Edith Cowan University
Research Online

ECU Publications Pre. 2011

2006

Honeypots: How do you know when you're inside one?

Simon Innes
Edith Cowan University

Craig Valli
Edith Cowan University

This article was originally published as: Innes, S. , & Valli, C. (2006). Honeypots: How do you know when you're inside one?. Proceedings of Australian Digital Forensics Conference. (pp. 78-83). Edith Cowan University, Perth, Australia. Edith Cowan University. Original article available here
This Conference Proceeding is posted at Research Online.
http://ro.ecu.edu.au/ecuworks/2105

Honeypots: How do you know when you are inside one?

Simon Innes & Craig Valli
Edith Cowan University
sinnes@student.ecu.edu.au
c.valli@ecu.edu.au

Abstract

This paper will discuss honeypots and their use and effectiveness as a security measure in an IT environment. It will specifically discuss various methods of honeypot implementations. Furthermore, this paper will look into the weaknesses within a honeypot system. This will include attacks against honeypots and methods a hacker may use to detect the presence of a honeypot or the fact that he/she is actually inside one. Finally this paper will discuss methods of further securing honeypots and ways the community is dealing with security flaws as they are identified

Keywords Honeypot, Honeynet, Detection

INTRODUCTION

As the world becomes more computer and more network centric, there is a required need for security. In the past, security practitioners would have to wait for an attack to occur before they could dissect it and then find methods to defend or mitigate against it. It is hard to predict what initialisation vector, covert channels or mode of operation attackers will take in an effort to compromise or disable a system. In an effort to try and learn about how the attackers operate and the kind of techniques they employ, a technology called a 'honeypot' was developed. A honeypot is basically a system that emulates a weak or vulnerable system in a effort to try and attract a potential attacker to try and crack or break the machine while the honeypot is logging all the activity that occurs (Spitzner, 2002). This paper will discuss various honeypot technologies and it will then go on to discuss the various weaknesses each type of honeypot has and methods that attackers would use to help determine that they are in fact inside a honeypot.

As the technology of honeypots develops, the community comes up with different methods of how to handle them. The first of these methods is known as User Mode Linux or UML. UML is basically a Linux kernel that can be run from within another running Linux system. From this, the user has the ability to forward any packets or network activity to the UML to give the person attempting a connection the feel that he or she is actually inside a live Linux system. With a UML system, a user can set up any program or service to run just like a normal Linux machine. When an attackers attempts a connection to a certain service, say ssh, they will be given a good representation of a live system because it will essentially be the same service. The advantage of UML is that it has the ability to carry out TTY logging. Most honeypots will attempt to do logging via packet sniffing which is ok until the attacker decides to connect via ssh. The TTY logging facility has the advantage of being able to record exactly what the attacker types in. This helps immensely when attempting to work out how an attack has occurred.

VMWare is another tool that is becoming increasingly popular as a honeypot tool. VMWare was created to act as a virtual machine that gave a user the ability to run another operating system at the same time as their current working environment. The VMWare virtual machine emulates hardware by providing a set of virtual device drivers. Because these drivers are the same on any VMWare virtual machine, the images are easily ported from one host to another. People take advantage of these virtual machines and set them up as honey pots. A VMware virtual machine is given it's own IP address and is contactable over a network so it would be easy for someone to simulate a specific host and network layout. The virtual machine will then simply exist on the network as it's own entity and people may set routers to point it when they need traffic directed to a honeypot. The advantage of

VMWare over User Mode Linux is that on any machine that can run VMWare, you can have a guest operating system of most x86 built operating systems such as Linux, Windows and Solaris 10 for x86. VMWare also has the added advantage of being able to boot from a livecd image. This can occur when the image is stored on a hard disk so there is no need to burn it to a CD.

One of the more popular honeypot solutions that are around today is called honeyd. Honeyd is a daemon that has been written for UNIX platforms but now also exists for Windows. It is designed to emulate multiple hosts and networks on a single machine. In emulating the hosts, honeyd has the ability to appear as any operating system the user likes. This is achieved by returning packets that look like the various nmap os fingerprinting packets. This way if someone was to run a scan of the network to try and determine the different types of hosts, an accurate response will be given. Certain services, such as IIS or the telnet daemon on a Cisco router, can be emulated by installing or writing perl scripts. These scripts will allow an attacker to connect to a host on that specific port and it will give them the impression they are on the actual machine. These scripts can be written for any service a users wishes to emulate. Honeyd works by listening for packets aimed at certain ip addresses and ports. From there, the configuration file will tell it what it has to respond with. On a single honeyd host, 65536 ip addresses can be used on an emulated network. This makes honeyd a very flexible solution for a honeypot.

