Institut Eurécom

Corporate Communications Department

2229, route des Crêtes

B.P. 193

06904 Sophia Antipolis

FRANCE

Research Report RR-03-082

White Paper: Honeypot, Honeynet: A comparative survey[1]

September 14, 2003

Fabien Pouget, Marc Dacier

Institut Eurécom
Email: {pouget,dacier@eurecom.fr}

[1] This research is supported by a research contract with France Telecom R&D, contract N. 425-17044

White Paper:
"Honeypot, Honeynet: A comparative survey[2]"

Fabien Pouget, Marc Dacier

Institut Eurécom

Email: {pouget,dacier@eurecom.fr}

Institut Eurécom

2229, Route des Crêtes ; BP 193

06904 Sophia Antipolis Cedex ; France

Abstract

Many different tools defined as honeypots, honeynets and other honeytokens have been proposed on the Internet during the last 3 years. However each solution suits well for some specific needs and can be inadequate in many other cases. In this document, we offer to help the reader having a good overview of existing tools. We present their main features and we describe some of them with more details.

1. Introduction

Honeypots, honeytokens and honeynets have been used for some time in computing systems even if the use of this terminology is recent. During the last two years, many different implementations of these concepts have been proposed and this paper intends to provide a complete overview of current available solutions. We present their main characteristics and we describe with more details three of them.

By exploiting well-defined concepts used by the dependability community, we define honeypots as specific environments where vulnerabilities have been deliberately introduced in order to observe intrusions [poSt03, page 32]. We report the interested reader to [PoDa03a] for a more

[2] This research is supported by a research contract with France Telecom R&D, contract N. 425-17044

detailed presentation of this honeypot terminology. Many simple implementations are classified as honeypots (an unused web server that logs intrusion attempts for instance), but more sophisticated solutions can also be found in the Internet. This paper focuses on these existing honeypot tools only. Our comparative survey aims at giving a complete overview of the current solutions: their advantages, limitations and implementation requirements. Moreover, we invite the interested reader to visit [HonWeb]. It is a web site dedicated to honeypots and IDS which provides a list of currently available honeypots as well as some links to relevant papers and web pages on the subject.

The paper is organized as follows. Section 2 lists honeypots solutions that are currently available. Section 3 focuses on so-called 'honeynet' solutions (see [PoDa03a]). And Section 4 provides more in-depth information on three of these solutions: Deception Toolkit (DTK), LaBrea Tarpit and Honeyd.

2. Today Available Tools

There are several free and a few commercial honeypots available on the market. Their functionality differs greatly, as well as their complexity and ease of use.

In this section a close look will be taken at today's available solutions. This is for information only and many changes are possible within the next few months.

- **Symantec Decoy Server** is the successor of **ManTrap™**, a commercially honeypot implementation by Recourse Technologies. Symantec Corp. acquired Recourse Technologies in July 2002. This acquisition brought Recourse's Mantrap into the Symantec portfolio with a new commercial name: Symantec Decoy Server. Consequently both names refer to the same product, which is characterized on its home page by:

  "Symantec Decoy Server can create a virtual minefield that an internal attacker must successfully navigate in order to reach his target. One step in the wrong direction and the attacker is exposed" [ManT03]

  The main concepts of Symantec Decoy Server are so-called cages (see figure 1). A cage is basically a copy of the host operating system connected to a dedicated network

interface card. During installation the operating environments inside the cages are generated to be essentially the same that of the host. The Symantec Decoy Server software also installs a kernel wrapper that controls the interaction between the cages and the host kernel. Consequently the cages are presented on the network as four individual systems, each with its own network interface. All relevant activities in the cages are logged, such as keystrokes, process invocation and file accesses for later analysis.

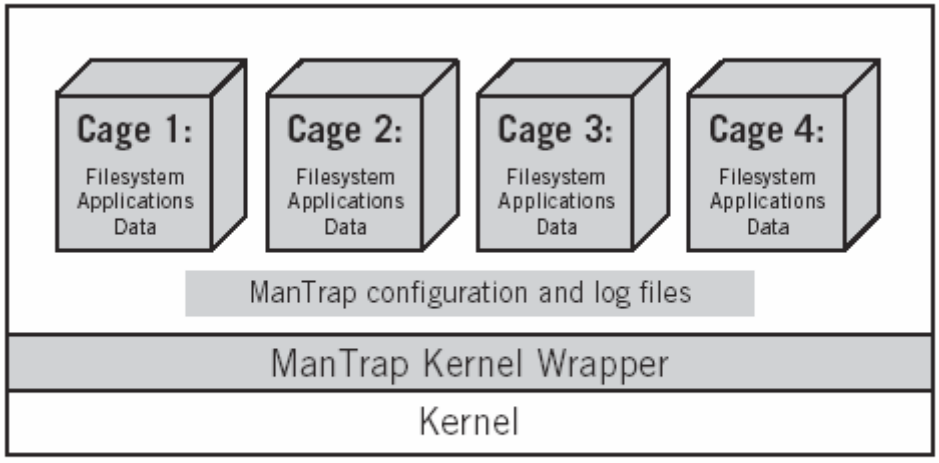

Figure 1 - ManTrap Host

- **Deception Toolkit (DTK)** is a set of free scripts written in Perl by Fred Cohen [Coh99]. *"DTK is a toolkit designed to give defenders a couple of orders of magnitude advantage over attackers."*

It uses deception to counter attacks. The basic idea is to make it appear to attackers as if the system running DTK has a large number of vulnerabilities. One very interesting feature of the DTK is the so-called *deception port*. Fred Cohen proposes that a listener on TCP port 365 should indicate whether a machine one is trying to connect to is running a deception defense in the hope that attackers who wish to avoid deceptive defenses will check there first.

- **Specter** is a commercially available honeypot by NeoWorx, a Swiss group [Spec03]. It simulates a complete machine, providing an interesting target for hackers to lure them away from the real machines.

- **BackOfficer Friendly (or simple BOF)** was developed by Marcus Ranum and Andrew Lambeth. They are members of the team that created the NFR -Network Flight Recorder- a commercially available IDS [Bof03].
BOF works basically like Specter with the difference that the program is much simpler. It was released in 1998 and it is freely available for personal use on the NFR website.

- **HoneyWeb** by Kevin Timm is a deception based web server program that can be used as a standalone server or in conjunction with Honeyd (see 4.3 for Honeyd information). This http server written in Python returns different server versions depending on http requests listened on port 80 and logs activity detected on it. It does basic regex comparison to incoming request to determine what associated headers to return. HoneyWeb works in two modes "Persistent" and "Non- Persistent". In "Non-persistent" mode HoneyWeb is basically a more intelligent netcat and returns back *200 OK* for every request, unless defined otherwise, along with the other associated headers for that type of server. In "Persistent" mode HoneyWeb remembers the IP and always returns the same server version to the same IP for a specified period of time, in addition it does basic request comparisons between server families to determine if a *404* should be sent back or not. For example a host whose requests are distinctly Unix like requests receives *404* for distinctly Microsoft style requests. Moreover, HoneyWeb does some bogus request checking and sends back server specific error pages on bogus requests. Attack specific pages can be specified to make HoneyWeb appear more real for interactive attackers.

- **KFSensor** is developed by Keyfocus [KFsens]. It is a host based Intrusion Detection System (IDS). It acts as a honeypot to attract and detect hackers by simulating vulnerable system services and Trojans by opening ports on the machine it is installed on and waiting for connections to be made to these ports. It does this in exactly the same way

as conventional server software, such as a web server or an SMTP server. By doing this it sets up a target, or a honey pot server, that will record the actions of a hacker. KFSensor has begun an open beta testing program and is currently available for free.

- **The Bait N Switch Honeypot** developed by Team Violating is defined as "an active and aggressive part of the network security infrastructure" [BaitSw]. It reacts to intrusion attempts by redirecting all traffic from 'bad' IP addresses to a honeypot that is partially mirroring the production server. Once switched, the hacker is unknowingly attacking the honeypot instead of the real data while the client and/or users still safely accessing the real system. This is not a honeypot use. It is based on snort, linux iproute2 and netfilter [Lin03]. The honeypot component itself can be chosen independently. Whereas its installation is quite arduous, its concept is very promising.

