UNIVERSITI
TEKNOLOGI
PETRONAS

Honeypot setup for education

By

Muhd Harith Fathillah Bin Shahabudin (2714)

Final Dissertation in partial fulfillment of
the requirement for the
Bachelor of Technology (Hons)
(Information System)

University Technology of Petronas
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

t
QA
76.9
.A25
M952

2005

1. Computer security
2. Computer networks -- Security measures
3. IT/IS -- Thesis.

CERTIFICATION OF APPROVAL

Honeypot setup for education

By

Muhd Harith Fathillah Bin Shahabudin(2714)

A final dissertation submitted to the
Information Technology Programme
Universiti Teknologi PETRONAS
in partial fulfillment of the requirements for the
BACHELOR OF TECHNOLOGY (HONS)
(INFORMATION TECHNOLOGY)

Approved by,

____

(Mr. Low Tan Jung)

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK
JANUARY 2005

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

[illegible]

MUHD HARITH FATHILLAH BIN SHAHABUDIN

Abstract

"Honeypot" is just a computer system or a network segment, loaded with servers and devices and data. It may be protected with a firewall, although you want the attackers to have some access. There may be some monitoring capability, done carefully so that the monitoring is not evident to the attacker. The reason to setup honeypot varies from avoid tampering to tracking and capturing the attacker. The purpose for this project is to set up an education tool for student to learn and know how honeypot works. Because of the increasing rate of cyber crime, it is hope that with this, it can encourage the student and the public generally to be more aware about the matter concerning information technology security. The Honeynet Project made up of a small group of security professionals dedicated to learning the tools and tactics of the black-hat community and sharing those lessons learned with the security community. Their contribution varies from how to setup a honeypot to how the attacker behaves and what triggers them is shown in this paper and research. To design this particular honeypot, I've use the waterfall model methodology as a guideline to build and design it. Honeypot are interesting research topic and open up a new field of technology and creative thinking.

Acknowledgement

I would like to acknowledge several people who contribute their effort and time to help me in completing this final year project.

Firstly, I thank God for without His guidance and wisdom, I'll be utterly lost. Thank you for giving me the strength and knowledge needed to finish this task.

I am greatly indebted to the project supervisor, Mr. Low Tan Jung, who had continuously monitored my progress throughout the duration of the project. His constructive comments, advice and guidance led to the final outcome of this project.

To my very family and friends who have given me the strength to finish this project. Your love and laughter will always be in my heart. To my IT and IS colleagues who help me out through this whole process. I cherish you all very deeply.

To any of the person(s) not mentioned here, I would like to say "thank you" for your support until the completion of this project.

TABLE OF CONTENT

LIST OF FIGURES

ABBREVIATION

TCP : Transmission Control Protocol

IP : Internet Protocol

UDP : User Datagram Protocol

IRC : Internet Relay Chat

IDS : Intrusion Detection System

DNS : Domain Name Server

RPC : Remote Procedure Call

IT : Information Technology

ICT : Information Communication Technology

SYN-FIN : Synchronize Finish

PC : Personal Computer

LKM : Loadable Kernel Module

MAC: Media Access Control

LAN : Local Area Network

MTU : Multicast Transmission Unit

Chapter 1

Introduction

Background of study

How do you catch a mouse? You set a trap with bait (food the mouse find attractive) and catch the mouse after it is lured into the trap. This metaphor use to describe the honeypot setup system.

In this ever changing information technology era, cyber crime has become an increasing problematic issue. Because of this awareness, the honeypot method was introduced. Honeypot has really no special features. It is just a computer system or a network segment, loaded with servers devices and data. It may be protected with a firewall, although you want the attackers to have some access. There may be some monitoring capability, done carefully so that the monitoring is not evident to the attacker. You put honeypot for several reasons:-

- To watch what the attackers do, in order to learn about new attacks (so that you can strengthen your defense against these new attacks)
- To lure an attacker to a place in which you may be able to learn enough to identify and stop the attacker.
- To provide an attractive nut diversionary playground, hoping that the attackers will leave your real system alone.

Problem Statement

As said earlier, cyber crime rates have increased drastically during this past decade. It seems that every time developer or researcher find a solution or a way to prevent this from happening, hackers or sometimes call 'blackhat' community come out with a way to counter this prevention method. Thus the battle continues. This situation has inflicted million of dollars of losses and resources to big company such as Microsoft. Security has been the main issues to prevent crackers (the terms use for individual who hack to cause damage) from doing illicit activities thus endangering the whole corporation

Honeypot is some of the alternative methods used to prevent this from happening. This honeypot that I intend to do is not to be set up as a security measure, but merely as an education tool for student. Therefore, no real hacking is necessary. The real problem is to make sure that the system deploy in a mock up network can work or show similar result as the real honeypot would do. For this to happen, a suitable network design must be developed. Currently, there are some network designs for honeypots. From these designs, I must choose a suitable and efficient design that can be implemented. The failure to choose and implement a suitable design may affect the outcome of the project.

Objectives and Scope of Study

This honeypot is intended to be implemented in the lab.

The objectives of this Honeypot setup are:-

- As an education tool of how the system works and how it is implemented
- To generate general awareness to the public concerning the danger of cyber crime and the need of a good security system
- If possible, to encourage and attract student more on the subjects concerning network security

The scope of study will only be in University Technology Petronas. The lack of resources in term of network devices in the lab, making me to find a suitable honeypot network design is a challenge. Thus, this remains the most important obstacle en route to success. Also considering that honeypot is still a new subject, to build one that actually work is a big challenge.

Chapter 2

Literature Review

Honeypot has no definite definition. Thus the only conclusion that I come by during my literature review is that honeypot is a computer systems or a network segment, loaded with servers and devices and data. I also learnt that a combination of many honeypots is called honeynet. There is an organization for discussing this purpose that is known as the honeynet project. The Honeynet Project is a small group of security professionals dedicated to learn the tools and tactics of the black-hat community and sharing those lessons learned with the security community.

In one of their research paper call "Know Your Enemy: Honeynets" (14th January 2002), they discuss about how honeypot's work. In the paper is stated that conceptually, Honeynets are a simple mechanism. We create a network similar to a fishbowl, where we can see everything that happens inside it. Similar to fish in a fishbowl, we can watch and monitor attackers in our network. Also just like a fishbowl, we can put almost anything we want in the network. This controlled network, becomes our Honeynet. The captured activity teaches us the tools, tactics, and motives of the blackhat community.

Traditionally, the greatest problem security professionals face in detecting and capturing blackhat activity is information overload. The challenge for most organizations is determining from vast amounts of information, what the production traffic and what is malicious activity. Tools and techniques such as Intrusion Detection Systems, host based forensics, or system log analysis attempt to solve this by using a database of known signatures or algorithms to determine what is production traffic and what is malicious activity. However, information overload, data pollution, unknown activity, false positives and false negatives can make analyzing and determining activity extremely difficult.

Like all honeypots, the Honeynet solves this problem of data overload through simplicity. A Honeynet is a network designed to be compromised, not to be used for production traffic. Any traffic entering or leaving the network is suspicious by definition. Any connection initiated from outside the Honeynet into the network is most likely some type of probe, attack, or other malicious activity. Any connection initiated from the Honeynet to an outside network indicates that a system was compromised. An attacker has initiated a connection from his newly hacked computer and is now going out to the Internet. This concept of no production traffic greatly simplifies the data capture and analysis.

There are two critical requirements that define every Honeynet, they are the Data Control and the Data Capture. If there is a failure in either requirement, then there is a failure within the Honeynet. Honeynets are extremely flexible tools, they can built and deployed a variety of different ways, as such almost no two Honeynets look the same. But they must all meet the requirements of Data Control and Data Capture. Data Control is what mitigates risk. It controls the attacker's activity by limiting what can happen inbound and outbound. The risk is that once an attacker compromises a system within the Honeynet, they can use that system to attack other non-Honeynet systems, such as organizations on the Internet. The attacker has to be controlled so they cannot do that. They can attack other systems within the Honeynet, but we have to protect non-Honeynet systems. Data Capture is what collecting all the activity that happens inbound, outbound, or within the Honeynet. This is how we learn, by capturing the attackers's activities. The trick to these requirements is meeting them without the attacker knowing. Our goal is to both control and capture all of the attacker's activity, without them realizing they are within a Honeynet.

