Honey, I caught a worm
Building yourself a honeypot, some practical issues

Arthur Donkers
arthur@reseau.nl
The latest version of this paper is available via http://www.reseau.nl/

Introduction
CodeRed and Blue, Nimda I and II, Linux BIND worms, they are now all part of Internet history and folklore. At the time of their breakout however, these worms created a lot of problems, chaos and headaches for system administrators, network administrators and security people, and we all hope everybody has learned from these happenings.

Chances are that such an outbreak can happen again due to a great many factors, amongst which the complexity of the software used nowadays, the inevitable bugs therein, the time constraints of administrators and a lack of information on security issues. So to prevent the same problems, chaos and headaches from happening again a number of things can and must be done. One of the more fun things one can do to keep up to date with current security affairs, is to build a honeypot. The excellent Honeynet project (http://project.honeynet.org/) is a great resource for people wanting to build a honeypot.

This paper describes the practical steps the author has taken to build a honeypot, catch worms and other exploits and what to do afterwards. And of course, there are a number of points for improvement (aren't there always ?).

What can you use a honeypot for (and what not !) ?
Different people have different views about the use of a honeypot. Some say it is a way to catch a hacker, or keep her busy so she does not hack your real server. Both of these probably apply to script kiddies, since these are of the point-and-click generation and do not care what system they hack. The hackers you really want to catch and keep from hacking your real server are not fooled by a honeypot. Even worse, these real hackers can sometimes use the honeypot as a stepping stone into the rest of your network.

Others say it is an advanced form of an Intrusion Detection System where you get a real intrusion to detect. You 'lure' a hacker into your intrusion detection system. Another application, related to Intrusion Detection, is an early warning system. Once your network gets attacked, chances are your honeypot will be included in the mix as well. So once your honeypot is triggered, you know you are under attack, or that an attack is at hand. Although both of these are valid applications of a honeypot, they put a lot of stress on your organization. It requires you to permanently monitor the honeypot and take appropriate action once some alarm is tripped.

This leaves us with the most convenient use of a honeypot as an educational and research tool. It provides an insight into the current state of affairs, provides you with information about new exploits (in some cases even the exploits themselves), rootkits and worms. Using this information you can take appropriate measures to protect your real production systems. And when you gather metrics and statistics about your honeypot you can even use it to 'sell' security to the upper management. You can provide them with factual data about threats, portscans and breakin attempts on your network.

Honeypots do's and don'ts

To make sure your honeypot does not pose any risk for the rest of your network, your company or the Internet in general you have to pay attention to a number of issues.

Never, ever use a honeypot in a production environment. It must be an isolated machine, dedicated to being a honeypot and nothing else. It should be connected to the network you would like to monitor (for instance your DMZ) and all traffic to and from the honeypot must be considered malicious. This also means that management of the honeypot platform, the system your honeypot is running on, should be done out of band, for instance via the console, a serial connection or a separate network interface (not the most secure way but often the most convenient). Everything else must be absolutely safe and secure, and you must make sure you can keep it like that over the lifetime of the honeypot.

You must assume that your honeypot will be compromised in the near future, so you must make sure it does not contain any real production accounts, passwords and data. Once a hacker has compromised your honeypot, she will scan the machine for interesting data. It would be a disaster if an account on the honeypot has the same password as its counterpart in the production network and the hacker can use this to break into the rest of your network. Using an old production database in your fake web application on your honeypot is therefore also a very bad idea™.