- **Big Eye** developed by Team Violating is a network utility (dump), which can be run in different modes. It can run as a sniffer, as a tcp/udp/icmp connection logger, be bound to a port and listen for tcp/udp incoming connections, or as a honeypot. The honeypot mode is an emulation scheme to mimic applications protocols such as: ftp or http. This is a low to medium interaction honeypot [BigEye].

- **Smoke Detector** is a commercially available hardware honeypot by Palisade [Smok03]. It is a drop-in network appliance that provides defensive decoy and detection capabilities including alerting and reporting of unauthorized access attempts. It mimics interesting or potentially vulnerable elements on a network for the purpose of attracting and detecting inappropriate activities. It can be configured to emulate up to 19 distinct networked machines in varying configurations of operating systems and services. Some complementary tools for analyzing logs are also available.

- **Tiny Honeypot** (also called THP) is developed by George Bakos. "The goal isn't to fool a skilled, determined attacker...merely to cloud the playing field with tens of thousands of fake services, all without causing unreasonable stress on the [tiny honeypot] host". It is a simple honeypot program based on IPTables redirects, an xinetd listener. It

listens on every TCP port not currently in use, logging all activity. Furthermore, it is possible to attach to various ports so called 'responders' which are simple scripts that provide limited interaction to fool most automated attack tools, as well as quite a few humans, at least for a little while. So it can be used as an addition to the state and content-aware Intrusion Detection System Snort [Snort03], insofar as it allows nearly every connection attempt to complete. Thus the content rules have a chance to actually fire, rather than depending on simple port and protocol "context" filters [THP03].

- **NetFacade** is a commercially available honeypot produced by Verizon since 1999 [NetF03]. The Verizon NetFacade Intrusion Detection service creates a Honeynet that exists to alert network security or management personnel of an intrusion. In addition, it distracts intruders from probing and attacking the real targets on a network. NetFacade can simulate a network of hosts running seemingly vulnerable services. A scan of the range of IP addresses the NetFacade is simulating will return information on the simulated services as if they were real network services running on actual hosts. Since there are no actual users of this virtual network of simulated hosts, all traffic to it is considered to be suspicious. All traffic to the NetFacade Intrusion Detection service on the virtual network is logged. Little information is currently available since it uses mostly proprietary techniques.

- **Honeyd** developed by Niels Provos and **LaBrea Tarpit** developed by Tom Liston will be presented in the next chapter. They are two promising mid-interaction honeypots.

Table 1 summarizes some honeypot functionalities which have been discussed previously. It is not exhaustive and information may change over time. However, it gives a first approach for today's available tools and some of their characteristics. The column 'Maintained' gives an indication of the dynamism concerning the tool updates and public discussions about its evolution.

Table 1: Honeypots comparison 1

	Level of Interaction	Open Source	Log files	Fake services	OS simulation	OS	Maintained	Requirements	Langage
BOF [Bof03]	Low	No	No	7 (telnet, ftp, smtp, http, pop3, imap2)		Win 32, Unix	Not really		
Specter [Spec03]	High	No	Yes	14 (smtp, ftp, pop3, http, dns, netbus, bo2K, telnet, finger, imap4, ssh,sun-rpc, sub-7, Generic Trap)	13	Windows NT, 2000, XP		Nothing special	
Decoy Server [Deco02]	High	No	syslog	Unlimited		Windows (9x, 2000, NT) Solaris		Java runtime	Java
DTK [Dtk03]	Low-Medium	Yes	Yes	Unlimited		Unix	Not really		Perl, C
Honeyd [Prov03]	Medium	Yes	Yes	Unlimited	unlimited	Unix	Yes	libdnet libpcap libevent (arpd)	C (scripts shell-perl…)
Labrea [Lab03]	Low-Medium	Yes	syslog	No	none	Win32s, Linux	Not really	libnet libpcap	C
Tiny HoneyPot [THP03]	Medium	No	Yes			Linux		netfilter	Perl
Smoke Detector [Smok03]	High	No	Yes	22 (auth, finger, ftp, http, imap, pop3, printer, rlogin, rsh, smtp, telnet, smb, ssh, echo, changen, domain, tftp, portmap, rpc.lockd, rpc.statd, mountd, nfsd)	9	Windows 2000		Nothing special	-
Bait N Switch Honeypot [BaitSw]	Medium	Yes	No	Switch	Not really	Linux	Yes	Iptables/netfilter Iproute2 Snort 1.9.0	-
KFSensor [KFsens]	Medium	No	No	Unlimited	none	Win32	Yes	http	-
HoneyWeb [HoWeb]	Medium	**Yes**	Yes	1 (web server)	none	Win32, Unix	Not really	Python 1.5 and better Honeyd ?	Python
NetFacade [NetF03]	High	No	Yes	13	8	Solaris	Not really	Nothing special	-
BigEye [BigEye]	Medium	**Yes**	No	2 (ftp,http)		Unix			C

3. Honeynets

3.1 Definition

As discussed in [PoDa03a], there is no commonly agreed definition of the term *honeypot*. To make a long stay short, we can say that, typically, a honeypot is characterized by the fact that its implementation resides on a single machine. This is to be compared with *honeynets*, whose implementation requires a set of machines.

As presented in [Honey1, Honey2], a typical Honeynet consists of multiple honeypot machines and a firewall to limit and log network traffic. An IDS is often used to watch for potential attacks and decode and store network traffic on the system.

By placing a firewall in front of the honeypots, it is possible to control the network flow, the inbound as well as the outbound connections.

Michael Clark is giving in [Clark01] the common elements of a Honeynet:

- A firewall computer which logs all incoming/outcoming connections and sometimes provides NAT service and protection against some Denial of Service attacks;
- An Intrusion Detection computer (IDS). The IDS box can be on the same box as the firewall but it should be on an entirely separate computer that can see all of the network traffic. It also logs all the network traffic and looks for known exploits and attacks;
- A remote syslog computer. The honeypot is slightly modified so that all commands an intruder would issue are sent to syslog. Syslog is configured to send the logs to a remote syslog box;
- The honeypot itself. It can be anything from a default installation to the tools presented before and a mirror of one of the production systems.

This list is not definitive and the *Honeynet* word interpretation can be slightly different. Won-Seok Lee wrote in his course slides that "*Honeynet is nothing more than a high-involvement Honeypot within which risks and vulnerabilities are the same that exist in many organizations today"* [Lee02]. According to his presentation, a honeynet also consists on a network of multiple systems but no further description is given at this point.

However, they all agree that the Honeynet value lies mainly in research and that three requirements should be taken into account: Data Control, Data Capture and Data Collection [Honey1].

- *Data Control*: Once compromised a honeypot cannot be used to harm any non-honeypot system. So the challenge consists in controlling the data flow without the intruders getting suspicious and to give them enough flexibility to execute whatever they need.
- *Data Capture*: The challenge consists in capturing as much data as possible without the intruders noticing they are monitored. The information needs to be stored remotely to guarantee its integrity.
- *Data Collection*: It concerns organizations that have multiple honeynets in distributed environments only. The challenge is to collect all of the captured information securely from several distinct honeynets.

Some architectural Honeynet models have been suggested [Honey2, Honey3]. Those of Lance Spitzner and the Florida Honeynet Project team have received most of the attention and many security groups coming from various universities as well as from the industry are testing them. As a consequence, the *Honeynet Research Alliance* (http://project.honeynet.org/alliance/) has been created. It is a forum dedicated to "share ideas, experiences and findings, helping to develop Honeynet research".

Two models used by this group will be described in the next two chapters. They represent two *Honeynet* generations and they differ mainly in the way they implement the three components mentioned here above.

3.2 First-Generation: GenI Honeynet

The first model which is the older is called GenI Honeynet. 'GenI' stems from the first generation Honeynet where one firewall separates the Honeynet into three different networks, as shown in figure 2.

