. It drops all the packets while creating a new connection from inside. Although the login count was quite high but no attacker downloaded any script.

c) **With 'root' user & Unrestricted firewall** : This scenario was deployed to capture scripts for the attacks called PDoS. This configuration of Honeypot invited the majority of the attacks. Made the attackers to download their scripts without any restriction. There was also increase in login attempts [6] . This scenario fetched around **91** unique scripts in span of just 10 days at three different locations. Quite a lot times the machine was compromised [7] in this case.

6. **Honeypot Recreation Timings** : When HoneySSH allows `root` user with unrestricted Firewall, there were several incidences of honeypot abuses in a day. Honeypot was recreated after 24 hours then the number of abuses were quite high and machine has to be brought down multiple times. But decreasing the time to

[6] Indicates a link between blackhats
[7] When the machine was use to abuse other machine to network

recreate the honeypot effectively changed the count of number of times the machine was used to abuse other machines. The statistics have been described in Figure 5.6 [8].

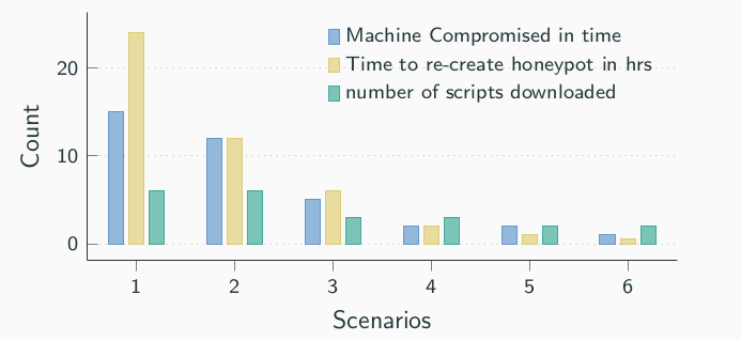

Figure 5.6: Statistics of time to recreate Honeypot v/s the Number of times the attacker downloaded scripts v/s Number of times the machine was compromised.

Careful observation of the log after the several abusing of Honeypots allowed to deduce the following observation out of it :

a) Attacker first scan the machine to check if it has any port open.

b) After finding the open port for ssh, he try to do brute forcing of ssh password for user 'root'.

c) After successfully brute forcing the system he keep the user name and password in the database to attack later.

d) Next login is usually after some time in which he downloads a malicious script or binary then again leaves the system.

e) In most of the cases the attacker uses the honeypot to attack other after his next attempt of login.

If after fetching the file, system is recreated then attack will be disguised about the honeypot and there will also be fewer cases of Honeypot abuses and the main purpose of Honeypot i.e to fetch the attacker scripts and executables will be fulfilled. This also allows to learn the parameter that in how much time the system need to be recreated after some malicious login. Exponential average can be a good technique to learn this parameter.

5.1.1 Analysis of files captured via HoneySSH

The total count of the malicious binary sample captured by HoneySSH are **88**. The sample are collected when the honeypot configuration was 'root' as username and password. The Table 5.4 clusters the binaries captured by the honeypot.

[8] Sudden increase in the network traffic of the Honeypot is counted as a Honeypot compromise

Table 5.4: Sample of Binary Files collected by HoneySSH

Nature of malicious file	No of malicious files
Linux/XorDDos	20
Linux/Bandwidth scripts	13
Linux/Dos	13
Linux/Full Dos Kit	13
Linux/Mirai	3
Linux/backdoor	2
Linux/ssh brute-force	2
Windows/Trojan-Ramgo.A	1
Windows/Trojan-Dridex	1
Classified as Data Files successfully	26

1. **XOR DDoS** : It is a trojan for Linux systems, that hijacks the system ad then uses it to DoS other Linux system. This creates a C&C server to control all the trojan that are spread across.

2. **Mirai** : It is a Malware that create a bot and exploits the device that runs the old version of Linux system. Creates a botnet that can be used to attack large scale networks.

3. **Scripts to Analyse Bandwidths** : Several scripts were captured that identifies the network bandwidth. This is commonly used by the attacker to first identify the capability of the network to attack other large scale infrastructures.