There is a third requirement, Data Collection, but this is only for organizations that have multiple Honeynets in distributed environments. Many organizations will have only one single Honeynet, so all they need to do is both Control and Capture data. However, organizations that have multiple Honeynets logically or physically distributed around the world have to collect all of the captured data and store it in a central location. This way the captured data can be combined, exponentially increasing its value. The Data

Collection requirement provides the secure means of centrally collecting all of the captured information from distributed Honeynets"

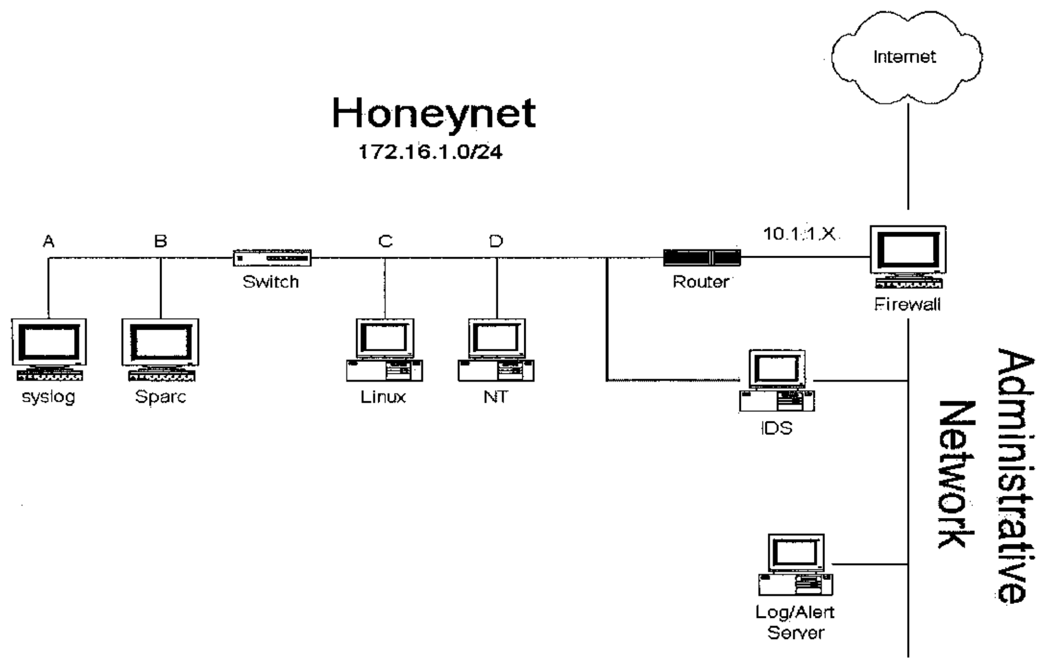


Figure 2.1

In figure 2.1 above, it shows an example of a network layout diagram of a honeypot setup. This network diagram is said to be use in a big organization where it involves complex configuration and expertise. The one that will be presented or create for this project will similarly look like the network diagram shown in figure 3.2.
The main difference in the both of the design is the level of complexity and resources involve. Figure 2.1 will show a working or real time networking environment. A hacker will usually see nothing suspicious of any kind when stumbling into this type of architecture. The honeypot that is implemented in this project is specifically use for an isolated network and is use for an education tool where the design is not purposely to attract hackers because there are no 'real' hacker involves.

In one of their paper they presented call “Know Your Enemy: Motives, the Motives and Psychology of the Black Hat Community, (22nd July 2001) it discusses on their findings and analysis of how hackers think and behave.

In the paper, it stated that,

"A honeynet is a network of various honeypots, designed to be compromised by the black-hat community. While some honeypots are used to divert the attention of attackers from legitimate systems, the purpose of a honeynet is to learn the tools and tactics of the black-hat community. Most of the information provided in this document has been sanitized. Specifically, user identities and passwords, credit card numbers, and most of the system names involved have all been changed. However, the actual technical tools and the chat sessions themselves have not been sanitized."

This paper also stated,

"What we have witnessed here are commonly used tools and tactics of the black-hat community. Our black-hat randomly scanned the Internet for a known vulnerability (in this case rpc.ttdbserv). Once identified, they quickly compromised the system and installed a rootkit using commonly scripted tools. Once they had control, they installed a bot, most likely to ensure they would maintain 'ops' on the IRC channels of their choice. What is uncommon are the two weeks of IRC chat sessions that their bot captured for us. In the next part of this paper, we discovered the motivations and psychology of the black-hat community, in their own words.
They also discovered that some of this individual involved in the black hat community has very little knowledge of network. Often you will see them attempting to figure out the most fundamental of Unix skills. And yet, they are still able to compromise or damage a large number of systems. This is not a threat that's need to be taken lightly.
This paper not only shows how the black hat community works, it also emphasized on their behaviour and intention that trigger their actions."

They have also made some quiet remarkable breakthrough in their research.

While researching the blackhat community, the Honeynet Project has been astonished to see just how active the blackhat community can be. The findings are scary. Below are some of the statistics we have identified from the eleven month period of data we collected. The purpose of these figures is to demonstrate the active behavior of the blackhat community. Keep in mind, these statistics represent a home network of little value that was neither advertised nor made any attempts to lure blackhats. Larger organizations that have great publicity or value most likely are probed and attacked in far greater numbers.

Post attack analysis:

- Between April and December 2000, seven default installations of Red Hat 6.2 servers were attacked within three days of connecting to the Internet. Based on this, we estimate the life expectancy of a default installation of Red Hat 6.2 server to be less then 72 hours. The last time we attempted to confirm this, the system was compromised in less than eight hours. The fastest time ever for a system to be compromised was 15 minutes. This means the system was scanned, probed, and exploited within 15 minutes of connecting to the Internet. Coincidentally, this was the first honeypot we ever setup, in March of 1999.
- A default Windows98 desktop was installed on October 31, 2000, with sharing enabled, the same configuration found in many homes and organizations. The honeypot was compromised in less than twenty four hours. In the following three days it was successfully compromised another four times. This makes a total of five successful attacks in less than four days.
- In May 2000, the first full month we archived Snort Intrusion Detection alerts, the Honeynet recorded Snort 157 alerts. In February 2001, the Honeynet recorded 1,398 Snort alerts, representing an increase of over 890%. This increase may be affected by modifications to the Snort IDS configuration file. However, we also see an increase of activity in the Firewall logs. In May 2000, the first full month we archived firewall alerts, the Honeynet firewall logged 103 unique scans (not counting NetBios). In February 2001, the Honeynet logged 206 unique scans (not counting NetBios). This represents an increase of 100%. These numbers indicate blackhat activity has continued to grow, most likely the result of more aggressive, automated scanning tools and their growing availability.
- In a thirty day period (20 Sep - 20 Oct, 2000), the Honeynet received 524 UNIQUE NetBios scans, averaging 17 unique NetBios scans every day.
- In the month of February, 2001, a total of 27 X86 exploits were launched against the Honeynet. X86 means these attacks were designed for systems using the Intel based architecture. Of these, 8 were launched against a Solaris Sparc system. These exploit attacks cannot work against the Sparc system, as the system

architecture is not compatible. This indicates that some blackhats are not bothering to confirm what operating system or what version of the service you are running. Some blackhats have streamlined their scanning process to merely look for a specific service. If they find the service, they launch the exploit without even first determining if the system is vulnerable, or even the correct system. This active approach allows blackhats to scan and exploit more systems in less time.

- From April 2000 through present, the most popular reconnaissance methods, besides general scanning, was DNS version query, followed by queries to RPC services.
- The most popular attack method was an overflow associated with rpc.statd for Intel based systems.
- The most popular scanning method detected was the SYN-FIN scan to search the entire IP range for specific ports (often in sequential order). This reflects the tactic of focusing on a single vulnerability, and scanning as many systems as possible for the vulnerability. Many blackhats only use a single tool or exploit that they know how to use, or is the most effective.

15th May 2005 (New Straits Times), Nurris Ishak title "In Malaysia, it's a hackers heaven", the writer talks about network security system and why most system are vulnerable. Some of the hackers know more about network security than any average person who are network literate. As far as they are concerned, the Internet security of most organisations in Malaysia is far from secure. The system administrators of the organisations or companies should pay attention to the latest in information technology.

It also stated that

"It is easy to hack a website and to find weaknesses in the system. Even a primary school kid can do it.
"If a hacker is malicious, he can do a lot of damage to a system or to individuals.
Hackers can re-create a bogus website that looks exactly like the real one and no one can tell the difference.
"This is not such good news if you are a banking website, for example."

It does not even take a computer genius to hack, according to hackers. You can find hacking software on the Internet, and downloading the programmes and using them maliciously is just a click away. A hacker can download a port scanner, which looks for an open door in a system. Usually, any system which can be accessed by the public has some extra ports open so that the public can have access to it. All a hacker has to do is to find the open port and enter whatever commands that they can create themselves, and they are in the system."

Most hackers said they do it just for the fun of it, but there is always the few who do it for malicious reasons or profit. Even individuals can be a hack victim, and anyone connected to the victim can be subjected to hack attacks. You can be a victim as easily as clicking on a button to a website or opening e-mail.

Users should always be careful in downloading attachments, as there may be a programme that is hidden in it that would allow a hacker to have access to their computer system. Hacking incidents are preventable, but this would depend on the person's awareness on the ways to protect themselves from it.

Online banking users, for example, are advised to type the website address themselves, instead of clicking on a link. This is because links can be quite deceiving.
A hacker can create a website which may look exactly like an online banking system website, and the average user wouldn't be able to know the difference.

Any website address that begins with http:// is an insecure website, which means that whatever information disclosed by the user would be open to whoever happens to be hacking in. A secure website address would appear as https://.

Putting up a firewall may well be the best protection but it's only a matter of time before a hacker can find the hole in the system. The best security is the Internet administrator himself. If he keeps up with the technology, he would know the weakness in the system and he would be able to patch it up.

IT education also plays an important part to prevent oneself from being a victim.
In this day and age, one should always be aware of the current technology. Education is the key to safety.