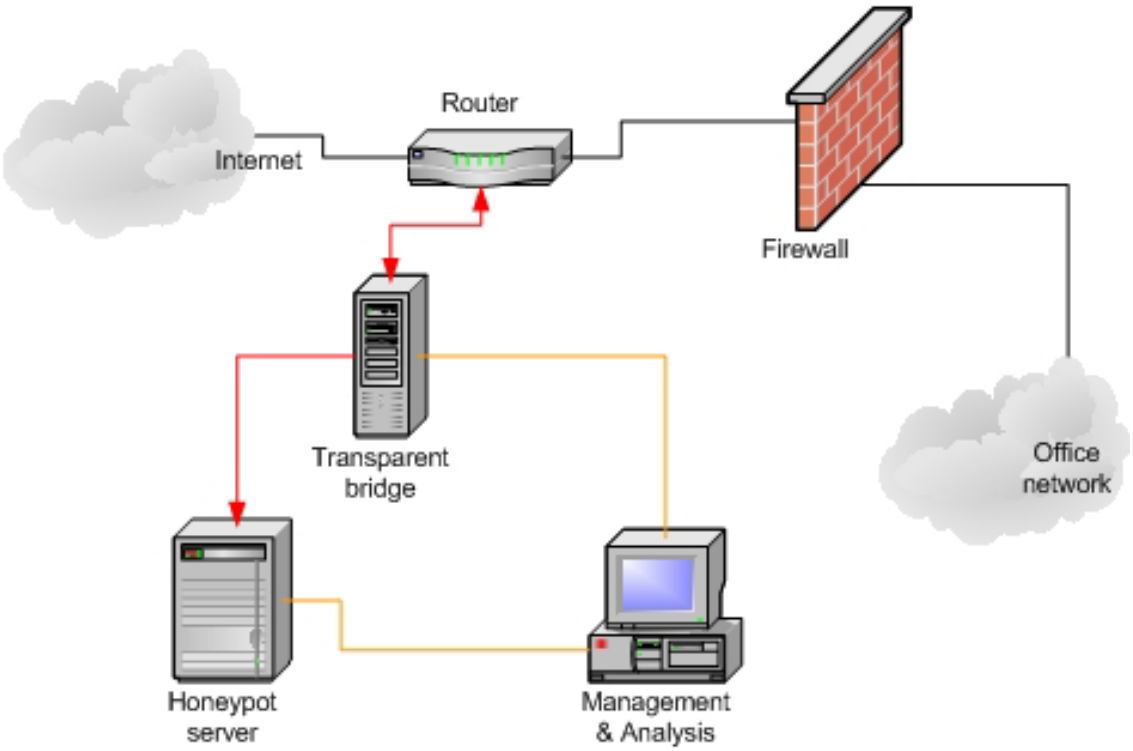
The hacker you really would like to monitor are most often very suspicious (they also keep abreast with developments of honeypots and the like). So to not raise their suspicions you should not make your honeypot unnecessarily vulnerable. Best thing to do is to make it a default installation of an operating system, perhaps with some very obvious patches installed. This makes it look like a poorly managed machine that someone has forgotten to upgrade or fix. This still makes it possible to break into the machine but raises the level somewhat above the capabilities of most script kiddies. For educational purposes you can also make the honeypot as secure as possible, by installing all available patches and disabling all unnecessary services. This raises the bar considerably and keeps all script kiddies away. Only the most determined hacker using the latest exploits will break into your honeypot. This way you are provided with new exploits and methods, without all the other noise. However in this case your honeypot may be compromised only once or twice a year in which case it may go unnoticed for a long time and you loose the security advantage of being early informed.

Since all traffic to and from the honeypot is potentially dangerous and suspicious, it should all be logged, on a different machine (more on that later). These logs should be analyzed once a break in has occurred to be able to determine what happened. This kind of logging only applies to traffic that is not encrypted, so rogue SSH or SSL sessions can not be monitored in this way. Also home brew crypto traffic can not be monitored (for instance netcat with aes, as can be found at http://mixtersecurity.de/progs.html).

And last but not least, you must make sure that once your honeypot has been compromised, it can not be used to scan and hack into other machines. Therefore you need to build some sort of network diode into your honeypot You would like to allow all traffic in, but severely restrict outgoing traffic. You must not allow for portscans to be performed, or 'dangerous' services to be used. However you should allow a hacker to use http and ftp so they can download their tools (and you can obtain a copy).