Another method of creating a honeypot is to use a chroot or a jail. This is similar to the virtual machine solutions, only it is run on the same host and kernel as the main operating system. A chroot basically limits a user to a specific area of a system but at the same time it gives them the impression that they are at the top level of the filesystem. This allows for the creation of a small, simulated environment in an area of a system that won't damage anything else. Say you had your honeypot environment set up in /tmp/honeypot, you could set a users chroot to be /tmp/honeypot and when they logged in, they would appear as though they were in /. From here you can set up your simulated server and see what the attacker attempts to do. A simulated system would need to consist, at the very least, dummy files that would give them impression of a real system. This would include things such as the /dev and /proc directories as well as the standard operating system files that exist within /usr and /etc. With that being said, it would also be possible to simulate different versions or distributions of Linux by placing the OS specific files within the chroot.

A honeypot seems to be a competent tool to help aid in the discovery of attack techniques so that people may design and implement defences against them, but as with any system, there are people around that try to break them. Honeypots are of concern to attackers because many of them do not want their methods known. If a hacker finds an exploit in a server or a piece of software, before this becomes public knowledge it is known as a zero day attack. If an attacker was to perform a zero day attack on a honeypot, it is likely to become public knowledge a lot faster and as a result, the value of the attackers advantage will be reduced if not expunged. Because of this, attackers have developed techniques to help them detect when they are inside a honeypot and in some cases they have developed ways to break them. This paper will now discuss methods that have been discovered for determining whether or not you are inside a specific type of honeypot.

HONEYPOT DETECTION AND ATTACK

Detecting a UML Honeypot

User Mode Linux has quite a few problems that can quite easily be picked up by a hacker that knows what he or she is looking for. For starters, UML doesn't use real hard disks to store data on. Instead it uses virtual devices that point at all ready existing portions of a file system that contain disk images. These images are mounted on the UML system as /dev/ubd*. The simplest way to check for this would be to run 'df –k' or 'mount' to see what is mounted where. In later releases of UML, modules were developed to try and hide the /debv/ubd* devices and make them look like real drives. To access these modules a user must start UML with the fake idea and fakehd options switched on. The limitation of this is that the major number identifying the /dev/ubd* devices is not the same as the one that is used for standard IDE or SCSI systems. This number is 98(0x62).

Another quick way to tell whether or not you are inside UML instead of a real system is to have a look at the /proc tree. There are several anomalies here that are quite obvious to an advanced attacker. Firstly there is cpuinfo:

```
cat /proc/cpuinfo
        processor : 0
        vendor_id : User
        model name : UML
        mode : tt
        [...]
```

The two obvious problems here are the fact that the model name is listed as UML and the mode being reported is Tracing Thread mode.

```
egrep "umllhoney" /proc/ksysms
        a02eb408 uml_physmem
        a02ed688 honeypot
```

The ksysms file shows a memory allocation to a honeypot.

```
cat /proc/devices
[...]
Block devices:
[...]
 60 cow
 90 ubd
```

The devices file shows the disks are labelled ubd which is how they are referred to in /dev/ before fakehd is run.

```
cat /proc/filesystems
[...]
nodev   hostfs
```

The filesystems file shows that there is a hostfs filesystem running which is a good sign that you are running on a virtual machine as it needs to link back to the hosts filesystem. A tool called ‘hppfs’ (Honeypot Proc File System) has been developed to allow the customisation of the /proc tree with UML, but this is a very long and complicated task and is often prone to human error. From a hacker’s perspective, a tool called UMLfp has been developed to help detect whether or not you are inside a UML virtual machine.

Detecting and Attacking VMWare Honeypots

VMWare, being a virtual machine solution like UML, has very similar problems with respect to masking virtual identities of hardware. Prior to version 4.5 of VMware, a lot of the hardware was not configurable. The following table shows what hardware was to remain at the default and what the default value was:

Hardware	Reported Type
Video Card	VMware Inc [VMware SVGA II] PCI Display Adapter
Network Interface Card	Advanced Micro Devices [AMD] 79c970 [PCnet 32 LANCE] (rev 10)
Hard disk	VMware Virtual IDE Hard Drive
CD Drive	NECVMWar VMware IDE CDR10
SCSI Controller	VMware SCSI Controller

This naturally makes the detection of a VMWare honeypot extremely easy. To help combat this problem, a member of the French Honeypot Project, Kostya Kortchinsky, designed a patch that would allow these default values to be changed.