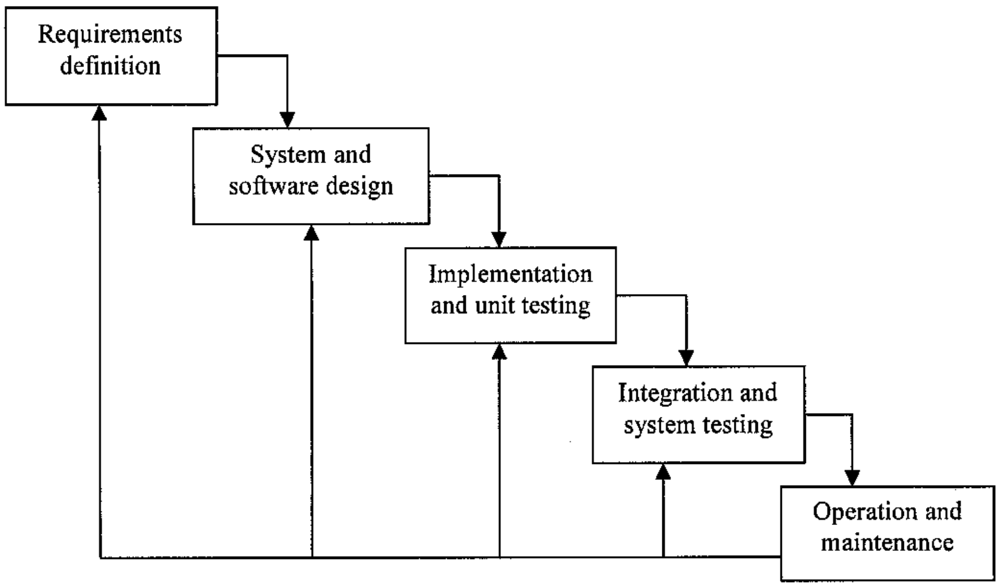
According to the National ICT and Emergency Response Centre (NISER), as of the first quarter of 2005, there were 300 reported hack incidents which comprised of intrusions, hack threats and denial of service. A spokesman for the centre confirmed that there has been a 100 per cent increase in hack incidents, compared to the fourth quarter of 2004.

We would say any financial transactions over the Internet are not 100 per cent secure without dual (two-factor) authentication from client and server side, which is not being implemented now. A user should always be wary of websites which requires them to disclose personal information over the Internet. There is always a potential threat out there.A user or an organization can protect their machines or systems from being hacked by applying several methods but it is a constant headache for corporations

Chapter 3

Methodology/Project Work

Because what the first intention is to create is a simple network model, thus there is no certain guideline on how to build or a platform to build a network, thus, it was decided to use a software model diagram to represent the project work flow. For this project, the waterfall model was use as the methodology to setup the honeypot. It consists of 5 steps. It is called as waterfall model because of the cascade from one phase to another phase. Each step is dependent with one another.



Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Figure 3.1

The waterfall model consists of:-

- Requirement analysis and definition:

  The require device network and the software that will be used for this honeypot will be determine and configure properly. The target result is determined here.

- System and software design:

  What kind of network design that this particular honeypot will use. For this, it was decided to choose a similar design to that of figure 3.2 .

- Implementation and unit testing:

  Each device and software that is already configured is then tested one by one. This is to test if network is working properly.

- Integration and system testing

  The individuals device network component and software are integrated and then tested wholly. If possible, this will be the part where some hacking occurs. After all, hacking is essential in this system to show if the system is working or not.

- Operation and maintenance

  After all the device and software are working properly, the honeypot must be maintained casually. Because this is an education tool, it should not be hard to maintain. The purpose of this project is to show how the system works.

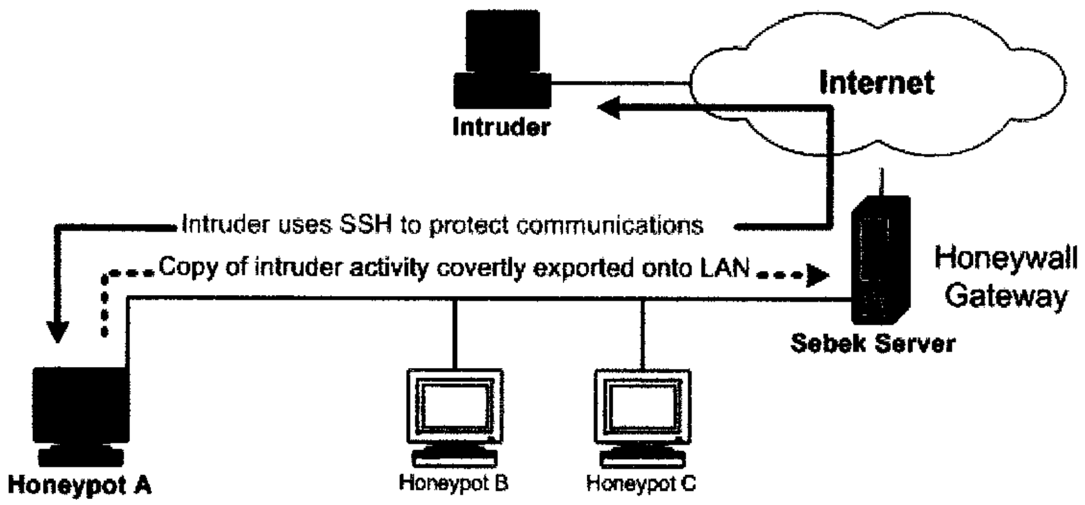

Internet
Intruder
Intruder uses SSH to protect communications
Copy of intruder activity covertly exported onto LAN
Honeywall Gateway
Sebek Server
Honeypot A
Honeypot B
Honeypot C

Figure3.2

Figure 3.2 above shows the design of the honeypot intended for this project. The network model that was decided to build are not 100 percent copy of that from figure 3.2 but still carry the same or similar characteristics of it. The heart of the honeypot comes from the Sebek system. Sebek have 2 components, there are; Sebek Client and Sebek server. Honeypot A consists of a Sebek file which is very vital in a honeypot setup. This Sebek file must be configured at the server and the client as well. Sebek will be discuss and explain later in the result and discussion chapter. For the operating system, it is decided that we use Fedora Core. It is basically a Linux base operating system and because of the Unix nature involve in this project, it will fit perfectly with the requirement needed. One more factor that was vital in considering implementing Fedora Core is that because it is said to be more stable and effective when it comes to networking area.

As mention earlier, the Sebek must be setup according to the specification and instruction given. Because this topic is still very new, many Sebek version are sometimes not stable and therefore, to get the require result is quiet a problem. One important factor that is vital for the Sebek setup is the kernel version of the operating system. Most Sebek installer for both client and server must always fit the kernel operating system requirement. For this project, the kernel version that I used is 2.1.7. The setbackl of this system is that it does not allow a higher version of the kernel in order to install it, thus, we must be very specific and concise over this matter.

As shown in figure 3.2, the first step is to setup an isolated network. To do that, four PC's were required located in the data communication lab. One will act as a server for the network, 2 will act as the client for the network. The 4th PC was initially to be setup as an external network but for the time being and because of the difficulties setup of the honeypot, this pc is use for a spare tool for future use.

In the server side, Sebek server version 2.1.7 was installed. This server is equipt with Fedora Core 2 operating system with a kernel version 2.1. As mentioned earlier, because of the instability of the system, there still exists some error. It has been recently discovered that Sebek doesn't seem to be unstable on some AMD systems. According to the team that develops this installation, the problem can be triggered by running tcpdump on a Sebek host. The kernel panics with the af_packet code trying to call function using a bad function pointer. They are still investigating the cause of this problem. Until this report is written, this is the latest version of Sebek server setup posted in the honeynet.org; the main website for honeypot research in the world. We will discuss this in the result and discussion method.

As for the client side, Sebek client version 2.1.7 was installed. This Sebek client is for Linux operating system that supports specifically 2.4 kernel version. As mention earlier, there are 2 PC connected in the network excluding the server. The positive thing about honeypots is that it gives the user freedom to set it anywhere you like. Bear in mind there are still some factors to consider first if you want to setup it in the real world. For example, in figure 3.2, both of the PC's have their own honeypot. Subsequently, they have their own Sebek system in it. To build a successful honeypot, the attacker must be able to penetrate and possible do malicious activities in the particular PC, without knowing that they are actually being monitor by the Sebek client itself. If you setup a honeypot where there is no probability of a hacker to attack, then the honeypot itself is a failure. Thus, determining the psychology and the intention of a hacker is also vital in the success of a honeypot. One of the objectives of honeypot itself is for this purpose only. With the data capture, the user can then make new counter measure and security to avoid their real system or PCs being compromise from attack.

For the second PC in the network, a Window background was choose for its operating system to install the Sebek client. Microsoft Windows XP was its operating system because the Sebek client that is available for Windows in Honeynet organization only supports this type of operating system. By using this, it was use to determined to see or observe the difference the Sebek have between 2 PC using different operating system. To my surprise, the PC that was initially choose for setup, after rebooting, encounter some problem whereby, it takes quiet some time just for it to reach the Window interface. The a strange message occur stating that the PC is encountering some low virtual memory thus making the session from rebooting, waiting for the windows interface and running the first application takes about half an hour to be process. The conclusion came to this by the fact that before installing Sebek client, it was running perfectly okay and smooth. Although it already has some programs and software installed, it was never occurred that the system will encounter this type of problem. Because of the nature of the Sebek itself is sometimes unstable, plus the factor that the Sebek will load itself before the operating system starts for Windows version, it was somehow evident that during this loading time and the transition from the Sebek program to Windows, some unexpected bug happen. For example, your computer might act strangely although all the possibilities of this problem have already been check. These act may varies from legging when running a software to fail to execute simple application

Chapter 4

Result and discussion

The honeypot that is implemented consist of 2 vital parts. First is the Sebek server part and the second is the Sebek client part. Sebek is a data capture tool. As with all data capture tools, the goal is to capture data that will allow us to accurately recreate the events on a honeypot. We want to determine information such as when an intruder broke in, how they did it, and what they did after gaining access. This information can, potentially, tell us who the intruder is, what their motivations are, and who they may be working with

Sebek server and Sebek Client

Sebek has two components: a client and server. The client captures data off of a honeypot and exports it to the network they are collected by the server (refer Figure 3.2).