4. Two attackers foolishly added windows binaries to the Linux SSH system.

5.2 Attacks On HoneyFTP

In total HoneyFTP captured **460** unique IP address. The following information have been extracted from the attacks performed by these IPs.

1. The attack/access pattern specified in Figure 5.7 was observed during the deployement of HoneyFTP on the machine in NKN network. On specific days there were evidences of sudden increase in attacks/access in the HoneyFTP.

2. The peak indicates the evidence of **'FTP Bounce Attack'** attacks. In this type of attack, attacker tries to scan the ports of other host or system using FTP **PORT** command. This was the exploit that was already present in FTP protocol. In the initial configuration of HoneyFTP, this type of attack was not allowed and the `PORT` packets were refused by the sever. In the next version, `PORT` command was allowed to scan the internal network only. The internal network was intentionally integrated with a docker machine that allowed brute attacks on itself. Attackers

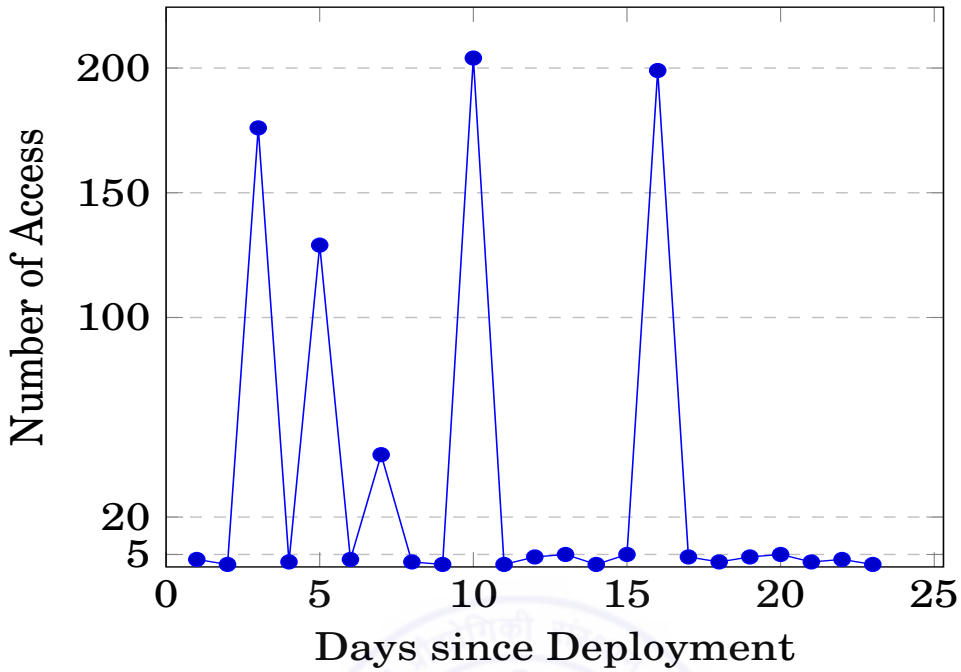

Figure 5.7: Per day trend of attack in HoneyFTP

performed brute forcing on port **22 and port 80**, scanning for some open ports on **ssh** and **http** protocol.

3. Attackers scanned `10.0.0.0/8,172.17.0.0/16` and `192.168.0.0/16` for the port `80, 22`.

4. There are evidences in the logs that 20 IPs belonged to the same `/16` network and the time stamp of access made by them was quite observable. Which indicates that these IPs may be behind a VPN.

5. HoneyFTP faced attacks from **8** geographically distributed locations where the proper distribution of these attacks have been depicted in Figure 5.8.

6. There is no attack identified in the system in which attacker tries to put some malicious files in the server to exploit some vulnerability of FTP service.

7. There is a one evidence from the IP that tried to over flow the buffer consecutively 45 times, but the command that it enters was not identified. As proposed HoneyFTP, server written in python was not able to handle it.