Building yourself a honeypot
If you, all things considered, still want to build yourself a honeypot, you can use the cookbook below to build a controllable honeypot. The author has built a honeypot using these techniques and it has proven to be a manageable honeypot with reasonable success.

The network layout for the honeypot system is like this:



The actual honeypot consists of the following machines:

- Transparent bridge
- Honeypot server
- Management and analysis system

Each of these machines plays a crucial role in the security and reliability of the honeypot.

Transparent bridge
The transparent bridge has a very important task to perform. It controls all traffic to and from the honeypot. In fact, it is the only real protection you have between your honeypot and your network, and the Internet in general.

Normally you would use a packet filter or firewall for this purpose. However, you do not want a hacker to know you are filtering traffic to or from the honeypot. A normal firewall or packet filter can easily be detected by using a number of different ways:

- traceroute (classic or with `hping2`). Most firewalls and packetfilters have IP addresses on their interfaces and are therefore a hop between the honeypot server and the rest of the Internet. They can therefore show up on the output of the traceroute command.

- TTL analysis. Although not very likely, they may determine from TTL values that there is one more hop between the honeypot and their system than expected.

- OS fingerprinting (using `nmap` or `queso`). By scanning the honeypot hackers will try to determine the operating system running on the honeypot. They can do this using different tools available on the Internet, but when they try to do it on your honeypot chances are they will detect the OS of the firewall/packet filter instead or get some indeterminate result.

Furthermore, since the firewall has at least one IP address, it becomes vulnerable to attack itself. If for some reason the firewall or packetfilter itself is compromised, due to a misconfiguration or newly discovered vulnerability (can you say SNMP ?), you loose all that protects you and the rest of the Internet from your honeypot.

So this system needs to be invisible from the network. This can easily be accomplished by using a standard PC hardware platform and OpenBSD. Mind you, transparent bridging can also be accomplished with Linux and bridging software, but OpenBSD was more convenient at the time.

For this purpose the author has used a simple PC machine (Pentium 200MMX, 64 Mb RAM and 4 Gb disk) with 3 ethernet cards to build a transparent bridge. Installation is easy and can be done in a few simple steps.

First of all a default installation of OpenBSD is done (in this case OpenBSD 2.8). This default installation has all the necessary tools on board to make this machine a bridge. Once the software has been installed two of the network cards are dedicated to the bridge. The third card is used to connect to the management station.

Bridging two ethernet cards is done via a number of configuration files on the system:

In this case the two ethernet cards used to build the bridge are `ep0` and `ep1`. For each of these files a `hostname.<interface>` file is created like this:

```
inet media 10BaseT
up
```

This tells OpenBSD that this interface is active but has no IP address. The management interface has been given an IP address and is configured like this:

```
inet 172.16.1.29 255.255.255.0 172.16.1.255
```

Both the ep0 and ep1 interface are then combined into a bridged set using the `bridgename.bridge0` file like this:

```
add ep0 add ep1 up
```

This is all there is to it. The `ifconfig` -a output will look like this:

```
lo0: flags=8009<UP,LOOPBACK,MULTICAST> mtu 32972
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x5
        inet6 ::1 prefixlen 128
        inet 127.0.0.1 netmask 0xff000000
lo1: flags=8008<LOOPBACK,MULTICAST> mtu 32972
xl0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        media: Ethernet autoselect (100baseTX full-duplex)
        status: active
        inet 172.16.1.29 netmask 0xffffff00 broadcast 172.16.1.255
        inet6 fe80::260:97ff:fe0c:3633%xl0 prefixlen 64 scopeid 0x1
ep0:
flags=8963<UP,BROADCAST,NOTRAILERS,RUNNING,PROMISC,SIMPLEX,MULTICAST>
mtu 1500
        media: Ethernet 10baseT
        inet6 fe80::260:97ff:fe7e:62d0%ep0 prefixlen 64 scopeid 0x2
ep1:
flags=8963<UP,BROADCAST,NOTRAILERS,RUNNING,PROMISC,SIMPLEX,MULTICAST>
mtu 1500
        media: Ethernet 10baseT
        inet6 fe80::2a0:24ff:fe8c:44%ep1 prefixlen 64 scopeid 0x3
```