The server collects the data from one of two possible sources: the first is a live packet capture from the network, the second is a packet capture archived stored as a tcpdump formatted file. Once the data is collected it is either uploaded into a relational database or the keystroke logs are immediately extracted. The communications used by Sebek are UDP based and as such are connectionless and unreliable.

The client module is installed on the honeypot. The attacker's activity captured by the honeypot is dumped to the wire (hidden to the attacker) and passively collected by the Honeywall Gateway. The client resides entirely in kernel space on the honeypot and, in the case of the Linux version, is implemented as a Loadable kernel Module (LKM). The client can record all data that a user accessed via the read() system call. This data is then exported to the server over the net in a manner that is difficult to detect from the honeypot running Sebek. The server then gathers the data from all of the honeypots sending data. Because there is a standard platform independent log format the server can collect from any honeypot independent of Operating System type. Let us now take a closer look at how the client actually captures the data.

To determine what an intruder did after gaining access, we often want data that provides the intruder's keystrokes and the impact of the attack. When encryption is not used, it is possible to monitor the keystrokes of an intruder by capturing the network activity off the wire and then using a tool like ethereal to reassemble the TCP flow and examine the contents of the session. This technique yields not only what the intruder typed but also what the user saw as output. Steam reassembly techniques provide a nearly ideal method to capture the actions of an intruder when the session is not encrypted. When the session is encrypted, stream reassembly yields the encrypted contents of the session. To be of use these must be decrypted. This route has proven quite difficult for many. Rather than trying to break the encryption of a session, others have looked for a way to circumvent encryption.

Information that is encrypted must at some point decrypted for it to be of any use. The process of circumvention involves capturing the data post decryption. The idea is to let the standard mechanisms do the decryption work, and then gain access to this unprotected data. The first attempts to circumvent such encryption took the form of trojaned binaries. When an intruder broke into a honeypot, he or she would then log into the compromised host using encrypted facilities such as SSH. As they typed on the command line, a trojaned shell binary would record their actions. To counter the threat posed by trojaned binaries, intruders started to install their own binaries. It became apparent that the most robust capture method involved accessing the data from within the Operating System's kernel. When capturing data from within the kernel, the intruder can use any binary they wish, and we are still able to record their actions. Furthermore, because user space and kernel space are divided, there is ample opportunity to improve the subtlety of the technique, by hiding our actions from all users including root.

The first versions of Sebek were designed to collect keystroke data from directly within the kernel. These early versions were the equivalent of a souped up Adore Rootkit that used a trojaned sys_read call to capture keystrokes. This system logged keystrokes to a hidden file and exported them over the network in a manner to make them look like other UDP traffic, such as NetBIOS. This system allows users to monitor the keystrokes of an intruder, but it was complex, easy to detect through the use of a packet sniffers and it had a limited throughput. This last issue made it difficult to record data other than keystrokes.

The next and current iteration of Sebek, version 2, was designed not only to record keystrokes but all sys_read data. By collecting all data, we expanded the monitoring capability to all activity on the honeypot including, but not limited to, keystrokes. If a file is copied to the honeypot, Sebek will see and record the file, producing an identical copy. If the intruder fires up an IRC or mail client, Sebek will see those messages. A secondary goal was to make Sebek harder to detect, we focused on switching from obfuscating the logging traffic, to completely hiding it from a Blackhat. Now when a Blackhat runs a sniffer to detect suspicious traffic he or she is unable to detect any Sebek traffic. Sebek is not just an alternative to TCP session reassemble, to be used only in the face of encryption. Sebek also provides the ability to monitor the internal workings of the honeypot in a glass-box manner, as compared to the previous black-box techniques. If an intruder wanted to install a piece of malware, and then log out, we can now track the local actions of the malware even if it does not access the network.

Client Data Capture:

Data capture is accomplished with the use of a kernel module. With this module we gain access to the kernel space of the honeypot. Using this access, we then capture all read() activity / data. Sebek does this by replacing the stock read() function in the System Call Table with a new one. The new function simply calls the old function, copies the contents into a packet buffer, adds a header, and sends the packet to the server. The act of replacing the stock function involves changing one function pointer in the System Call Table.

When a process calls the standard read() function in user space, a system call is made. This call maps to an index offset in the System Call Table array. Because Sebek modified the function pointer at the read index to point to its own implementation, the execution switches into the kernel context and begins executing the new Sebek read call. At this point Sebek has complete view all data accessed with this system call. This same technique could be used for any System Call that we may wish to monitor. Data that remains encrypted is of little use; to view the data or act on it in some way it must be decrypted. In the case of an SSH session the keystrokes are decrypted and send to the shell to have actions performed. This act typically involves a system call. By collecting data in kernel space, we can gain access to the data within the system call, after it has been decrypted but before it has been passed to the process that is about to use it. Thus we circumvent the encryption and capture the keystrokes, file transfers, Burneye passwords, etc.

Client Module Hiding:

To make the presence of the Sebek module less obvious we borrow a few techniques used in modern LKM based rootkits, such as Adore. Because Sebek is now entirely resident in kernel space, most of the rootkit techniques no longer apply, however, hiding the existence of the Sebek module is one example of direct technological benefit. To hide the Sebek module we install a second module, the cleaner. This module manipulates the linked list of installed modules in such a way that Sebek is removed from the list. This is not a completely robust method of hiding and techniques for detecting such hidden modules do exist. There are 2 side effects of this removal. First, users can no longer see that Sebek is installed. Second, once installed, users are unable to rmmod the Sebek module. This hiding ability can be disabled by setting the "Testing" variable to "1" in the sbk_install.sh install script.

Client Packet Export:

Once the Sebek client captures the data, it needs to send the data to the server without the intruder detecting that the host is sending this data. Although using the LAN network to send traffic to the server is not the most secure communication channel, it was decided that because of its ubiquity, it would be used to send data to the server. If Sebek were to simply send the data to the server over an UDP flow, an intruder could simply check for the presence of such traffic on the LAN to determine if Sebek was installed. Sebek does send data to the server using UDP, however, before it does this it modifies the kernel in a few ways to prevent users from seeing these packets. First it modifies the kernel such that system is unable to see Sebek Packets, not just the packets generated by the local host, but any appropriately configured Sebek Packet. Next, when Sebek transmits data onto the network, it ensures that the system cannot block the transmission or even count the packets transmitted. We will get into the details of packet hiding shortly.

If every honeypot on a LAN has Sebek installed, none of them can see any Sebek data, however the server has full access to this data. By deploying in this manner we have created a covert channel that allows the server gain access to the data captured by the client. For every read() request, Sebek generates one or more log packets. Each packet contains a bit of information about the context of the call made and the data that was accessed with the call. Each packet contains one Sebek record. The record contain fields that describe the Process that made the call, the time the call was made and the length of the data recorded as well as the data itself.

These packets are generated entirely within Sebek without using the TCP/IP stack to generate or send the packets. Because of this, the system is unable to see or block the packets. After each packet is built, it is sent directly to the device driver, bypassing the raw socket code path as wellas the packet filtering code path. Since packet sniffers are libpcap based and libpcap uses the raw socket interface to collect packets, sniffers installed on a host running Sebek are unable to see the Sebek packets generated. 2 Phrack issue 61 has an article on detecting hidden kernel modules in its Linenoise section. The article describes a brute force method for detecting hidden modules by looking for what appears to be the key module structure.