5.3 Attacks on HoneyWEB

During first 600 hours uptime, HoneyWEB was running the copy of server of `cse.iitk.ac.in` and it was targeted by `568` unique IP addresses. The complete description of these attacks

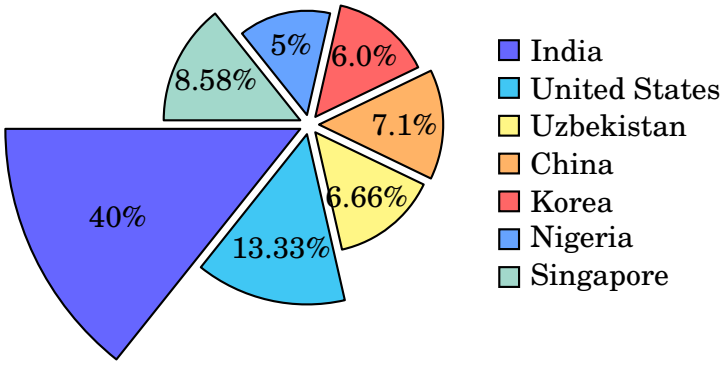

Figure 5.8: Country Wise Malicious Traffic Distribution on HoneyFTP

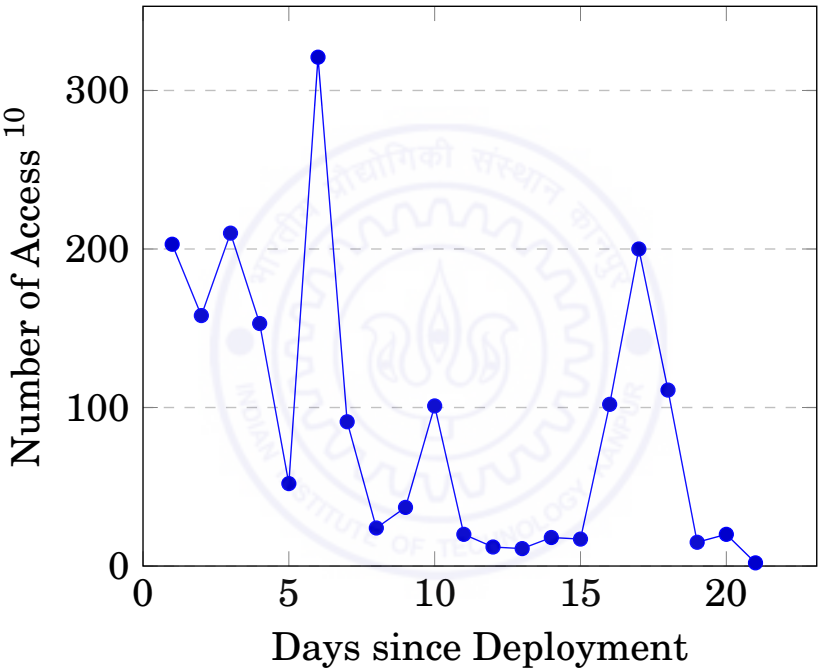

Figure 5.9: Attack trend during different days on HoneyWEb

is explained below:

1. The attack/access trend identified by the HoneyWEB is depicted in Figure5.9. On the first day of deployment the nubmer of attacks were quiet good, with increase in number of days the attacks count perday was also decreasing. The posssible reason could be probably the share of information amongst the attackers. During the first few days they might have tried various techniques to exploit the server and that exploitation keep on decreasing with respect to time. [9].

[9]These attack have been counted excluding the website download

2. On 23^{rd} day of the deployment, the attack/access count was less than 5. Then from consecutive days the count was either zero or very less.

3. HoneyWEB had been targeted by **38** different countries. Out of which, maximum number of attacks had been originated from China based servers. The distribution of attacking IPs targeting HoneyWEB is illustrated in Figure 5.10.