However, this bridge will happily bridge all traffic on the network, and this is not quite what you want. A nice feature of OpenBSD is that you can apply IP filters to bridged interfaces as well. So using the `ipf` filter on both `ep0` and `ep1` will allow you to

regulate traffic in and out of your honeypot, completely transparent to the hacker. The bridge will even allow for a proper OS fingerprinting of the honeypot itself.
Mind you, since the ep0 and ep1 interface do not have an IP address, you cannot use NAT (Native Address Translation) or QoS software like. (not that you would like to use NAT anyway, but still).

So filtering the traffic in and out of the honeypot is done with ipf, just like for a normal firewall. When reading these rulesets keep in mind that the honeypot(s) use IP address `193.78.174.86` and `193.78.174.87`. The management of the honeypot is done via a private space address in the `172.16.1.0/24` network.

```
#       $OpenBSD: ipf.rules,v 1.6 1997/11/04 08:39:32 deraadt Exp $
#
# IP filtering rules.  See the ipf(5) man page for more
# information on the format of this file, and /usr/share/ipf
# for example configuration files.
#
# Pass all packets by default.
# edit the ipfilter= line in /etc/rc.conf to enable IP filtering
#
#pass in from any to any
#pass out from any to any

# Allow traffic to and from lo0
pass in quick on lo0 all
pass out quick on lo0 all

# Allow traffic to port 22 on xl0
pass in quick on xl0 proto tcp from 172.16.1.1 to 172.16.1.29 port =
22 keep state
pass in quick on xl0 proto tcp from 172.16.1.10 to 172.16.1.29 port =
22 keep state
pass in quick on xl0 proto tcp from 172.16.1.11 to 172.16.1.29 port =
22 keep state
# Allow traffic to port 25 on mail server
pass out quick on xl0 proto tcp from 172.16.1.29 to 172.16.1.1 port =
22 keep state
pass in quick on xl0 proto icmp from 172.16.1.1 to 172.16.1.29 keep
state
block in quick on xl0

# Block all ip options, short packets and fragmented packets
block in quick all with ipopts
block in quick all with short
block in quick all with frag

# Statefully pass all input traffic from Internet to the honeypots
pass in quick on ep0 proto tcp from any to 193.78.174.87 keep state
pass in quick on ep0 proto udp from any to 193.78.174.87 keep state
pass in quick on ep0 proto icmp from any to 193.78.174.87 keep state
pass in quick on ep0 proto tcp from any to 193.78.174.86 keep state
pass in quick on ep0 proto udp from any to 193.78.174.86 keep state
pass in quick on ep0 proto icmp from any to 193.78.174.86 keep state
#block in quick on ep0 all
```

```
# block nmap fingerprint attempts
block in quick on ep0 proto tcp all flags FUP

# We do not allow outgoing traffic from honeypot to Internet
# except DNS requests to our name server
# and ftp data connections
pass in quick on ep1 proto udp from 193.78.174.87 to 193.78.174.65
port = 53  keep state
pass in quick on ep1 proto tcp from 193.78.174.87 port = 20 to any
keep state
pass in quick on ep1 proto udp from 193.78.174.86 to 193.78.174.65
port = 53  keep state
pass in quick on ep1 proto tcp from 193.78.174.86 port = 20 to any
keep state
block in quick on ep1

# that should do the trick
```

Only ssh traffic is allowed on the management interface to have some sort of control. The bridged interface allow all traffic in, but only limited traffic out, namely DNS requests and ftp sessions. The first is a necessity to allow name resolution to take place, the second is to allow a hacker to download software. http traffic is blocked to prevent a possible spread of web based worms like Codered.