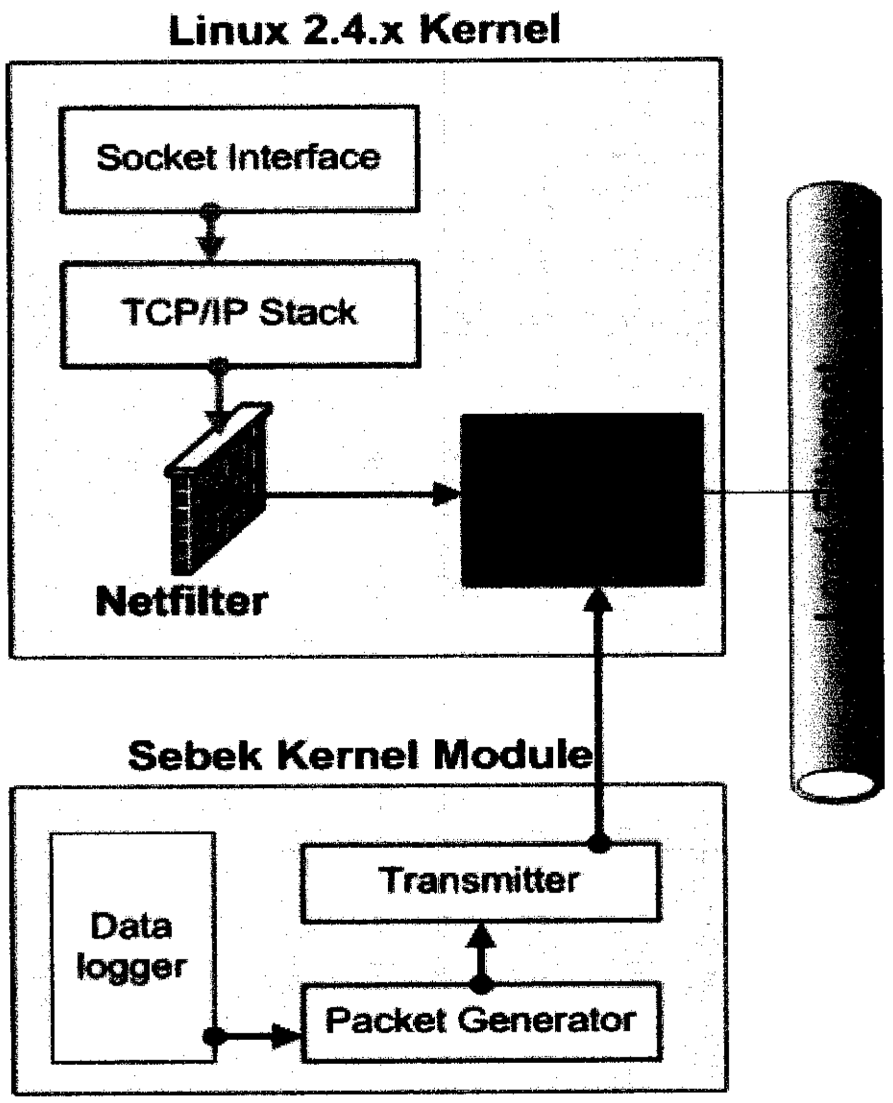


Figure 4.1: Conceptual representation of Sebek packet generation. Note how packets created by Sebek bypass the stack and go directly to the network device driver. This makes it far more difficult for attacker to detect Sebek activity.

Because Sebek generates its own packets and sends them directly to the device driver, there is no ability for a user to block the packets with IPTABLES or monitor them with a network sniffer. This prevents an intruder on a honeypot from detecting the presence of Sebek by examining the LAN traffic. A secondary problem that must be addressed is the need to keep honeypot A from detecting Sebek packets from honeypot B. The use of Ethernet switching does not solve this problem. Sebek is naturally impervious to ARP spoofing because it does not use ARP to obtain the destination MAC address that corresponds to the destination IP address, however there are a couple of situations where A would see B's Packets3. In these situations, an intruder would be able to detect the presence of Sebek packets on the LAN by running a sniffer on honeypot A and would see honeypot B's Sebek packets.

To solve this problem, Sebek installs its own implementation of the Raw Socket interface. This new version is programmed to silently ignore Sebek packets. Sebek packets are defined as those that have both a predetermined destination UDP port and the proper magic number set in the Sebek header. If these two values match what is expected, then we know this a packet to ignore. The implementation simply does nothing with Sebek packets; it drops them on the floor

Sebek Protocol Specification:

To ensure interoperability with all versions of Sebek, a common protocol has been defined. The communication channel between the client and the server is unidirectional, with packets originating from the client and destined to the server. This channel is UDP based and is connectionless and unreliable. Each packet contains one record. Records are variable length up to MTU, and have a fixed length header. Each record has a 48 byte header.

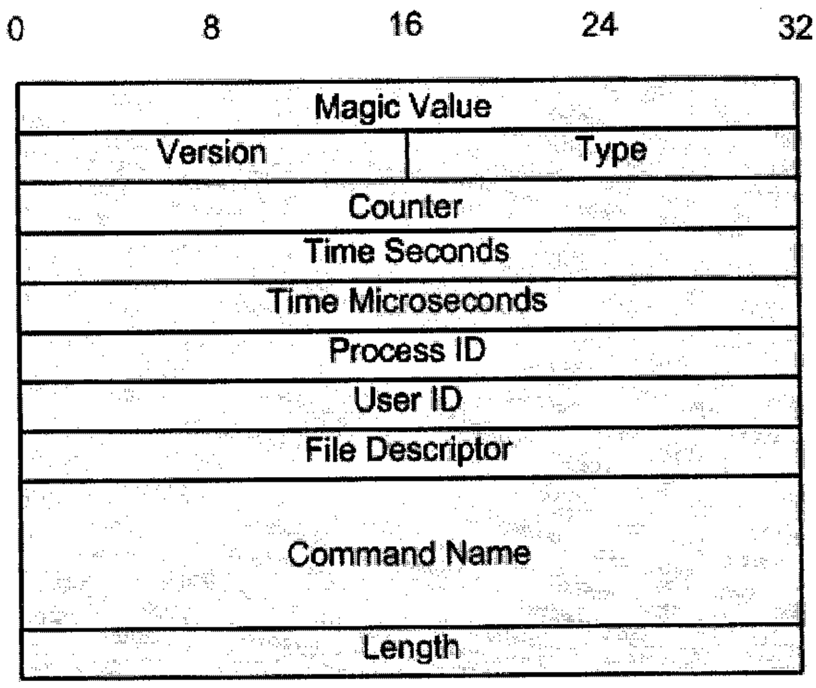


Figure 4.2
Sebek Record Header. This header comes after the IP/UDP header, and is followed bythe data representing the activity on the honeypot (attacker keystrokes, files, passwords,etc).

Risk Involve

For Sebek to be of use in a honeypot it must not be detected by an intruder. It is possible to detect Sebek, or any covert tool of this type using techniques common to rootkit detection. As a result, the decision to use this system must consider the increased potential for detection by an intruder. In the linux client, this increased risk is caused by the presence of kernel module support as well as the /dev/kmem feature. These facilities are very powerful and make the linux kernel quite flexible. Just as we used them to install Sebek, intruders can use them to detect and disable Sebek. Fortunately, by the time Sebek has been disabled, the code associated with the technique and a record of the disabling action has been sent to the collection server. Thus in the future one avenue to limit the risk of detection will be to have the server detect when Sebek has been attacked and then disable the client.

The current sebek file consists of some parts that is integrated together during the installation. The first part is to monitor the keystroke activity on the host running the sebek client from a command line prompt on the server.

```
[2003-07-23 20:03:45 10.0.0.13 6673 bash 500]whoami
[2003-07-23 20:03:48 10.0.0.13 6673 bash 500]who
[2003-07-23 20:03:50 10.0.0.13 6673 bash 500]su
[2003-07-23 20:03:57 10.0.0.13 6886 bash 0]cd /var/log
[2003-07-23 20:03:57 10.0.0.13 6886 bash 0]ls
[2003-07-23 20:04:01 10.0.0.13 6886 bash 0]mkdir ...
[2003-07-23 20:04:20 10.0.0.13 6886 bash 0]tcsh
[2003-07-23 20:04:20 10.0.0.13 6921 tcsh 0]0
[2003-07-23 20:04:20 10.0.0.13 6920 tcsh 0]vt
[2003-07-23 20:04:20 10.0.0.13 6920 tcsh 0]en
[2003-07-23 20:04:20 10.0.0.13 6920 tcsh 0]en
[2003-07-23 20:04:27 10.0.0.13 6920 tcsh 0]cd /tmp
[2003-07-23 20:04:28 10.0.0.13 6920 tcsh 0]ls
[2003-07-23 20:04:42 10.0.0.13 6920 tcsh 0]cd /usr/lib
[2003-07-23 20:04:42 10.0.0.13 6920 tcsh 0]ls
```