3.2.1 Data Control

The firewall is the primary tool for data control. It allows any inbound connections but control outbound connections. If a honeypot reaches a certain threshold of outbound connections, the firewall will then block all further attempts. The South Florida Project gives some firewall implementation examples in [Sfp03]:

- CheckPoint Firewall-1 and Shell scripts [ChkPt]
- IPTables with its limit functionality [IPTab]
- OpenBSD's pf with a session-limit pf path [Opbsd]

We observe on figure 1 that the layer-three firewall separates the Honeynet into three different networks: specifically, the Honeynet, the Internet, and the Administrative Network.

A router is used to supplement this filtering. Any packet entering or leaving the Honeynet has to go through both the Firewall and the router. The Firewall is the primary tool for controlling inbound and outbound connections. The router acts as second access control device. It can supplement the firewall; ensuring compromised honeypots are not used to attack systems outside the Honeynet.

The firewall is designed to allow any inbound connections, but control outbound connections. The outbound policies depend on the honeynet administrator choice. Won-Seok Lee suggests for instance in [Lee02] to allow only packets with the IP source address of the Honeynet and to block all ICMP outbound traffic.

The firewall keeps track of how many connections are initiated from a honeypot out to the Internet. Once a honeypot has reached a certain limit of outbound connections, the firewall blocks any more attempts. The South Florida Project *"found five to ten outbound connections per an hour to be a good number to keep blackhat's happy, while protecting others from attacks. This protects the Honeynet from being used as a platform to scan, probe, or attack most other systems."* [Honey1].

The router acts as a second layer of access controls for avoiding the Honeynet to depend on a single source for Data Control. The South Florida Project primarily uses this to protect against spoofed, DoS, or ICMP based attacks. The router allows only packets with the source IP address of the Honeynet to leave the router. This prevents most spoofed based attacks, such as SYN flooding or SMURF attacks.

3.2.2 Data Capture

It can be done directly from the firewall. It logs all connections initiated to and from the Honeynet and sends alerts if necessary. Another tool could be an Intrusion Detection System (IDS) such as Snort, which will alert the administrator of any suspicious activity and give detailed information. In the GenI description whitepaper, Snort captures all network activity to a binary log file [Honey1]. In addition, snort logs all ASCII communication (such as keystrokes from an FTP session) to session breakout files. Both binary and ASCII logs are logged to their own directory for each day. Then, all snort alerts are forwarded to a syslog server by a simple cron script [Honey1].

However, Data Capture can be initiated from the honeypots themselves. Capture keystrokes and screen shots can be made thanks to a modified version of bash for Unix systems and to ComLog for Windows systems. Generally speaking, logs are not kept locally but sent to a remote log server. This exchange must be as secure and discrete as possible [Sys03].

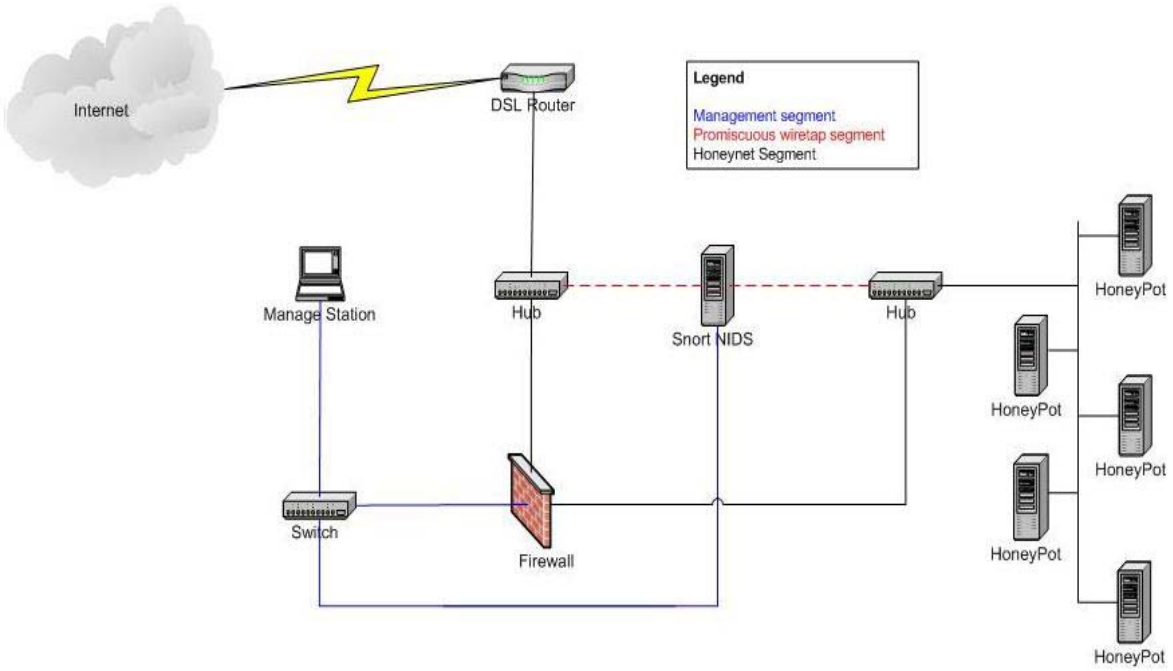

Figure 2: GenI Honeynet example from [GenH03]

3.3 Second-Generation: GenII Honeynet

The second model is called GenII Honeynet. 'GenII' stems from the second generation Honeynet which was suggested in 2002 by the South Florida Honeynet Project as shown in figure 3 [GenH03].

Won-Seok Lee has characterized this new generation as "easier to deploy, yet more difficult to detect" [Lee02]. The main difference with GenI is that it uses a layer2 gateway which acts as a bridge, instead of a router, as shown in figure 3.

3.3.1 Data Control

Described in [GenH03], the GenII Data Control has been designed and developed by members of the Honeynet Project. GenII incorporates firewall and intrusion detection in one system to produce a more stealthy and flexible level of control. Within the Honeynet environment, packets traverse the GenII Data Control system from the Internet to the Honeynet using layer2 frames. The Data Control system sits in line between the Honeynet and the internet watching, capturing and controlling packets as they move along the wire as shown in figure 3.

Unlike GenI Honeynets, GenII Honeynets have "*all requirements combined onto a single device. This means all Data Control, Capture, and Collection happen from a single resource. It makes it easier to both deploy and manage. This single device consists in a layer2 gateway which acts as a bridge*" [GenH03]. This provides several advantages. The fact the device is layer2 makes it more difficult to detect, as it has no IP Stack. There is neither routing of traffic nor any TTL decrement. The device is stealthier as it avoids the bad guys to easily know their traffic is being analyzed and controlled. The second advantage is, as a gateway, all inbound and outbound traffic must go through the device. This means both control and capture of all inbound and outbound traffic can be done from the single device.

The second change is in the way the Honeynet responds to unauthorized activity. Instead of simply blocking connections, it intends to modify or throttle the attacker's activity. The Honeynet Alliance suggests modifying packets as they travel through the layer2 gateway. For example, once an attacker has taken over a system within the Honeynet, they may attempt to launch an FTP exploit against a non-Honeynet system. With GenI technology, the data control is limited. After the fifth attempt outbound (see 3.2.1), all further activity, including any exploits, would be

blocked. However, with the GenII architecture, the exploit attempt would be identified and then modified to make the attack ineffective. The layer2 gateway would modify several bytes within the exploit code, disabling its functionality, and then allow the crippled attack to proceed. The attacker would see the attack launched and packets return, but would not understand why her exploit never worked. The Honeynet also has the ability to fake responses, such as blocking entire connections, but it returns RST packets to the attacker, forging a dropped connection.

The tool which is used in [GenH03] is Snort-Inline, a hybrid version of snort that can drop or modify packets. We note that the Honeynet Project ambition to gather all data Capture, Control and Collection in one single place (the layer-two gateway) is not respected in figure 3. Data Control is done thanks to the *snort-inline* tagged machine, while all data are centralized on the *syslog-ng/mysql* system. Thus, there are some differences between their GenII whitepaper and their own implementation.