The next major failure of a VMWare honeypot is the network MAC address that is bound to the network card. The vendor part of the MAC address (the first three octets) on a VMWare virtual network interface is always one of the following three values:

00-05-69-xx-xx-xx

00-0C-29-xx-xx-xx

00-50-56-xx-xx-xx

This means on a Windows machine, an attacker simply has to run ‘ipconfig /all’ and on a UNIX machine run ‘ifconfig –a’ to see if the MAC address is one from a virtual device. The running of arp –a would also allow an attacker to scan a network for these MAC addresses.

The last weakness of a VMWare honeypot is an I/O backdoor that the developers have left open for the configuration of the virtual machine during runtime. There are a few lines of assembly code that are used to access this backdoor:

```
mov eax, VMWARE_MAGIC ; 0x564D5868
mov ebx, b ; <parameter of command>
mov ecx, c ; <number of command>
mov edx, VMWARE_PORT ; 0x5658
in eax, dx
```

Some examples of the commands that can be used via this are as follows:

04h - Get current mouse cursor position.

05h - Set current mouse cursor position

06h - Get data length in host's clipboard.

07h - Read data from host’s clipboard

08h - Set data length to send to host's clipboard.

09h - Send data to host’s clipboard

0Ah - Get VMware version

0Bh - Get device information

An attacker could potentially run this code, see that the command they have attempted to execute was successful and determine that they are inside a VMWare virtual machine. The use of this backdoor could also aid in disrupting the running VMWare virtual machine. This could be achieved by causing buffer overflow or disrupting mouse movement across the screen. Another patch by Kostya Kortchinsky will change the value of VMWARE_MAGIC in an attempt to hide the backdoor from an attacker, although it is unknown how effective this method is.

Detecting a chroot environment

One of the common ways of securing honeypot binaries and systems is the use of chroot jail environments. The easiest way to tell whether or not you are inside a chroot enrvironment is to simply run a ls –lia on the root directory. The thing to take note of when doing this is the inode of the '.' and '..' directories. On a standard system you will notice that the inodes for these 2 directories are both 2.

```
2 drwxr-xr-x   21 root     root        2096 Oct 16 18:17 .
2 drwxr-xr-x   21 root     root        2096 Oct 16 18:17 ..
```

If you are in a chroot and you run the same command, the inodes will be quite different:

```
1553552 drwxr-xr-x    6 1000     100        4096 Dec 14 12:58 .
1553552 drwxr-xr-x    6 1000     100        4096 Dec 14 12:58 ..
```

The inodes in this example are the same inodes as the directory that has been chrooted to.

Detecting honeyd honeypots

Honeyd has different functionality and principles of operation of example honeypots previously covered, but it still has some limitations of its own. The first main problem with honeyd is that it has tendencies to respond to packets it has not business responding to. If someone was to send a malformed network packet to a honeyd host (such as a packet that opens and closes a connection straight away) the honeyd host will still respond as though it has received a valid packet. This will normally be missed typically an attack tool that is used is not expecting to hear a response. To prove this fact the attacker would need to be watching the network with a packet sniffer.

The next limitation of honeyd is the way it handles operating system fingerprinting via TCP/IP stack interactions. Unfortunately the nmap method of TCP/IP stack fingerprinting to estimate a network devices operating system is not perfect and can often return false positive results (Valli, 2003). If this was to happen to someone's honeypot during an attackers scan, it would be likely to raise suspicion.

Because honeyd relies a lot upon user configuration, the concept of human error needs to be taken into account. There are many semantic errors that one could type into the honeyd configuration file that would not be picked up at start time because technically they are not wrong. A mistake of this nature that could be made is emulating an IIS server on a Linux machine, or even a new version of IIS on something like Windows NT 4.0.