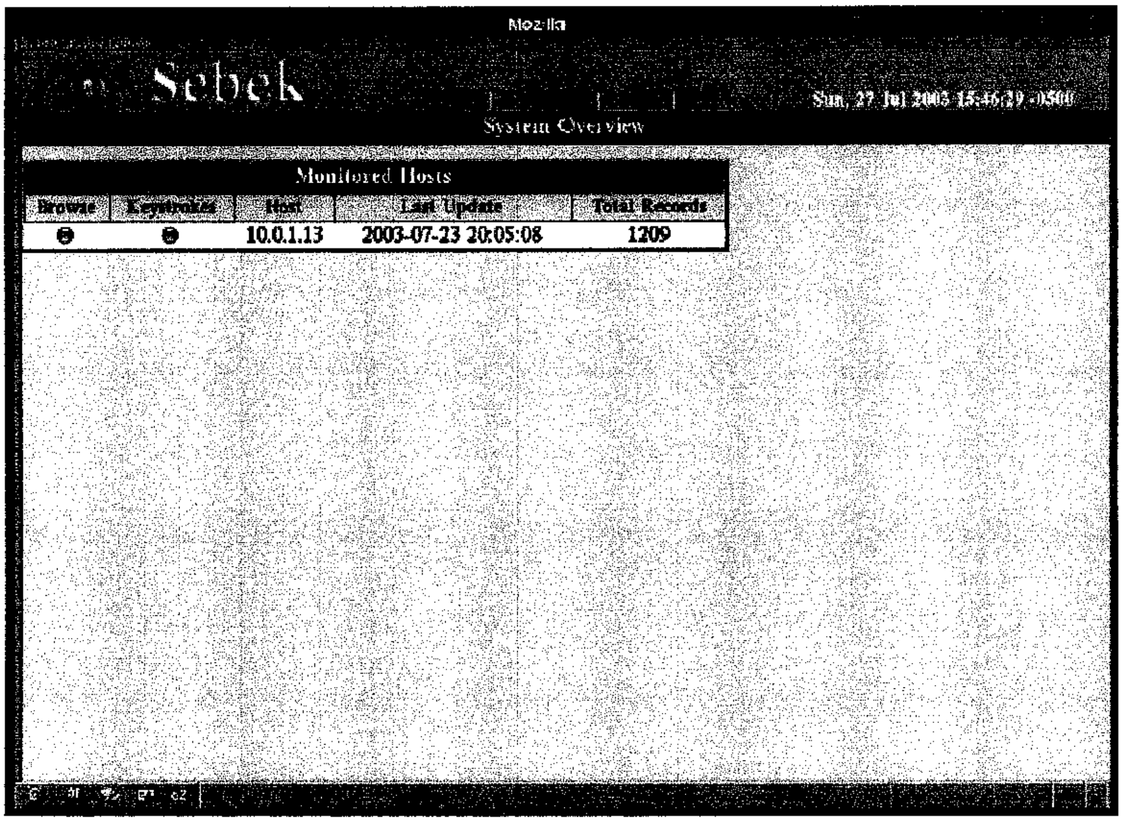
Figure 4.3

The expected result should look like those shown in figure 4.3. The output from the command prompt is similar to what users see when on a terminal. However, we will only see commands entered and not the output of those commands. Note that sometime a hacker infiltrates a system using command prompt line. This is very common especially in Unix command because of the vulnerability that exist in the system. Control characters are escaped when present. For example, when a Backspace is present, it is replaced with [BS].

Sebek now comes with a web based analysis interface. This interface provides users with the ability to monitor keystroke activity, search for specific activity, recover SCPed files and in general provides an improved data browsing capability. This interface is implemented with PHP and only examines the data contained in the database; it does not use data from other sources such as packet captures or syslogs. It is designed to support the workflow of forensic investigation; however it does require a fair degree of technical skill to understand. The intention was to make this a tool for Sebek data like ethereal is for packet captures. The interface has 3 primary options: viewing keystrokes, searching, and browsing.

- The Keystroke Summary view provides a summary of all keystroke activity.
- The Search view allows users the ability to query for certain information.
- The Browse view, or table view, provides a summary of all activity, including non keystroke activity.

First of all, it is not the intention that this web interface will be use for this project, rather just to show the expected and capabilities of these sebek file towards the honeypot setup.

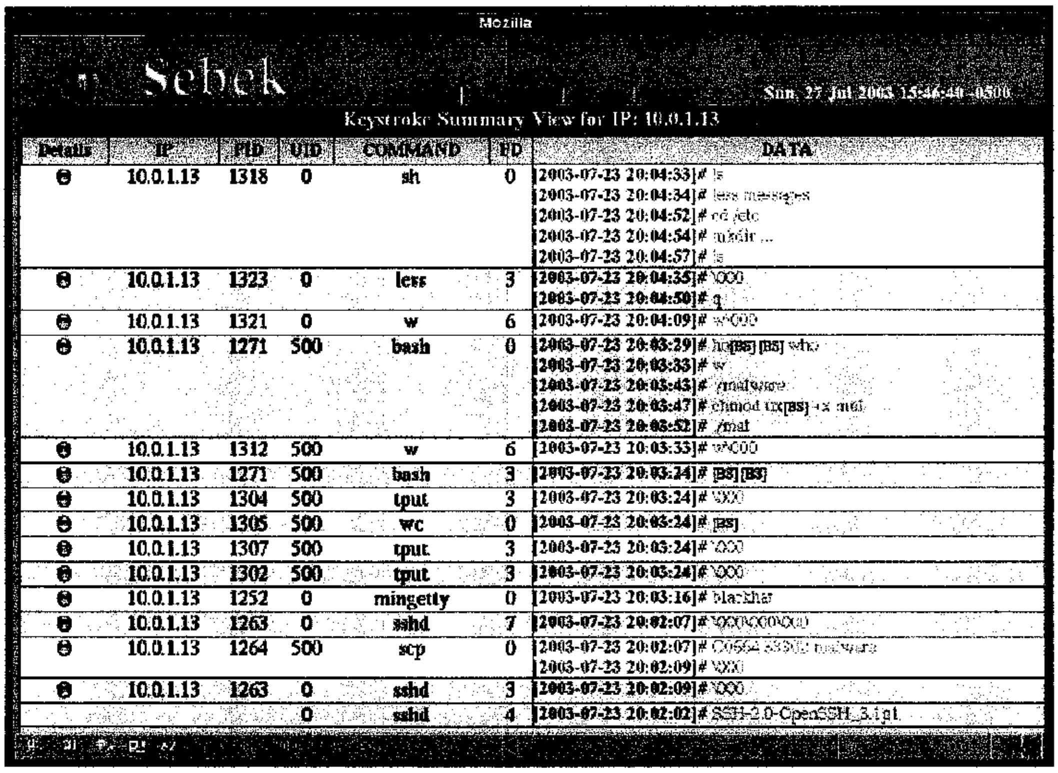


Monitored Hosts

Browse	Keystrokes	Host	Last Update	Total Records
●	●	10.0.1.13	2003-07-23 20:05:08	1209

Figure 4.4

Figure 4.4 shows an example of the system overview of the interface. It provides a list of all the hosts we are monitoring and the last time we saw any activity from the host. By clicking on the Keystrokes button we can get a summary of what most likely is humanly derived activity.



Sebek

Sun, 27 Jul 2003 15:46:40 -0500

Keystroke Summary View for IP: 10.0.1.13

Details	IP	PID	UID	COMMAND	FD	DATA
●	10.0.1.13	1318	0	sh	0	[2003-07-23 20:04:33]# ls [2003-07-23 20:04:34]# less messages [2003-07-23 20:04:52]# cd /etc [2003-07-23 20:04:54]# mkdir ... [2003-07-23 20:04:57]# ls
●	10.0.1.13	1323	0	less	3	[2003-07-23 20:04:35]# \000 [2003-07-23 20:04:50]# q
●	10.0.1.13	1321	0	w	6	[2003-07-23 20:04:09]# w\000
●	10.0.1.13	1271	500	bash	0	[2003-07-23 20:03:29]# ho[BS] [BS] who [2003-07-23 20:03:33]# w [2003-07-23 20:03:43]# ./malware [2003-07-23 20:03:47]# chmod u[BS] +x mal [2003-07-23 20:03:52]# ./mal
●	10.0.1.13	1312	500	w	6	[2003-07-23 20:03:33]# w\000
●	10.0.1.13	1271	500	bash	3	[2003-07-23 20:03:24]# [BS] [BS]
●	10.0.1.13	1304	500	tput	3	[2003-07-23 20:03:24]# \000
●	10.0.1.13	1305	500	wc	0	[2003-07-23 20:03:24]# [BS]
●	10.0.1.13	1307	500	tput	3	[2003-07-23 20:03:24]# \000
●	10.0.1.13	1302	500	tput	3	[2003-07-23 20:03:24]# \000
●	10.0.1.13	1252	0	mingetty	0	[2003-07-23 20:03:16]# blackhat
●	10.0.1.13	1263	0	sshd	7	[2003-07-23 20:02:07]# \000\000\000\000
●	10.0.1.13	1264	500	scp	0	[2003-07-23 20:02:07]# C0664 [illegible] malware [2003-07-23 20:02:09]# \000
●	10.0.1.13	1263	0	sshd	3	[2003-07-23 20:02:09]# \000
			0	sshd	4	[2003-07-23 20:02:02]# SSH-2.0-OpenSSH_3.1p1

Figure 4.5

The Keystroke Summary view in Figure 6 provides the last 5 or so lines of text from all sessions on a host. Sessions are nothing more than activity on a specific File Descriptor for a given process. The sessions are sorted by time of last activity, with most recent at the top. Within each summary the keystroke logs are sorted in the order they occur

Chapter 5

Conclusion and Recommendation

For conclusion, honeypot is still a new concept in terms of security measure. Honeypot subject itself is very interesting. It involves not only the technology aspect, but a more detail research on individual psychology and their behavior. The main objective of the project is to create an education tool for university student to show how honeypot works. The main component for this honeypot is the Sebek server and Sebek client. So it is vital or essential to explain this feature to understand the main concept of honeypot. The operating system for this particular network will consist of mainly Linux product that is Fedora Core operating system. Because it was intentionally built and test in a UNIX environment, choosing Linux as the operating system is a safe way. For that reason, I opted to choose Fedora Core 3 as the operating system. Because it support network and can be setup as a server. As for the recommendation, it is hope that this honeypot setup project will grow from an education tool, to a fully workable system for university security. It is reported that university has the most fragile net security. Through out my research and literature review, I've stumble upon some university where they setup their own honeynet and the research, analysis, and statistic that they collect are use widely for numerous reason.

Perhaps maybe, University Technology Petronas can be the pioneer for this field in this region by using this project as a stepping stone.