3.3.2 Data Capture

The South Florida Project members suggest in [GenH03] to capture data from Kernel space. Indeed GenI operates mainly at the network level (sniffer and firewall information), which might prevent from reading some encrypted data and so gathering precious information.

Some attempts were made by the Honeynet Alliance to obtain data from the honeypots themselves, such as Trojaned versions of /bin/bash, however these solutions have limited capabilities. GenII Honeynet enhances these capabilities by capturing data from kernel space. It ensures that regardless of the communication means, such as SSH, SSL, or IPSEC, this information is still captured.

Another idea they point out is to encapsulate the captured activity in spoofed packets that appear as normal traffic. So attackers do not realize logs are going out of the system. One implementation could be to send naive NETBIOS broadcast timing packets.

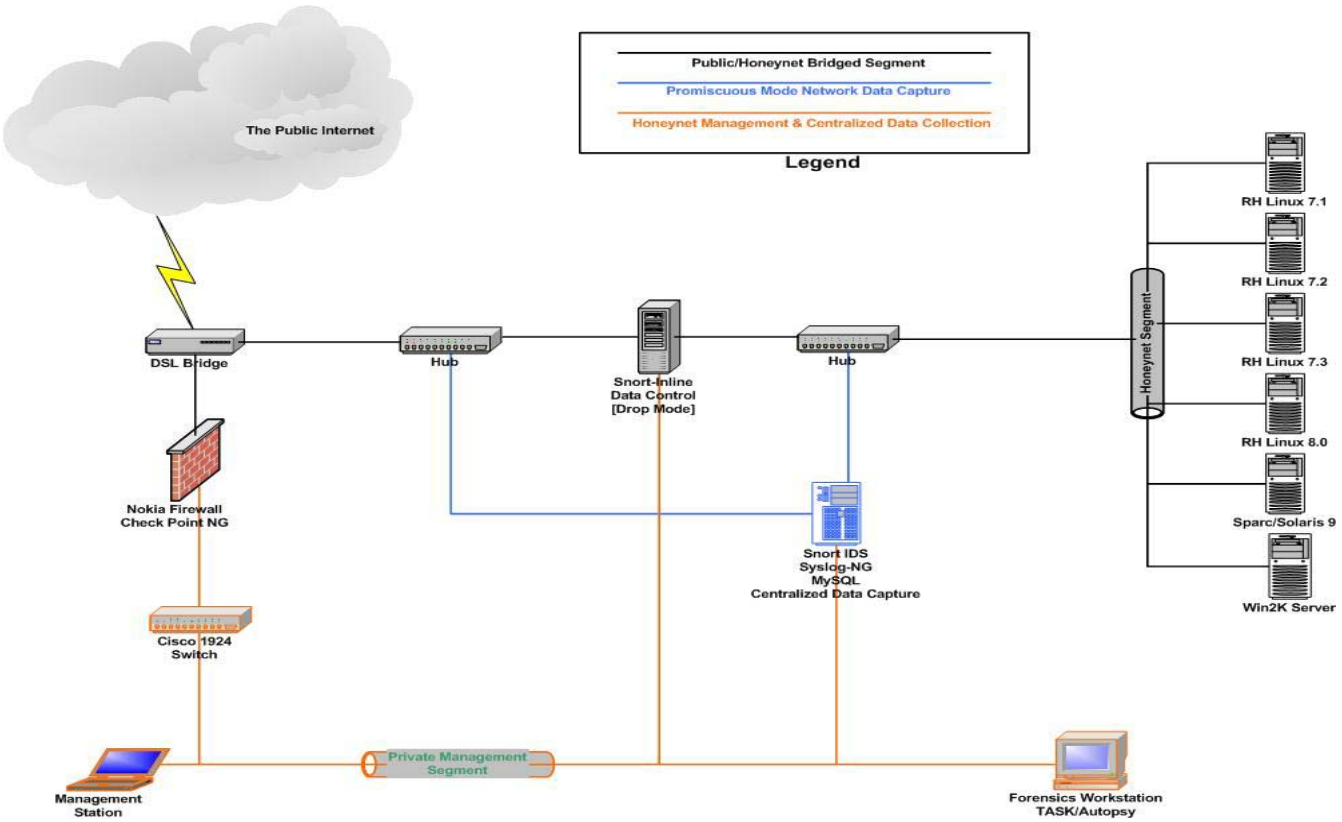

Figure 3: GenII Honeynet example from [GenH03]

So far, we have given a high level overview of the various existing honeypots and honeynets. In the next chapter we intend to provide a more concrete perception of their implementation and use. Consequently, three honeypots are analyzed thoroughly: Deception Toolkit from Fred Cohen, LaBrea Tarpit from Tim Liston and Honeyd from Niels Provos. Their levels of interaction cover the three categories: low, medium and high. Some implementation details are also given to facilitate their installation.

4. Detailed presentation of three specific honeypots

4.1 Deception Toolkit: DTK

4.1.1 Presentation

The Deception Toolkit (DTK) was designed by Fred Cohen in the early 2000. It is based on the *Deception* concept explained in [Coh88] and [PoDa03a]. The basic idea is not new. A system running DTK seems to have a large number of widely known vulnerabilities. The system does not actually have these vulnerabilities, but the attacker cannot discover this from an 'innocent scan'. He must actually try to exercise the vulnerability. This, of course, increases his risk of being detected. Moreover an additional *deception port* (TCP port 365) is opened and acts as a pre-signaling port. DTK's principle is thus to increase the attacker's fear of being detected in order to discourage him from attacking the machine.

4.1.2 Modus Operandi

Written in Perl, DTK uses TCP wrappers to process incoming service requests on ports that would be normally blocked. Subroutines are used to log an attacker's activity and to build appropriate responses to inputs. Responses can be customized to lure the attacker into thinking he has come across a poorly secured and readily exploitable service.
DTK simply listens for inputs and provides responses that seem normal (i.e., full of bugs). In the process, it logs what is being done, provides sensible answers, and lulls the attacker into a false sense of insecurity.

4.1.3 General Remarks

DTK's deception is programmable, but it is typically limited to producing output in response to attacker input in such a way that it simulates the behavior of a system which is vulnerable to the attackers' method.

To be flexible enough, the *deception port* can be changed but IANA has assigned these ports to DTK [34]:

> dtk 365/tcp Deception ToolKit

> dtk 365/udp Deception ToolKit

Finally DTK is not interfering with the normal operations of a system (tcp-wrappers) insofar as deception can be done on the ports that are not used or ports that tcp-wrapper would deny.

As Cohen replied in a FAQ on his webpage, DTK is not the end all to information protection. It is not a strong protection against serious attackers. Today, it is not even very good against experts. But it is a beginning that has some reasonable value. It works, it is reasonably secure, and it definitely increases the uncertainty level for the bad guys. The code running on the deception port is made of only 1-line in C. The whole DTK today is only 100K [Coh88].

4.1.4 Implementation Details

DTK currently uses Perl and C programs. So a C compiler and Perl interpreter must be installed before running DTK.

DTK currently has the following components:

- Generic.pl - a generic interface that works via tcp wrappers to service incoming requests.
- listen.pl - a port listener that listens to a port and forks slave processes to handle each inbound attempt.
- logging.pl - the subroutines and initialization to log what is happening.
- respond.pl - the subroutine for responding based on 'response' file content.
- notify.pl - a sample program to notify administrators of known attacks by email.
- coredump.c - produces a coredump message on a port (what a fakeout).
- deception.c - working on a C version of the program - don't even think about compiling it yet.
- makefile - makes the C programs into executables - truly trivial.