The output of traceroute (from a system on the same network segment) shows that the bridge is invisible:

```
[root@osiris arthur]# traceroute 193.78.174.86
traceroute to 193.78.174.86 (193.78.174.86), 30 hops max, 38 byte
packets
 1  vmw2k (193.78.174.86)  93.406 ms  2.772 ms  4.134 ms
[root@osiris arthur]# traceroute 193.78.174.87
traceroute to 193.78.174.87 (193.78.174.87), 30 hops max, 38 byte
packets
 1  secure (193.78.174.87)  1.822 ms  1.889 ms  0.941 ms
```

The OS fingerprinting works as well as can be seen from the output of nmap –O on both machines:

```
[root@osiris arthur]# nmap -O 193.78.174.86

Starting nmap V. 2.54BETA30 ( www.insecure.org/nmap/ )
Interesting ports on vmw2k.reseau.nl (193.78.174.86):
(The 1530 ports scanned but not shown below are in state: closed)
Port       State       Service
7/tcp      open        echo
9/tcp      open        discard
13/tcp     open        daytime
17/tcp     open        qotd
19/tcp     open        chargen
21/tcp     open        ftp
25/tcp     open        smtp
```

```
53/tcp     open        domain
80/tcp     open        http
135/tcp    open        loc-srv
139/tcp    open        netbios-ssn
443/tcp    open        https
445/tcp    open        microsoft-ds
1025/tcp   open        listen
1026/tcp   open        nterm
1032/tcp   open        iad3
3389/tcp   open        msrdp
5800/tcp   open        vnc
5900/tcp   open        vnc

Remote OS guesses: Windows Me or Windows 2000 RC1 through final
release, MS Windows2000 Professional RC1/W2K Advance Server Beta3,
Windows Millenium Edition v4.90.3000
Nmap run completed -- 1 IP address (1 host up) scanned in 8 seconds

[root@osiris arthur]# nmap -O 193.78.174.87

Starting nmap V. 2.54BETA30 ( www.insecure.org/nmap/ )
Interesting ports on secure.reseau.nl (193.78.174.87):
(The 1542 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
22/tcp     open        ssh
23/tcp     open        telnet
79/tcp     open        finger
80/tcp     open        http
111/tcp    open        sunrpc
1024/tcp   open        kdm

Remote operating system guess: Linux Kernel 2.4.0 - 2.4.9 (X86)
Uptime 2.867 days (since Fri Mar 29 14:18:36 2002)

Nmap run completed -- 1 IP address (1 host up) scanned in 9 seconds
```

The last thing this transparent bridge is used for is to log all traffic in and out of the honeypot. This is done using tcpdump which writes its output to a logfile. For each of the honeypots the traffic is logged in a separate logfile and these logfiles are rotated on a daily basis. Whenever the honeypot is compromised, all traffic that lead up to this compromise is available for manual analysis (with a minimal gap when the logfiles are rotated).

The transparent bridge is a essential piece of equipment in securing your honeypot and preventing a break out.