References

1. Edward Balas 22 July 2001, *Know Your Enemy: The motives and psychology of the black hat community*, retrieved on January 13,2005 <http://project.honeynet.org/article1.htm>

2. Edward Balas 14 January 2002, *Know Your Enemy: Honeynet What a Honeynet is, its value, how it works, and risk/issues involved* retrieved on February 15, 2005 <http://project.honeynet.org/article2.htm>

3. Pfleeger, C. and Pfleeger, S. " *Security in computing*" Third edition, retrieved on February 15, 2005

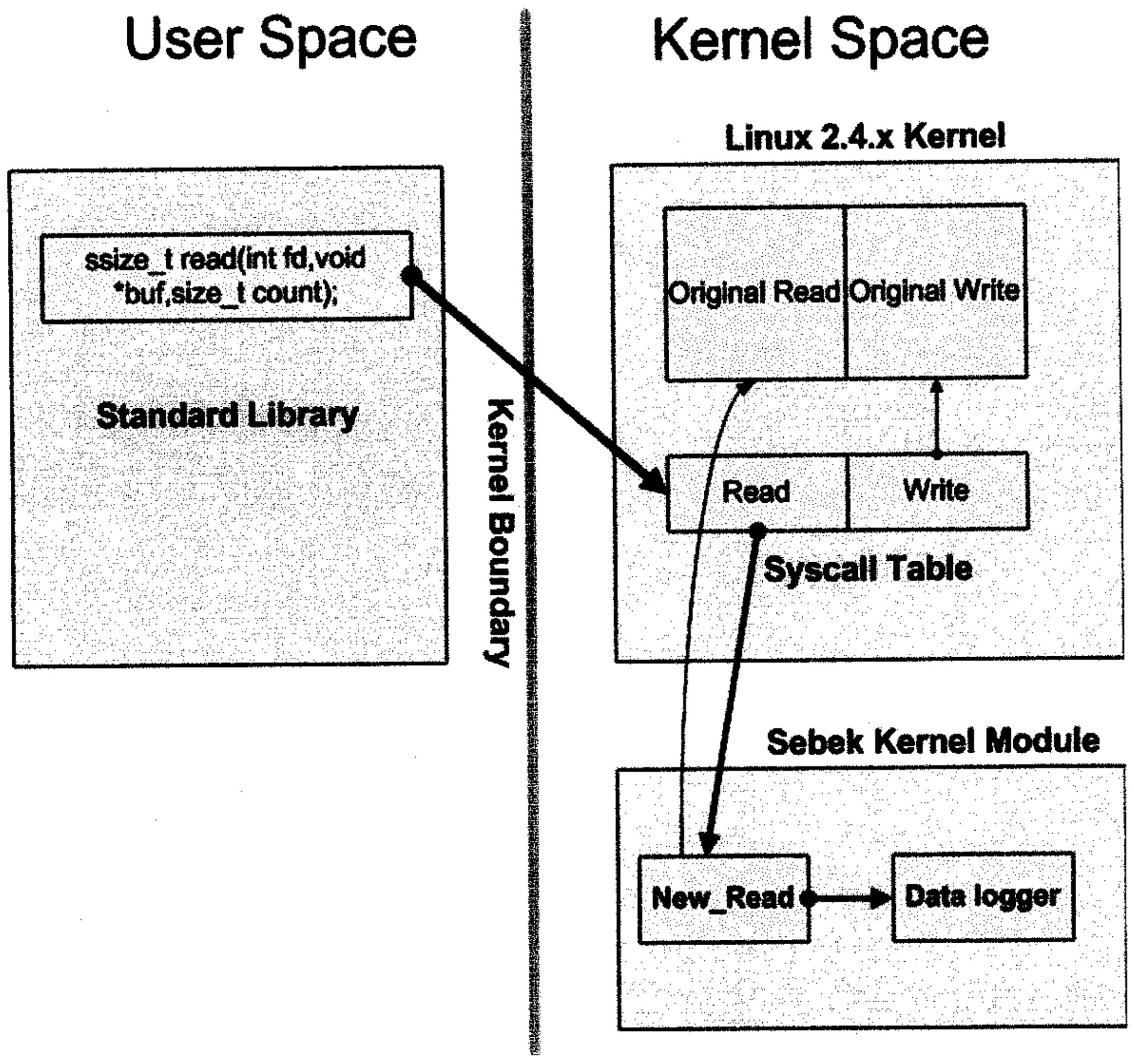
4. William Stallings " *Business Data Computing*" Fourth Edition, retrieved on February 15, 2005

5. Edward Balas 22 July 2001, *Know Your Enemy: Sebek, A kernel base data capturing tool*, retrieved on January 13,2005 <http://project.honeynet.org/article3.htm>,

6. Nurris Ishak (New Straits Times), 15th May 2001 "In Malaysia, it's a hackers heaven" retrieved on 2 June 2005 <http:www.mycert.org.my/press.htm>

Appendices

Appendix 1



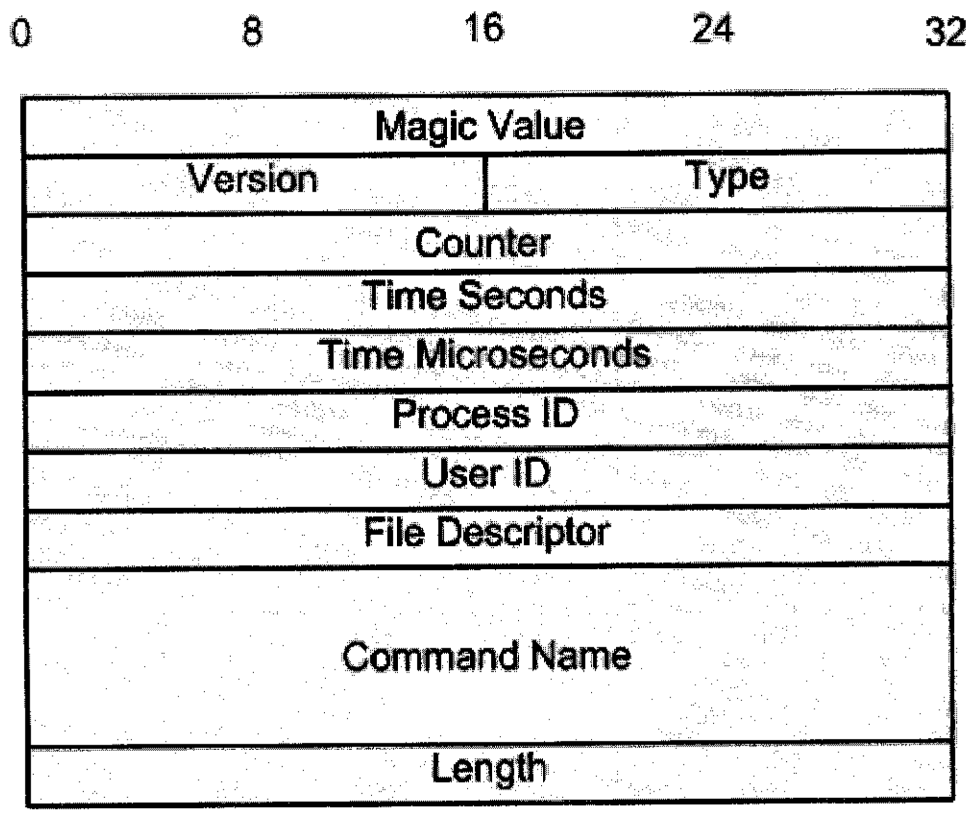
Appendix 2: keystroke example

```
[2003-07-23 20:03:45 10.0.0.13 6673 bash 500]whoami
[2003-07-23 20:03:48 10.0.0.13 6673 bash 500]who
[2003-07-23 20:03:50 10.0.0.13 6673 bash 500]su
[2003-07-23 20:03:57 10.0.0.13 6886 bash 0]cd /var/log
[2003-07-23 20:03:57 10.0.0.13 6886 bash 0]ls
[2003-07-23 20:04:01 10.0.0.13 6886 bash 0]mkdir ...
[2003-07-23 20:04:20 10.0.0.13 6886 bash 0]tcsh
[2003-07-23 20:04:20 10.0.0.13 6921 tcsh 0]0
[2003-07-23 20:04:20 10.0.0.13 6920 tcsh 0]vt
[2003-07-23 20:04:20 10.0.0.13 6920 tcsh 0]en
[2003-07-23 20:04:20 10.0.0.13 6920 tcsh 0]en
[2003-07-23 20:04:27 10.0.0.13 6920 tcsh 0]cd /tmp
[2003-07-23 20:04:28 10.0.0.13 6920 tcsh 0]ls
[2003-07-23 20:04:42 10.0.0.13 6920 tcsh 0]cd /usr/lib
[2003-07-23 20:04:42 10.0.0.13 6920 tcsh 0]ls
```