- [nn].response - the responder finiate state machine for each port. This takes some understanding of finite state machines and will be detailed later in this document.
- @[nn].[something] - a response file for non-trivial outputs.
- @fake.passwd - a fake password file that nobody will ever be able to decode.
- expandlog.pl - expand's compressed logfiles into more readable form

DTK has been tested on Linux machines (RedHat 7.3). In this case the "Generic.pl" program is used and it requires TCP wrappers to be in place as well as the InterNET services daemon inetd [Inet02].
Script files are not so well documented and one possible implementation procedure is given in the next paragraphs to facilitate the installation. The distribution file should be copied to a convenient directory (an empty one) and unpacked, unzipped, untared, etc. A precaution should be taken here. A different folder must be used to install DTK. Otherwise it will crash.

Then configuration is done by typing: “Configure”. The working directory ('.'), Perl libraries and the Perl interpreter must be in the path. Defaults can be chosen for most of the entries most of the time. /dtk is assumed here to be the location of the running programs for the rest of this instruction. Configure helps implement the deception by renaming all of the system-dependent entries in the deceptions and the programs so that everything appears to be coming from the system and so that email and other things done by DTK go to the right places.
The relevant lines from the /dtk/dtk.hosts.allow file must then be copied into the /etc/hosts.allow file to implement the desired deceptions from addresses not otherwise authorized to perform the applicable services. The same operation must be done with the appropriate lines from the /dtk/dtk.inetd.conf file. They must be copied into the /etc/inetd.conf file to enable the services that are going to provide deception.
At this point of the installation the command 'kill -HUP' on inetd should not be used.
Now, the appropriate lines from the /dtk/dtk.services file must be added into the /etc/services file to reflect the services which are going to provide deception and to add the now official DTK "deception active" port - 365 to the services file.
Thus, if the telnet service (telnetd daemon) has to be replaced by the fake telnet service of DTK, the following lines should be put:

Into /etc/hosts.allow:

in.telnetd:all:twist /dtk/telnetd –aL/dtk/Telnet.pl %a 80 %u %d testing

Into inetd.conf:

telnet stream tcp nowait root /usr/sbin/tcpd in.telnetd

The process of the inetd daemon must be found ("ps -a | grep inetd"). For recent Unix kernels this might be a problem as inetd is replaced by xinetd.

To conclude a hangup signal must be sent to reload the /etc/inetd.conf file into the memory - "kill -HUP <ID>" where <ID> is the process Id discovered by the ps command previously.

The deception can be tested out by trying something like: "telnet localhost 365". The result should be a slight delay followed by a message indicating that DTK is operating. The same method can be applied to the other services.

In addition, the rc.local file might be modified to include services that do not need to go through TCP wrappers. This will enable them at startup. A typical example is given in /dtk/dtk.rc.local which is to be added (on some Unix systems) to /etc/rc.d/rc.local to be run at startup. To start these services, the commands must be taken as they appear in the /dtk/dtk.rc.local file and run as root.

A commercial DTK GUI is also available on the web site [Dtk03]. It can be found with the White Glove Linux suite produced by Fred Cohen & Associates.

4.2 LaBrea Tarpit

4.2.1 Presentation

LaBrea is a program written in C by Tom Liston that creates a tarpit or, as some have called it, a “sticky honeypot”. It consists in a small, open-source network application that monitors traffic on the local segment. It takes over unused IP addresses on a network and creates ‘virtual machines’ that answer to connection attempts. LaBrea answers those connection attempts in a way that causes the machine at the other end to get ‘stuck’, sometimes for a very long time. Worms trapped in the tarpit are unable to move along to infect other computers. Stuck hackers first waste

their time flailing away at a non-existent machine; they are then forced to shut down their hacking program or computer to escape.
Liston programmed LaBrea in response to Code Red, the worm that has been scouring the Internet since June 1999.
Some of Liston's nasty little visitors have been stuck in his tarpit for over a week [List03].

4.2.2 Modus Operandi

LaBrea works by watching ARP requests and replies. When it sees consecutive ARP requests spaced several seconds apart, without any intervening ARP reply, it assumes that the IP in question is unoccupied. It then creates an ARP reply with a bogus MAC address, and fires it back to the requester.
That way the router associates this MAC address to the IP address and acts accordingly. So LaBrea watches for TCP traffic destined for this MAC address and 'tarpits' the connection attempt: for instance when it sees an inbound TCP SYN packet, it replies with a SYN/ACK.
LaBrea tries to give its 'virtual machines' some character: They can be pinged, they can response to a SYN/ACK with a reset… To be more precise LaBrea answers connection attempts in two different ways that tie up the connecting process: Tarpitting and Persist Trapping [List03].

Tarpitting:
It is the tactic of slowing down a TCP connection when a hostile party's autonomous agent is on the other end. LaBrea completes the connection initialization, tells the connecting machine that it will only accept small (~5 byte) chunks of data, and then ignores any other traffic.
To be more concrete, LaBrea mimics the TCP handshake but dutifully checks and ignore for instance the ACK packet. At that point the requesting computer has committed many other resources to the connection (computer time, memory for counters, buffers, etc) and it starts a countdown clock, waiting for the destination computer (the tarpit honeypot) to start talking. However, LaBrea does not keep track of the connection.
Consequently, the requesting will wait a decent time interval and send a few ACK packets spaced out over time and which are still ignored, and then eventually the requesting computer drops the connection. While the timeout values may vary between Operating Systems, most are measured

in minutes (about 5 to 30 minutes in general), compared to the usual milliseconds for a handshake. Any scanner looking for vulnerable systems will get very slow answers.

Persisting Trapping:
This is an "extra sticky" option that can be chosen. After the handshake the source computer waits for LaBrea station to call back but it never happens as it was said previously. After a certain amount of time has passed, the source computer sends another ACK packet. If LaBrea is set for persistent connections it promptly replies by sending a packet with the TCP RECV window set to 0 byte. LaBrea ACKs the first inbound data packet with a WIN 0 and responds to all following WIN probes with a WIN 0, causing the connecting machine to hang in the "persistent" state. Each time, the source computer thinks the connection is good so it resets its timeout clock and starts waiting again. Indeed, as long as TCP is receiving the proper responses within the times it expects, the connection is never broken. The connection is hold open for an indefinite period of time so that only a process reset at the other end will end it. And LaBrea Tarpit offers to trap connections this way for days, weeks, or according to the LaBrea site, even months [List03].

4.2.3 General Remarks

LaBrea is a very friendly tool that gives a simple overview of what a honeypot looks like. The C files are very well documented and based on them few arrangements can be made. Options are abundant and correctly explained.
This tool was built with the same motivation than another one, called DTK (Deception Tool Kit by F.Cohen). The argument consists in giving a lot of work to the attacker so that she becomes annoyed and decides to give up [MicRi01]. This is typically a production honeypot.
It basically accepts connections to nonexistent IPs, then just sits on them, forcing the remote end to tie up a socket until it times out. Tom Liston thinks that "*this effectively drops a worm's scan rate from a few dozen tries per second per socket, to a few dozen tries per hour per socket. Even if a next-generation worm had the intelligence not to wait on a connection for the full default timeout, they need to wait at least 5 seconds or so to give the other side a fair chance to respond. That would still drop the scan rate by a factor of 10 or more (ratio between the traditional handshake delay and the TCP timeout value)*" [Lab03].

As a bonus, this even provides an easy way of logging and recognizing attempted port-scans.

On the other hand LaBrea can be seen as an insensitive for some malicious people: Security experts doubt that LaBrea will have a big impact on the Internet as a whole. And they argue that LaBrea only gives "antisocial" responses to unsolicited connection attempts [Foc03, Cook02, ScoYu01]. This might be dangerous if this method becomes too widespread by tarpitting on purpose network connections attempts. Tim Liston's answer was that LaBrea is a concept that is on the "benign" end of the spectrum of possible responses and it is still legitimated.

To conclude laBrea is a very interesting tool for people to initiate themselves to honeypots and for people who are looking for very low interaction in order to capture specific activity, such as Worms or scanning activity. Its "antisocial" behavior should be kept in mind while using it.

4.2.4 Implementation Details

LaBrea tarpit requires both libnet and libpcap to compile. Other than that, a "make" with the included Makefile should work. No more configuration procedure is required. It has been tested under Linux Red Hat 7.3. However a new version 2.4 was released on February 10th, 2003 and should enable Labrea to work on all Win32 platforms.