Honeypot server

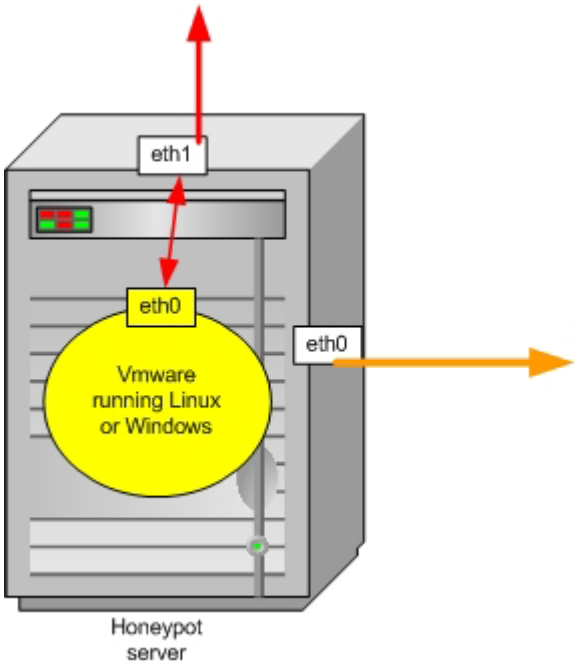
When building the honeypot it is very important it can serve a number of different operating systems (to see which exploits were used on different operating systems). It is also essential that a honeypot can be restored once it has been compromised.

The obvious choice then is to use a linux platform with VMware as a honeypot server.

VMware can support a number of 'interesting' operating systems as guest, among which are Linux, Windows (NT, 2000 and XP) and FreeBSD. It runs on relatively simple hardware as long as there is enough memory and diskstorage available.

When installing a guest os under VMware, it creates a file on the Linux file system that will serve as a disk for the guest os. So restoring a compromised honeypot is as easy as restoring a copy of the VMware disk file.

VMware can use different modes of networking. The only mode of interest for the honeypot is bridged mode in which VMware creates a virtual network adapter in the guest OS that talks (almost) directly to a real network adapter under Linux. It is just as if the network adapter under Linux has an extra MAC address to which it will respond. Using this bridged mode it will bypass almost all security filtering that can be done on Linux. But by using a trick the server can still be made secure. This is how the server looks like, from a network point of view.



The `eth0` and `eth1` of the server correspond to the two network interfaces present under Linux. The `eth0` device (the Linux one that is) is connected to the management network and runs sshd to allow for starting and stopping VMware. Using a standard ipchains setup this interface is more or less protected from unwanted network probes.

The `eth1` device has been configured differently. It has no IP address and does not support ARP. In effect, this interface is invisible from the Internet as it does not respond to IP traffic or ARP requests. But because the VMware virtual network adapter is

bridged to this interface, the guest OS running in the VMware session will respond to network traffic on this interface, as if it were directly connected to this network.

The honeypot platform itself is based on a RedHat 6.2 distribution with all services, except sshd, disabled. A new kernel (2.4.18) has been installed and the sshd has been upgraded as well. By making the system (almost) invisible to the outside world it is hard to break into this machine.

The only risk that remains is that someone manages to break free from the VMware 'prison' due to some bufferoverflow or unknown feature. This would compromise the honeypot platform itself and render the machine useless, or at least unreliable. To prevent this doom scenario from happening, the Linux on the honeypot server has been patched with the grsecurity patch. (http://www.grsecurity.net/). The grsecurity patch installs a ACL system for files and processes that makes it possible to restrict access, even for root. There are more systems available that perform tasks like these, LIDS (Linux Intrusion Detection System, http://www.lids.org/ or LSM, Linux Security Module, http://lsm.immunix.org), but grsecurity was the most mature when developing the honeypot server. Furthermore, grsecurity allows you to hide processes and files, effectively hiding some security measures from prying eyes. It also makes the kernel memory read only so inserting modules via `/dev/kmem` becomes impossible as well.

Additionaly one could harden the honeypot platform further by using the Bastille Linux hardening scripts (http://www.bastille-linux.org/).