Appendix 3: *Structure of a Sebek packet sent by the client.*

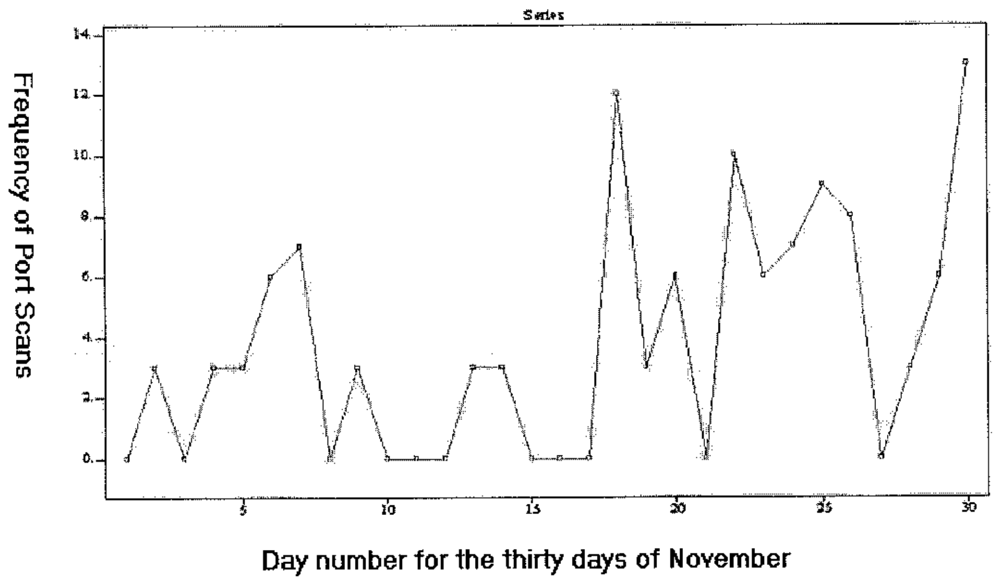
Field Name	Data Type	Description
Magic	**Unsigned 32 bit Int**	Along with the DST Port, Magic is used by Sebek to identify packets which should be hidden
Version	**Unsigned 16bit Int**	Sebek Protocol Version, current version is "1".
Type	**Unsigned 16bit Int**	Type of record represented. Read data is type 0, Write data is type 1. Currently only Read is implemented.
Counter	**Unsigned 32bit Int**	PDU counter, used to identify when packets are lost. This counter restarts at 0 when installed.
Time_sec	**Unsigned 32bit Int**	Seconds since UNIX epoch according to the honeypot
Time_usec	**Unsigned 32bit Int**	Residual Microseconds
PID	**Unsigned 32bit Int**	Process ID
UID	**Unsigned 32bit Int**	User ID
FD	**Unsigned 32bit Int**	File Descriptor
Com	**12 Character Array**	Records the first 12 characters of the command's name.
Length	**Unsigned 32bit Int**	Length in octets of the PDU's body.

Appendix 4: Sebek Record Header. This header comes after the IP/UDP header, and is followed bythe data representing the activity on the honeypot (attacker keystrokes, files, passwords,etc).



Appendix 5

Number of Port Scans per Day



7. The numbers of illegal port scanning for the entire November 2000 in the case study from the paper entitle ‘ know your enemy: motive, The motives and psychology of the black hat community’

Appendix 7 (Coding)

sbk_ks_log

```
use strict;
use 5.004;
use Getopt::Std;
use Time::gmtime;
use POSIX;
use FileHandle;
use Socket;


my $ks_read_length_limit = 100;  #----- this is the maximum size of a read we
                                   #----- will consider to be kestroke based
                 #----- yes, this approach does suck


sub main{

  my %dat;
  my $line;


  #---- take records from sebeksniff via STDIN
  while(read(STDIN,$line,52,0) > 0){

       my $ip;
    my $magic;
       my $ver;
    my $type;
       my $counter;
       my $time_sec;
       my $time_usec;
       my $pid;
       my $uid;
       my $fd;
       my $com;
       my $len;
       my $data;

       my $return_code;


       ($ip,$magic,$ver,$type,$counter,$time_sec,$time_usec,$pid,$uid,$fd,$com,$len) =
         unpack("NNnnNNNNNNa12N",$line);


       read(STDIN,$data,$len,0);

       next if($type != 0 ||$len > $ks_read_length_limit);

       $com =~ s/\0//g;

       my $tm = gmtime($time_sec);
```

```
        my $datetime = strftime("%Y-%m-%d %H:%M:%S",$tm->sec,$tm->min,$tm->hour,$tm-
>mday,$tm->mon,$tm->year,$tm->wday,$tm->isdst);


        $dat{$ip}{$pid}{$fd}{"data"}           .= $data;
        $dat{$ip}{$pid}{$fd}{"uid"}{$uid}       = 1;
    $dat{$ip}{$pid}{$fd}{"com"}{$com}       = 1;

        if($data =~ m/\n|\r/){
          my $log;
          my $uid_str;
          my $com_str;
          my $u;
          my $c;

          my $addr = inet_ntoa(pack("N",$ip));
          $log  = $dat{$ip}{$pid}{$fd}{"data"};


          #----- map control characters
          $log =~ s/\x1b\[A/[U-ARROW]/g;
          $log =~ s/\x1b\[B/[D-ARROW]/g;
          $log =~ s/\x1b\[C/[R-ARROW]/g;
          $log =~ s/\x1b\[D/[L-ARROW]/g;
          $log =~ s/\x1b\[3~/[DEL]/g;
          $log =~ s/\x1b\[5~/[PAGE-U]/g;
          $log =~ s/\x1b\[6~/[PAGE-D]/g;
          $log =~ s/\x7f/[BS]/g;
          $log =~ s/\x1b/[ESC]/g;


          #----- scrub other nonascii values
          $log =~ s/[^\x20-\x7e]//g;

          my $x = 0;
          foreach $u (keys %{$dat{$ip}{$pid}{$fd}{"uid"}}){
                if($x++){
                  $uid_str .= "/$u";
                }else{
                  $uid_str .= "$u";
                }
          }

          $x = 0;
          foreach $c(keys %{$dat{$ip}{$pid}{$fd}{"com"}}){
                if($x++){
                  $com_str .= "/$c";
                }else{
                  $com_str .= "$c";
                }
          }


          print "[$datetime  $addr $pid $com_str $uid_str]$log\n";

          #----- delete the record
```

```
            undef $dat{$ip}{$pid}{$fd};

          }

    }

}

main();
```

sbk_update

```
use strict;
use 5.004;
use Getopt::Std;
use Time::gmtime;
use DBI;
use DBD::mysql;
use POSIX;
use FileHandle;
use Socket;



my $dbh;

my $database  = "sebek";
my $dbpasswd   = "";
my $dbuid      = "sebek";
my $dbserver   = "localhost";
my $dbport     = "3306";


sub main{

   my $old_counter = 0;
   my $total;
   my $lost;



   my $ip;
   my $magic;
   my $ver;
   my $type;
   my $counter;
   my $time_sec;
   my $time_usec;
   my $pid;
my $uid;
my $fd;
my $com;
my $len;
my $data;
my $return_code;



eval{
        require DBI;
};
if($@){
        print STDERR " $0: needs DBI\n";
        exit 1;
}

#------- get user input
my %opt;

getopts("u:p:d:s:P:h",\%opt);

if($opt{u}){
        $dbuid = $opt{u};
}

if($opt{p}){
        $dbpasswd = $opt{p};
}

if($opt{d}){
        $database = $opt{d};
}

if($opt{s}){
        $dbserver = $opt{s};
}

if($opt{P}){
        $dbport = $opt{P};
}



if($opt{h}){
        print "$0:(Loads Sebek records into specified mysql database)\n";
        print "\t-u  User ID\n";
        print "\t-p  Passwd\n";
        print "\t-d  Database Name\n";
        print "\t-s  Server Name or IP\n";
        print "\t-P  Port Number\n";
        print "\t-h  Help\n";
        exit;

}
```

```
  $dbh = DBI-
>connect("DBI:mysql:database=$database;host=$dbserver;port=$dbport",$dbuid,$dbpasswd);

  if(!defined($dbh)){
        warn "Unable to get access to database\n";
        exit;
  }
  $dbh->{LongReadLen} = 16384;


  #----- only need to create query and prepare it once.
  my $sql  = "INSERT INTO read_data (ip_addr, time, counter, command, filed, pid, uid, length, data)";
  $sql .= "  VALUES ( ? ,? , ? , ? , ? ,  ? , ? , ? , ? ) ;";

  my $query = $dbh->prepare($sql);

  my $line;

  #---- take records from sebeksniff via STDIN
  while(read(STDIN,$line,52,0) > 0){


        ($ip,$magic,$ver,$type,$counter,$time_sec,$time_usec,$pid,$uid,$fd,$com,$len) =
          unpack("NNnnNNNNNNa12N",$line);

        read(STDIN,$data,$len,0);

        next if($type != 0);

        $total++;

        if($counter - $old_counter > 1){
           $lost = $counter - ($old_counter +1 );
           warn "   $lost records missing\n";
        }
        $old_counter = $counter;

        $com =~ s/\0//g;

        my $tm = gmtime($time_sec);
        my $datetime = strftime("%Y-%m-%d %H:%M:%S",$tm->sec,$tm->min,$tm->hour,$tm-
>mday,$tm->mon,$tm->year,$tm->wday,$tm->isdst);



        $query->bind_param( 1,$ip       );
        $query->bind_param( 2,$datetime );
        $query->bind_param( 3,$counter  );
        $query->bind_param( 4,$com      );
        $query->bind_param( 5,$fd       );
        $query->bind_param( 6,$pid      );
        $query->bind_param( 7,$uid      );
        $query->bind_param( 8,$len      );
        $query->bind_param( 9,$data     );
        $return_code = $query->execute;

    if(!$return_code){
            sleep 2;
            warn ("reconnecting to the datbase\n");
            $dbh = DBI-
>connect("DBI:mysql:database=$database;host=$dbserver;port=$dbport",$dbuid,$dbpasswd);
            $query = $dbh->prepare($sql);
        }

  }


}

main();
```