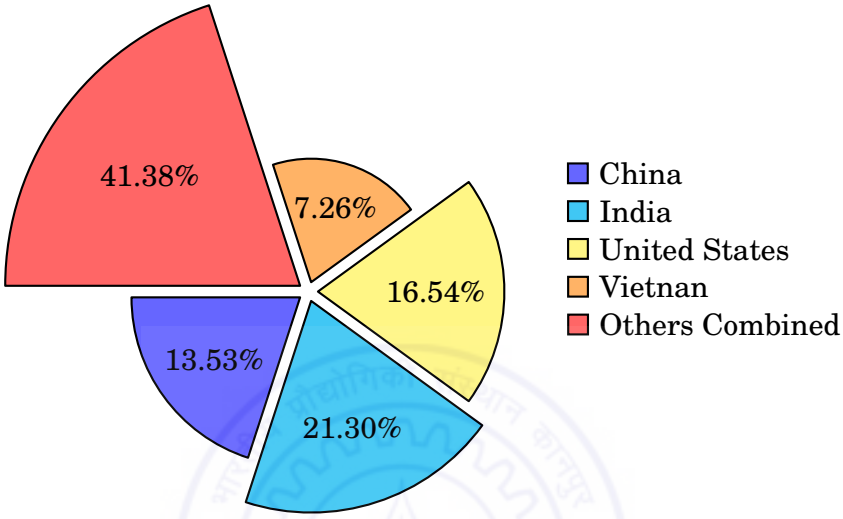

Figure 5.10: Attacks from the country to HoneyWEB for top 4 countries

4. There is an attack in which the IP belonged to **Microsoft CERT** organization from Virgina Boydton. This IP attacked the Honeypot with no user agent configuration. This IP did many notorious things:

   - It tries to scan all the versions of PHP and try to get access of the files like setup.php.
     `//phpMyAdmin-*.*.*.*-all-languages/scripts/setup.php`
   - It tries to exploit the **Linksys** vulnerability in Asus router by identifying **tmublock.cgi** file in the web server. This vulnerability allows the attacker to execute shell commands and to propagate worms in network.
   - It tries to identify if the Website is built using the wordpress framework. This framework initlialy allow everybody to access files that are setup.php and db.php. One of the attacks from this IP targeted towards this.
   - Scan for pages like
     `//w00tw00t.at.ISC.SANS.DFind:)`
     `//w00tw00t.at.blackhats.romanian.anti-sec:)`
     These are pages that are added by attacker to the compromised machine as a signature of compromised machine. The scan for this page indicates that attacking machine was compromised via this botnet and was scanning this IP if it is part of botnet or not.

5. There is an evidence that attacker was trying to run shell commands in url. She was assuming that website is built using `php` and uses `exec()` function of php to process arguments. Through this she was downloading the files. Total of 4 files were captured via this attack. These files were **hackqz**, `tfar`, **linux.arm** and **dlr.arm**. **dlr.arm** ind **tfar** are variants of Linux **Mirai**.

6. **6** IP address was trying to exploit **CVE 2012 1823** vulnerability. This vulnerability allows the php method to skip or turn off the owner matching of files. This was achieved by
`/cgi-bin/php?-d+allow_url_include=on+-d+safe_mode=off+-d+suhosin.simulation=on+-d+max_execution_time=0+-d+disable_functions=""+-d+open_basedir=none+-d+auto_prepend_ file=http://191.96.249.97/ok.txt+-d+cgi.force_redirect=0+-d+cgi. redirect_status_env=0+-n` Interestingly the IP address mentioned in the payload i.e `191.96.249.97` was same in all the six cases. This IP address belongs to Moscow Russia from where the attackers trying to download the attacking scripts.

7. There is an IP address which was scanning for file name **muieblackcat** in the server. Attacker may be trying to figure out whether the targeted machine was already compromised using Ukrainian origin bot net or not. Same IP address also tries to exploit php vulnerability in webserver.

8. There is an IP address tries to check whether the web site was built using `php` server then it called php `exec()` function to its query. The arguments of the query passed were :
```
$(nslookup on4kbAiL)
-1 OR 2+294-294-1=0+0+0+1 -
${10000490+10000460}
;print(md5(acunetix_wvs_security_test));
!(()&&!|*|*|
```

9. **3** This attack might be from **Jogree** compromised machines. This `Jogree` made access to around **336** files of the web server.

10. More than 30 IP addresses tried to exploit the vulnerability of struts framework first by identifying the version of **strut** framework and then accessing **login.action**, **test.action** & other ***.action** files.

CHAPTER 6

CONCLUSION & FUTURE WORK

This report presents the use of Honeysystem, how they are different from traditional research techniques and can be they useful for any organization. The report starts with the introduction about different Honeysystems. The implementation design and application of different honeypot systems have been covered in chapter 2, 3 and 4. It had been also outlined that how simple techniques like honeytokens can be used to identify device breaching using malware like gooligan. In that chapter, it has been demonstrated that how security of Android systems can be breached and how it can be used to steal the data from any user. The model of different honeypots proposed in this thesis are quite useful and their usefulness can be identified by the type of attacks they have captured. The entire analysis of attacks have been made in chapter 5 that suffices the effectiveness of honeypots.