A sample ACL configuration for grsecurity is shown below (file.acl):

```
/ rwx
/boot r
/etc r
/etc/rc.d rx
/etc/grsec hr
/etc/samhainrc hr
/etc/lilo.conf r
/etc/shadow r
/etc/passwd r
/etc/ld.so.cache xr
/etc/bastille-tmpdir-defense.sh xr
/sbin rx
/bin rx
/tmp rw
/dev r
/dev/tty rw
/dev/tty1 rw
/dev/zero rw
/dev/null rw
/dev/console rw
/lib rx
/usr rx
/usr/src hr
/usr/local/sbin/samhain hrx
/usr/local/sbin/samhain_stealth hrx
```

```
/var/lib/samhain hr
/var/log ar
/var/log/wtmp rw
/var/log/samhain_log hr
/var/log/snort hr
/etc/rc.d/init.d/samhain hrx
/etc/rc.d/rc2.d/S99samhain hrx
/etc/rc.d/rc2.d/K10samhain hrx
/etc/rc.d/rc3.d/S99samhain hrx
/etc/rc.d/rc3.d/K10samhain hrx
/etc/rc.d/rc4.d/S99samhain hrx
/etc/rc.d/rc4.d/K10samhain hrx
/etc/rc.d/rc5.d/S99samhain hrx
/etc/rc.d/rc5.d/K10samhain hrx
```

and proc.acl:

```
/etc/rc.d/rc vk {
+CAP_NET_ADMIN
+CAP_SYS_ADMIN
/ rwx
}

/sbin/init {
/ rwx
/etc rwo
}

/sbin/mingetty {
/ rwx
/dev/tty1 rwxo
}

/sbin/iptables {
/ rwx
+CAP_NET_RAW
}

/usr/bin/sudo {
/ rwx
/etc/shadow ro
}

/usr/sbin/logrotate {
/ rwx
/var/log ow
}

/usr/sbin/named {
/ rwx
+CAP_SETPCAP
+CAP_SYS_RESOURCE
}

/bin/su {
/ rwx
```

```
/etc/shadow ro
/bin/bash Rx
}

/usr/local/sbin/samhain hp {
+CAP_SYS_ADMIN
+CAP_SYS_RAWIO
/ rwx
/etc/samhainrc forw
/var/lib/samhain forwx
/var/lib/samhain/samhain_file forwx
/var/log/samhain_log forw
/root/archief for
/etc/grsec for
/etc/samhainrc for
/usr/src for
/usr/local/sbin/samhain forx
/var/lib/samhain for
/etc/rc.d/init.d/samhain for
/etc/rc.d/rc2.d/S99samhain for
/etc/rc.d/rc2.d/K10samhain for
/etc/rc.d/rc3.d/S99samhain for
/etc/rc.d/rc3.d/K10samhain for
/etc/rc.d/rc4.d/S99samhain for
/etc/rc.d/rc4.d/K10samhain for
/etc/rc.d/rc5.d/S99samhain for
/etc/rc.d/rc5.d/K10samhain for
}

/usr/sbin/sshd {
+CAP_NET_BIND_SERVICE
/dev/tty row
/ rwx
/var/log/lastlog rwo
/dev/ptmx xrow
/etc/shadow ro
/dev/pts row
/bin/bash Rx
}

/usr/sbin/snort hp {
/ rwx
/var/log/snort rwo
+CAP_NET_RAW
}
```

In this setup both the samhain and snort processes are hidden so they do not appear in the `ps` output or in the `/proc` filesystem.

The honeypot guest os

Well this is the simplest part of it all. Just install the OS you would like compromised and enable the services you would like to have broken into.

Make sure you make a copy of the disk file first before really enabling your honeypot. Only then you can be sure to have a clean and reliable diskimage.

So, now I've caught an exploit, what's next ?
Once the honeypot is up and running it is a matter of time until someone breaks it (which is exactly what it is intended for). Monitoring the honeypot is important to be able to detect this as soon as possible. After detecting a break in a number of things need to be done.

First of all, the broken into VMware session should be stopped (effectively removing the honeypot from the network). This prevents further damage and will 'freeze' the machine as it is. After stopping it, the VMware disk file can be copied to another computer for further analysis. The logfiles of the network monitor should be saved, as that may contain valuable clues as well.

Once these data have been secured, the compromised disk file can be overwritten with the clean copy and the honeypot can be restarted.