Installation is relatively easy and many options are available:
All the free IP addresses might not be used by LaBrea. To specify those which should not be chosen the /etc/LaBreaExclude file has to be modified. It contains a list of IPs (1/line) to exclude from LaBrea's attention. LaBrea won't do anything to these IPs.
For instance if the NetBEUI ports (137, 138 and 139) and all local addresses have to be excluded from being tarpitted, these lines must be added to the /etc/LaBreaConfig file:

137-139 portignore

<x.x.x.x>-<y.y.y.y> ipignore

where the <x.x.x.x> and <y.y.y.y> are the starting and ending addresses of the local subnet.
IP address can be specified either as single addresses (i.e.: "192.168.0.2") or they can be specified as a range of addresses: (i.e.: "192.168.0.10 - 192.168.0.20"). It works the same way with ports.

Syslog logging can be chosen instead of the standard output (log to the screen). That way logs are stored and can be monitored using *"tail –f <system log file>"*.

However the LaBrea logging may grow by megabytes, which is far too large to examine without filtering. A specific tool called LaBrea Reporter is designed to summarize what happened. It is a script written by Sverre Stoltenberg in the Python open-source language [Pyth02]. It gives summaries of the packet activity, the source hosts in order and the destination hosts in order, along with short comments (ports count. The following picture (fig. 4) is part of one report that has been shortened. The full version can be found on the LaBrea web site [Lab03]. It lists all the source and target addresses, which is important information.

```
LaBrea rapport:
===============

Start date:    Wed Jun 26 00:01:28
End date:      Wed Jun 26 23:55:05

Total transactions: 380634

New source hosts tarpitted this period: 105
               Number of target hosts: 1145
 New tarpitted connections this period: 12695
      Answered SYN/ACK & FIN/ACK scans: 32969       ...

Tarpit Target ports
===================
   21:     2
   22:   996
   25:     6
   80:  2676    ...

Source address
==============
   996 129.81.42.202   Wed Jun 26 01:07:08 - Wed Jun 26 01:08:32 EST
  1134 213.10.150.199  [ipd50a96c7.speed.planet.nl] Wed Jun 26 12:23:27 - Wed Jun 26 12:25:13
EST ...

Target address   Total    Port: Count   Port: Count   Port: Count
==============   =====   ============  ============  ============
 xxx.xxx.xxx.10:    13 |    22:     1     80:     2   1433:     7
xxx.xxx.xxx.101:    26 |    22:     2     80:    17   1433:     7 ...


This report is created with LaBrea-file.py, a variation
of LaBrea-stats.py. The latest version can be found at
http://people.opera.com/sverrest/LaBrea/

Information abount what LaBrea is can be found at
http://www.threenorth.com/LaBrea/
```

figure 4: LaBrea Report from [Lab03]

Finally a simple executable file (LaBrea@Home) originally built against CodeRed and Nimda worms is also available. It can send specially crafted packets to the worm. The other end is lured into thinking it has a genuine connection on port 80 and then prepares to send its payload. But LaBrea@Home will then instruct the other end to wait by setting what is known as the TCP "window" to zero and replying the same way each and every time the other end attempts to send information. The other end - the scanner or worm - will then be held up forever, or until LaBrea@Home releases it [Lab03, MicBiz02].

(More information is available on the web page: http://www.hackbusters.net/LaBrea/)

4.3 Honeyd

4.3.1 Presentation

Honeyd, created by Niels Provos in 2002, is an extremely powerful, open source honeypot. It is designed to run on the Unix system and it has the ability to emulate over 400 different operating systems and thousands of different computers. Like Specter, Honeyd emulates operating systems at the application level stack but it also emulates operating systems at the IP level stack. As mentioned before, Honeyd is an open source solution which is free to use and the number of fake services will grow as members of the security community develop and contribute code.

4.3.2 Modus Operandi

Honeyd introduces several new concepts to honeypots. First, it does not detect attacks against its own IP address, as BOF and Specter do. Instead, Honeyd assumes the identity of any unused IP address. The goal is to forward the traffic of all non-existent systems to the Honeyd honeypot.
Indeed Honeyd is receiving attacks by implementing ARP Spoofing [Arps98]. This layer2-based method binds an IP address of the intended victim (one which is currently not in use) to the MAC address of the honeypot. This way all systems on the network (including routers) send IP packets of non-existent system to the Honeyd honeypot.

This method is not intrinsically coded within Honeyd insofar as two approaches can be applied here:

- Honeyd can depend on another program called Arpd [Arpd] that will help to do that. Arpd, developed by Dug Song, identifies non existent systems and then forwards any connections to them to the Honeyd honeypot. The IP address of the non existent system is bound to the MAC address of the Honeyd honeypot. Arpd process confirms periodically that the IP is not in use by sending ARP requests. And since this spoofing happens at layer 2, it works in switched environments just as well as in hubbed environments.
- An alternative to Arpd is ARP Proxy [Arpp03]. It is working quite the same but the ARP entries are statically introduced. Non existent IP addresses can be statically bound to Honeyd's MAC address. Most versions of Unix allow a system to assign and broadcast such ARP assignments. The *arp –s* command on the honeypot is used to accomplish this. The *–s* parameter statically assigns the MAC address to an IP.

For instance if the following addresses 192.168.0.201, 192.168.0.202 and 192.168.0.203 are destined to be used by Honeyd, Arpd will automatically check that they are not used before ARP spoofing. On the contrary, with ARP Proxy ARP spoofing will be activated by means of static entries such as:

arp -s 192.168.1.201 <honeypot MAC address> permanent pub
arp -s 192.168.1.202 <honeypot MAC address> permanent pub
arp -s 192.168.1.203 <honeypot MAC address> permanent pub

The result from the two previous approaches will be the ARP Table being updated. So when an attacker attempts to connect to a system that does not exist, Honeyd receives the connection attempt, assumes the identity of the non-existent system and then replies to the attacker.

Something interesting is that Honeyd can emulate different operating systems at the same time. In comparison, Specter can emulate more than 13 different operating systems but it can only emulate one system at one time [Spec03]. Honeyd can emulate many different systems at the same time. It takes the very same database of signatures that Nmap uses and replies to Nmap

probes based on the emulated operating system [Fyo98, Toor01]. This can be seen as responses in order to lure the OS fingerprinting of Nmap-based tools.

Table 2 gives an example of the Nmap signature database. So if Linux 2.2.14 is emulated, Honeyd will use the given test descriptions to fill the packets coming from the virtual machine running Linux 2.2.14. From the outside an Nmap OS fingerprinting attempt will only reveal that this virtual machine is effectively running on Linux 2.2.14.

Table 2: Nmap Fingerprinting database

```
# This collection of fingerprint data is (C) 1998,1999 by
# Fyodor (fyodor@dhp.com, fyodor@insecure.org ).
# The usage license for this file is the same as that for which
# you acquired nmap (probably the GNU General Public License)

# TEST DESCRIPTION:
# Tseq is the TCP sequence ability test
# T1 is a SYN packet with a bunch of TCP options to open port
# T2 is a NULL packet w/options to open port
# T3 is a SYN|FIN|URG|PSH packet w/options to open port
# T4 is an ACK to open port w/options
# T5 is a SYN to closed port w/options
# T6 is an ACK to closed port w/options
# T7 is a FIN|PSH|URG to a closed port w/options
# PU is a UDP packet to a closed port


# Contributed by mouse-aj3d@datastacks.com, Samuel Knapp,
madranis@madranis.com
Fingerprint Linux 2.2.14
TSeq(Class=RI%gcd=<6%SI=<2DD9C88&>755F7)
T1(DF=Y%W=7C38|7F53%ACK=S++%Flags=AS%Ops=MENNTNW)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=7C38|7F53%ACK=S++%Flags=AS%Ops=MENNTNW)
T4(DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(DF=N%W=0%ACK=S%Flags=AR%Ops=)
PU(DF=N%TOS=C0%IPLEN=178%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=F|E)

(...)
```

Furthermore, Honeyd may be configured to interact with the attacker using an emulated service. This is done by executing one specific script.