6.1 Future Works

1. Data collected from these honeypots is presently analysed using manual techniques. Few parts of the analyses can be computed automatically.
2. The system has to be manually integradted with ELK stack. This need to be automated.
3. The HoneySSH still can be compromised and get alerts from the network admins, the approach for making it abuse, safe or free is still needs to be figured out.
4. The HoneyWEB model cannot automatically download the files in which the attacker tries some shell cmd that wgets the binary from some IP. This can be automaticlly saved to captured binaries.

APPENDIX A

APPENDIX A

The code for the applications disscussed in the report:

1. **HoneyFARM** - https://github.com/r0hi7/HoneySSH
2. **Augmented ssh client** - https://github.com/r0hi7/ssh4honeypot
3. **HoneyFTP** - https://github.com/nishitm/HoneyFTP
4. **HoneyWEB** - https://github.com/r0hi7/HoneyWEB
5. **ELK** - https://github.com/r0hi7/ELK
6. **Logstash conf file for all Honeypots** - https://github.com/r0hi7/logstash-conf
7. **Clientpot** - https://github.com/r0hi7/clientpot
8. **Malware captured** - https://github.com/r0hi7/Malware

BIBLIOGRAPHY

[1] Kyle York.
DYN statement on DDoS Attack.
http://dyn.com/blog/dyn-statement-on-10212016-ddos-attack/, 2017.

[2] Nicky Woolf.
Ddos attack that disrupted internet was largest of its kind in history.
The Gaurdian, 2016.

[3] Lance Sptizner.
Honeypots: Tracking Attacker.
Addison Wesley, 2002.

[4] Jamie Riden.
Server honeypots VS.client honeypots.
https://www.honeynet.org/node/158, 2008.

[5] Ari Juels and Ronald L Rivest.
Honeywords : Making Password-Cracking Detectable.
2013.

[6] Kevin D Fairbanks.
Forensic Framework for Honeypot Analysis.
2010.

[7] Armin Garcia.
Capture-HPC Client Honeypot: Honeyclient.
https://projects.honeynet.org/capture-hpc, 2006.

[8] Linux Man Pages, inotify.
`http://man7.org/linux/man-pages/man7/inotify.7.html`.

[9] Linux Man Pages, ld.so, ld-linux.so.
http://man7.org/linux/man-pages/man8/ld.so.8.html.

[10] Check Point Research Team.
More than 1 million google accounts breached by gooligan.
http://blog.checkpoint.com/2016/11/30/1-million-google-accounts-breached-gooligan/.

[11] Dirty Cow Linux Vulnerability.
https://dirtycow.ninja/.

[12] Sagie Dulce.
Dirty COW Vulnerability: Impact on Containers.
http://blog.aquasec.com/dirty-cow-vulnerability-impact-on-containers.

[13] Jeff Zacuto.
Adware or APT-SnapPea Downloader-An Android Malware that implements 12 different exploits.
http://blog.checkpoint.com/2015/07/10/adware-or-apt-snappea-downloader-an-android-

[14] Security Enhancements in Android 4.3 .
https://source.android.com/security/enhancements/enhancements43.

[15] AOSP source code V4.3 : run-as binary.
https://android.googlesource.com/platform/system/core/+/ac5c122/run-as/run-as.c.

[16] FTP bounce attack.
https://en.wikipedia.org/wiki/FTP_bounce_attack.

[17] Top 6 Most Targeted Countries For Cyber Attacks.
https://themerkle.com/top-6-most-targeted-countries-for-cyber-attacks/.

[18] India ranks among the top 5 countries at risk for cyber attacks in 2016.
http://www.businessinsider.in/India-ranks-among-the-top-5-countries-at-risk-for-cy articleshow/52364243.cms.

[19] Top 5 Countries Where Cyber Attacks Originate.
https://securitytoday.com/Articles/2017/03/03/Top-5-Countries-Where-Cyber-Attacks-Originate.aspx.