When analyzing the break in, the first thing to examine is the network log. This often contains important clues of the type of exploit used. More often than not this will prove to be a well known exploit and it can be added to the history books. However, when the exploit is unknown, the real puzzle begins.

The copied compromised disk file should be installed on a standalone VMware machine. Once copied to that machine, it should be copied again and written to CDrom to make sure it remains unchanged. Using a copy of this compromised disk file one can fire up VMware to examine the actual status of the system.

If the VMware is running Linux or some other form of Unix, the analysis can be done with The Coroners Toolkit.

If the VMware is running a form of Windows, some other options are available. The registry and often contains important information about the events that have taken place. Also the cache of Temporary Internet Files often reveals a lot of information about the break in, just as the Event log and IIS log files will do.

Honeypots, further thoughts, issues and todo's
This paper describes a practical setup of a honeypot environment. After running it for more than six months and seeing a lot of exploits come and go, a lot of issues still have to be addressed, some of which technical, others of a different nature. Below is a list of the most pressing issues that need to be addressed.

- Luring hackers …
  One of the main problems is, and continue to will be, how to 'lure' hackers to your honeypot. There is no surefire way of doing this. Some people drop hints in more or less obscure IRC channels, others post on bugtraq. Probably the best way to do this

is to often change the the actual domain and IP address the honeypot is active for. This way it appears to be a different system each time you move it. To be able to do this you need to be connected to a backbone or have some really good contacts with a number of ISP's.

- Legal aspects …
  Related to the first issue are the legal aspects of luring someone to your honeypot. These legal implications can be a problem when running a honeypot in a corporate environment. There is no easy solution for this.

- Wireless honeypots …
- Dialup honeypots …
  Current honeypots focus on Internet and other IP networks. It would be a very interesting exercise to build a honeypot for a wireless or dialup network as well. This could make wardriving interesting in more than one way.

- Application honeypots …
  This also applies for application specific honeypots. Wouldn't it be an interesting experience to build a fake Oracle application and see how quickly a hacker tries to break it ?

- Outgoing traffic protocol restrictions and rate throttling …
  The current setup of the transparent bridge serves its purpose well. However, it does sometimes puts too much of a restriction on the outgoing traffic. When running the honeypot the author has observed that sometimes a hacker just gave up or rm –rf'ed the honeypot out of frustration. So there is a need for a more subtle control of the outgoing traffic, both in bandwidth, connection rate and content. This will probably evolve into some sort of management imposed by the upstream router and a set of transparent proxies on the bridge, without loosing the transparency

- Automated analysis of traffic logfiles …
  Currently the logfiles are only analyzed when a break in has been detected. It would be far more convenient if the logfiles created by the network monitor could be analyzed automatically. This would require some sort of intelligent pattern matching engine to recognize an attack pattern.

- Snort …
  This one is related to the previous issue. The honeypot server could easily be equipped with snort, listening on the external interface, to perform real time alerting of break in attempts. It does mean that one has to maintain the ruleset of snort on a very regular basis, and be able to act on these real time alerts. Furthermore, it can be argued that you use the honeypot to detect new exploits that are not yet recognized by snort or any other IDS in the first place.

- Encrypted traffic …

More and more rootkits and other hacker tools use ssh as a backdoor connection. Using ssh means that the connection is encrypted end to end and that the network monitor only sees encrypted traffic. The only way to solve this is by patching your shells with keystroke loggers or use a kernel based keystroke logger found at: (http://packetstorm.widexs.nl/UNIX/security/kernel.keylogger.txt) for linux honeypots.
Windows honeypots have a bigger problem since most keyloggers work on the console and not on netcat based network enabled cmd sessions. They may use the NT rootkit which provides some keylogging functionality.
In both cases you have to install some program on your honeypot, so you also run the risk of being detected. It is therefore important to hide both this program and the data it generates.