Each unique script determines the interaction with the attacker. Also, each unique script determines the OS type of the application, so different scripts will be used depending not only on the service type but the OS type as well. They can be developed with any scripting or coding language: Shell Code, Perl… Some are already available on the web (google search: Honeyd <service_name> script).

4.3.3 General Remarks

Honeyd has many advantages including the fact that it is a free and open-source tool.

One application: *blackholing*. This enlightens a concept first demonstrated by the Cooperative Association for Internet Data Analysis (CAIDA). Without exploiting any honeypot but thanks to their own tools, they conducted an analysis of source-spoofed distributed denial-of-service attacks based on return connections for an entire /8 network [Caid03]. This can also be applied to monitoring the level of 'noise' on the Internet, including worms, exploit tools, and automated attacks.

Honeyd offers an interesting way to apply this method. Instead of monitoring a single IP address, entire networks with non-existent systems can be identified. Then all traffic from that network could be routed to Honeyd honeypot. The intent of blackholing is not to identify a single attack but to identify trends.

Configuration can be very granular: Honeyd may be used to create a virtual honeynet or for general network monitoring. It supports the creation of a virtual network topology including dedicated routes and routers. The routes can be attributed with latency and packet loss to make the topology seem more realistic and to defeat the attacker's attempt to understand the network topology.

Currently, this ability to interact with different attackers is limited to TCP services, ICMP requests and ICMP replies. Currently all UDP ports are assumed to be either blocked or proxied

or opened as it is specified in the *Implementation Details* part. But no script is designed on UDP protocol.

As Honeyd OS emulation is based on Nmap signatures, a Nmap file can be found in the Honeyd source package with more than 473 different operating systems fingerprinted. This is where Honeyd gets its limitation of emulating 473 different operating systems. This method is not really foolproof. Nmap is but one of many ways to determine the OS type (Ofir Arkin's Xprobe tool, passive OS fingerprinting) [Ark1, Ark2, ArkYa02]. This makes Honeyd detectable.

Finally a special attention can be paid to new systems that are going to be introduced in the network. Valid systems will start using the IP addresses for which ARP entries were added (via Arpd or ARP Proxy), this will cause conflicts on the local network.

As it is typical of most low-interaction honeypots, Honeyd introduces limited risk to an organization. The honeypot is not designed to provide a complete operating system to attackers; instead attackers are limited to the functionality emulated by the scripts.

4.3.4 Implementation Details

Installation is not commented yet but it is not complicated insofar as *nix basis are acquired.
In order to compile Honeyd, you need the following libraries:

libevent - an asynchronous event library.

libdnet - the [not so] dumb network library.

libpcap - a packet capture library.

One more tool is eventually required: Arpd (or proxy arp as described before).
Then install commands are:

> *./configure*

> *make*

> *make install*

One example of Honeyd configuration file is given in table 3:

Table 3 : Honeyd configuration example

```
## Honeyd configuration file ##
### Windows computers (default)
create default
set default personality "Windows NT 4.0 Server SP5-SP6"
set default default tcp action reset
add default tcp port 110 "sh scripts/pop.sh"
add default tcp port 80 "perl scripts/iis-0.95/main.pl"
add default tcp port 25 block
add default tcp port 21 "sh scripts/ftp.sh"
add default tcp port 22 proxy $ipsrc:22
add default udp port 139 drop
set default uptime 3284460
### Cisco router
create router
set router personality "Cisco 4500-M running IOS 11.3(6) IP Plus"
add router tcp port 23 "/usr/bin/perl scripts/router-telnet.pl"
set router default tcp action reset
set router uid 32767 gid 32767
set router uptime 1327650
# Bind specific templates to specific IP address
# If not bound, default to Windows template
bind 192.168.1.150 router
```

Different types of computers are created. Honeyd calls them templates. These templates define the behavior of each emulated operating system. In this configuration file, two different emulated computers have been created: *default* and *router*. For each template, one defines its "personality": the operating system to be emulated at the IP level. Personality names are defined based on those used in the Nmap fingerprint database. The personality does not affect the behavior of the emulated services; it only modifies the behavior of the IP stack. For the emulated services, different scripts can be selected based on what type of OS you want to emulate. In other words, if the personality is Windows, nothing forbids you from choosing to emulate an Apache server on the HTTP port. A script emulating an IIS server should be used instead. However this would likely be suspicious to an attacker.

The next step is to define the behavior of each port. Specific ports can be assigned specific behavior, or general behavior. For example, in the template *default* all the TCP ports are assigned the *reset* behavior: they respond with a RST to any connection attempts. Other options are *open*

(will respond with ACK) or *block* (will not respond). A fourth option is the use of scripts to emulate services. In the case of the template *default* scripts are bound to the ports 21, 80, and 110. These are the actual scripts that are executed and interact with the attackers. The option to proxy connection attempts to other systems is also available. In the *default* template, all SSH connections are proxied back to the attacker. There are several other more advance features of Honeyd, such as creating virtual, routed networks and spoofed timestamps, but a detailed explanation is beyond the scope of this paper.

Once templates are created, IP addresses must be bound to one of them. In the example given in table 3, IP address 192.168.1.150 is bound to template *router*. In this case, if anyone attempts to connect to IP address 192.168.1.150, they will be interacting with the Honeyd honeypot using the *router* template. The *default* template is a key template to Honeyd. The template with the name *default* becomes the default for all other connections to non-used IP space. So if any connections in the example are made to any unused IP space in the 192.168.1.0/24 network, they will get a Windows box emulated by Honeyd, except for the IP 192.168.1.150, for which they will get the Cisco router.

When the configuration is correct Honeyd can be launched. Default command series (with root acess) might be:

> *arpd –i eth0 193.55.112.50/24*
> *Honeyd –d –p nmap.prints –f /etc/Honeyd/Honeyd.conf 193.55.112.50/24*

In this case, we use Arpd instead of Arp Proxy. The arguments are the network interface to communicate with and the range of IP addresses that will be checked by Arpd daemon in order to find non-used addresses.
‘-d’ is optional. It prints directly logs on the screen. However ‘-p’ and ‘-f’ are mandatory. They give the access path to the OS fingerprint file of Nmap and Honeyd configuration file respectively.
The address class specified to Arpd and Honeyd should be identical or Honeyd sub-class should be at least included in Arpd larger one.

We refer the interested reader to http://www.citi.umich.edu/u/provos/Honeyd/ for more detailed information.

A Honeyd-win32 version has been released in March 2003 [Dav03]. Thanks to the efforts of Michael A. Davis, it has all the capabilities of the Unix version.

The installation is quite simple. It requires Winpcap Developer Pack, as well as libdnet-msvc and libevent-win32 libraries [CDref, Winpcap]. All of them must be extracted with Honeyd-WIN32 source file into a common directory. Then the Honeyd.dsw project has to be loaded into MS VC++ 6 (there is currently no support for MS VC.NET). Finally the Platform SDK must be added to the directory search path (Tools->Options->Directories). And the Honeyd.exe executable can be built.

5. Conclusion

In this document, we have offered a survey of the various honeypots implementations that existed as of June 2003. A more detailed presentation of the following tools has been proposed: DTK, LaBrea and Honeyd. Each one is associated to specific usages: LaBrea intends to “tarpit” attackers while DTK hopes to deceive them into making them think they are observed. This highlights the need of a careful study of the means and goals to achieve, before choosing any particular honeypot solution. This step is all the more mandatory that many tools appear every month in the Internet, driving by different motivations. The default choice consists in implementing a honeypot that is more general and highly configurable, such as Honeyd.