The final problem with honeyd is the actual service emulation. As these are just simple perl scripts it would be easy for someone to, again, get something semantically or topically incorrect and recognise the ruse. Also the reliability of the scripts can play a role in letting someone know they are inside a honeypot. For example, the default route-telnet.pl script that comes with the standard install of honeyd has tendencies to malfunction and stop processing input. Something like this would be cause for concern for an attacker and probably prompt them to investigate further.

General issues with honeypots

There are a few issues that can pose problems to honeypots in general. Most of these centre around timing. Because the honeypots are logging as the attacker is using the machine, extra instructions are being executed and as a result the total execution time of a process or a command could be extended to the point where it is obvious

that something is circumspect. Furthermore, honeypots do not scale well in an attack scenario when there are a large number of attacking hosts and sufficient available bandwidth. Consequently they suffer significant performance degradation under a sustained attack and in some cases may fail altogether.

Also commands that are being emulated will need to ensure they mimic the original command well so as not to raise suspicion. This is not an easy task when dealing with an attacker who has intimate, expert knowledge of a service or particular protocol.

CONCLUSION:

Honeypot technologies have proven themselves as valuable tools for researching new attacks and malware. are a really aid in determining the types of attacks and vulnerabilities that exist and allow for the identification and examination of new attacks. Unfortunately, attackers and malicious code are cognisant of the existence of honeypot systems and are developing countermeasures to their existence. Experienced attackers are now being cautious of machines they penetrate and are taking steps to ensure they are not being watched by using the techniques elucidated above.

There is significant research opportunity on both sides of this argument i.e. detection and obfuscation of honeypots. This research is not only the domicile of the traditional good vs. evil, black vs. white fight between defender and attacker it has other potentialities. As one example a honeypot developer may garner significant competitive advantage by releasing how to detect their competitor's honeypot in the wild and hence making their competitors systems less effective.

Like most security technologies, it is simply a case of building a better mouse trap. Eventually people will come up with ways of logging that are stealthy, less detectable and look more realistic to potential attackers. In turn attackers will become more adept at detecting the cheese is rancid and the cycle goes through another iteration.

REFERENCES

Corey, J. (2005). Advanced Honey Pot Indentification and Exploitation. Retrieved October 9th 2005 from http://www.phrack.org/fakes/p63/p63-0x09.txt

Dike, J. (2005). UML as a honeypot. Retrieved October 9th 2005 from http://user-mode-linux.sourceforge.net/honeypots.html

Holz, T. Raynal, F. (2005). Defeating Honeypots: System Issues, Part 1. Retrieved October 9th 2005 from http://www.securityfocus.com/infocus/1826

Holz, T. Raynal, F. (2005). Defeating Honeypots: System Issues, Part 2. Retrieved October 9th 2005 from http://www.securityfocus.com/infocus/1828

Oudot, L. Holz, T. (2004). Defeating Honeypots: Network Issues, Part 1. Retrieved October 9th 2005 from http://www.securityfocus.com/infocus/1803

Oudot, L. Holz, T. (2004). Defeating Honeypots: Network Issues, Part 2. Retrieved October 9th 2005 from http://www.securityfocus.com/infocus/1805

Provos, N. (2004). Honeyd FAQ. Retrieved October 2nd 2005 from http://www.honeyd.org/faq.php

Spitzner, L. (2002). *Honeypots : tracking attackers*. Boston: Addison-Wesley.

The Honeynet Project. (2002). Know Your Enemy: Learning With User-Mode Linux. Retrieved October 9th from http://www.honeynet.org/papers/uml/

The Honeynet Project. (2003). Know Your Enemy: Sebek. Retrieved October 15th 2005 from http://www.honeynet.org/papers/sebek.pdf

The Honeynet Project. (2005). Sebek FAQ. Retrieved October 15th 2005 from http://www.honeynet.org/tools/sebek/faq.html

Valli, C. (2003). *Honeyd - A fingerprinting Artifice*. Paper presented at the 1st Australian Computer, Information and Network Forensics Conference, Scarborough, Western Australia.

COPYRIGHT

Simon Innes & Craig Valli ©2006. The author/s assign SCISSEC & Edith Cowan University a non-exclusive license to use this document for personal use provided that the article is used in full and this copyright statement is reproduced. The authors also grant a non-exclusive license to SCISSEC & ECU to publish this document in full in the Conference Proceedings. Such documents may be published on the World Wide Web, CD-ROM, in printed form, and on mirror sites on the World Wide Web. Any other usage is prohibited without the express permission of the authors