6. Bibliography

[Ark1] O. Arkin. XProbe tool home page: http://www.sys-security.com/

[Ark2] *XPRobe2* home page: http://www.xprobe2.org/

[ArkYa02] O. Arkin, F. Yarochkin, « *Xprobe v2.0: A fuzzy Approach to Remote Active Operating System Fingerprinting*". August 2002. Available on line: http://www.xprobe2.org/

[Arpd] Arpd RPM packages for Linux Red Hat, available at: http://rpmfind.net/linux/RPM/cooker/contrib/alpha/arpd-0.2-1mdk.alpha.html

[Arpp03] ARP Proxy presentation by Freenix. Available on line at: http://www.freenix.fr/unix/linux/HOWTO/mini/Proxy-ARP.html

[Arps98] *ARP Spoofing*, Vergenet Presentation, April 98, available on line at: http://www.vergenet.net/linux/redundant_linux_paper/talk/html/node3.html

[BaitSw] *Bait N Switch Honeypot* from Team Violating home page: http://violating.us/projects/baitnswitch/

[BigEye] *BigEye* home page: http://violating.us/project/bigeye/

[Bof03] *Back Officer Friendly BOF* (NFR Security) home page: http://www.nfr.com/resource/backOfficer.php

[Caid03] Cooperative Association for Internet Data Analysis, CAIDA home page: http://www.caida.org

[CDref] F. Pouget, M. Dacier. *Research Report CD*. Institut Eurecom. Ref CD-RR-03-089. Sept. 2003.

[ChkPt] *CheckPoint Firewall*. CheckPoint Software Technologies Ltd. Product. Home page: http://www.spiderneteurope.com/chekpoint.htm

[Clark01] M. Clark, "*Virtual Honeynets*", SecurityFocus. November 2001, available on line: http://online.securityfocus.com/infocus/1506

[Coh88] F. Cohen, "*Deception and Perception management in Cyber-Terrorism*". 1988

[Coh99] F. Cohen, "*A Mathematical Structure of Simple Defensive Network Deceptions*", 1999.

[Cook02] C. S. Cook, "*Tarpits for an Imperfect World*", July 2002.

[Dav03] *Honeyd-Win32* README file at : http://www.securityprofiling.com/Honeyd/WIN32_README.txt

[Deco02] *Symantec Decoy Server* page: http://enterprisesecurity.symantec.com/products.cfm?ProductID=157

[Dtk03] *Deception Toolkit* home page: http://www.all.net.dtk/

[Ether02] *Ethereal Network Protocol Analyzer* version 0.9.3. Network Associates, Inc. July 2002. Available on line: http://www.ethereal.com

[Foc03] *Security Focus mailing lists* available at: http://www.securityfocus.com/archive

[Fyo98] Fyodor, "*Remote OS Detection via TCP/IP Stack Fingerprinting*", October 1998. Available on line: http://www.nmap.org/nmap/nmap-fingerprinting-article.html

[GenH03] "*Know Your Enemy: GenII Honeynets Easier to deploy, harder to detect, safer to maintain*", by the honeynet Project members, June 2003. Available on line: http://project.honeynet.org/papers/gen2/

[HonWeb] Honeypots web site: http://www.honeypots.net

[Hone02] "*Know Your Enemy: Learning with User-Mode Linux Building Virutal Honeynets using UML*", Honeynet Project, December 2002. Available on line: http://www.honeynet.org/papers/uml/

[Honey1] Honeynet Project, "*Know Your Enemy: Defining Virtual Honeynets*". Sep. 2002

[Honey2] Honeynet Project, "*Know Your Enemy: Part I*". 2001. Available on line at: http://project.honeynet.org/papers/index.html

[Honey3] Honeynet Project, "*Know Your Enemy: Part II*". 2001. Available on line at: http://project.honeynet.org/papers/index.html

[Honey4] Honeynet Project, "*Know Your Enemy: Motives*". 2002. Available on line at: http://project.honeynet.org/papers/index.html

[Honey5] Honeynet Project, "*Know Your Enemy: A Forensic Analysis*". 2002

[HoneyW] *Honeyd* from N. Provos home page: http://www.citi.umich.edu/u/provos/Honeyd/

[HoWeb] *HoneyWeb* download page: http://www.var-log.com/files/

[Inet02] *Internet Service Daemon* Inetd home page: http://www.xinetd.org

[IpTab] *The IPTables/netfilter project* home page: http://www.netfilter.org/

[Incid02] Incidents Mailing List archives available on line: http://www.incidents.org/archives/intrusions/msg03763.html

[KFsens] *KFSensor*, by KeyFocus, home apge: http://www.keyfocus.net/kfsensor/

[Lab03] LaBrea-The Tarpit home page: http://hackbusters.net/LaBrea.May2002

[Lee02] Won-Seok Lee, "*Honeypots*". Ajou University Info. Comm. & Security lab.

[Libevt] Libevent-win32, lidnet-msvc and winpcap Librairies available on line: http://securityprofiling.com

[Lin03] Linux documentation on Netfilter and IPTables available on line at: http://www.linux.org/docs/

[List03] "LaBrea::Tarpit HELD SINCE" home page: http://www.hackbusters.net/cgi-bin/guests.cgi?captured

[Mant03] "*Intrusion Detection Systems: Symantec™ Mantrap ™*" technical paper available at http://enterprisesecurity.symantec.com/

[MicBiz02] R. Michael and Bizsystems, "*LaBrea::tarpit*" Perl Module, release 1.03. June 2002. Available on line: http://scans.bizsystems.net

[MicRi01] J.B. Michael, R.D. Riehle, "*Intelligent Software Decoys*". California US. 2001

[NetF03] NetFacade Intrusion Detection Service, by Verizon, home page: http://www22.verizon.com/fns/netsec/fns_netsecurity_netfacade.html

[Opbsd] *The OpenBSD Project* home page: http://www.openbsd.org/

[PoDa03a] F. Pouget, M. Dacier, *"Honeypot, Honeynet, Honeytoken: Terminological Issues"*. Eurecom Research Report RR-03-081. August 2003.

[PoSt03] D. Powell, R. Stroud, *"Conceptual Model and Architecture of MAFTIA"*. MAFTIA Project (IST-1999-11583), Deliverable D21, January 2003, available on line at: http://www.maftia.org

[Prov02] N. Provos, "*Honeyd: A Virtual Honeypot Daemon (Extended Abstract)*", University of Michigan. US.

[Prov03] *Honeyd* home page: Niels Provos, http://www.citi.umich.edu/u/provos/honeyd/

[Pyth02] Python Scropting Language. Release 2.2 Python Organization. Available on line: http://www.python.org

[ScoYu01] B. Scotteberg, W. Yurcik, D. Doss, "*Internet Honeypots: Protection or Entrapment?*". US University of Illinois. 2002

[Seif02] K. Seifried, "*Honeypotting with VMware – basics*". www.seifried.org/security/ids/20020107-honeypot-vmware-basics.html

[Sfp03] South Florida Honeypot project. Home page: http://www.floridahoneynet.org/

[Smok03] *Smoke Detector*, product home page: http://palisadesys.com/products/smokedetector/index.shtml

[Snort03] *Snort Intrusion Detection System* home page: http://www.snort.org/

[Spec03] *Specter* home page: http://www.neoworx.com/products/specter/

[Spit02] L. Spitzner, "*Honeypots: Tracking Hackers*", , Addislon-Wesley, ISBN from-321-10895-7, 2002.

[Spit03] L. Spitzner, "*Honeytokens: The Other Honeypot*", 2003. www.securityfocus.com/infocus/1713

[Stol03] S. Stoltenberg, *"LaBrea Reporter script Release 1.10*". June 2002. Available on line: http://people.opera.com/sverrest/LaBrea/

[Sys03] M. Bishop, *"Computer Security: Art and Science"*. Addison-Wesley book published in 2003.

[THP03] *Tiny Honeypot* home page: http://www.alpinista.org/thp/

[Toor01] "*The Science of OS Fingerprinting*", Computer Security Conference Toorcon 2001. Available on line: http://toorcon.org/2001/lineup/osfinger.ppt

[Usm03] *User-Mode-Linux* home page: http://user-mode-linux.sourceforge.net

[Winpcap] *Winpcap version 3.0 beta* home page: http://winpcap.polito.it/