Honeyd – A low involvement Honeypot in Action

Reto Baumann
GCIA, GSEC, CCNA

me@rbaumann.net
http://security.rbaumann.net

Originally published as part of the GCIA practical

Table of Contents

Conventions Used in this Paper

Normal text is written in 12-point Arial. A lot of command-line command are used, they look the following

```
$ command to issue on a shell, the '$' indicates the
command-line prompt
```

A lot of mysql commands are also used, the look similar to the command used on a shell

```
Mysql> Here comes a mysql command
```

Due to the heavy use of log entries, they have a somewhat smaller appearance

```
Logs are written in 9-point Courier-New
```

Output from all commands is enclosed in a box as if it would be a screenshot

```
Output from a command is surrounded by a box and therefore treated
like a screenshot. ASCII art should also appear correctly
        +---------------------+
        | Small Box in ASCII  |
        +---------------------+
```

Introduction

Honeyd – What could that be? Well, honeyd is a small little program with a great effect – you can spend hours of watching and fine-tuning honeyd and the associated scripts and it is even fun. Honeyd is an application which enables the setup of multiple virtual honeypots on a single machine, each with different characteristics and services. But let's start at the beginning, let's first have a look at the honeypot technology before we are coming back with more details for honeyd.

What is a Honeypot

Honeypots aren't something that new – the basic idea of a honeypot is quite old and was used already for quite a long time. Although, the word "Honeypot" is a new one and the technology is getting more and more important. But let's first have a look at a possible definition of what a honeypot is

"A honeypot is a resource which pretends to be a real target. A honeypot is expected to be attacked or compromised. The main goals are the distraction of an attacker and the gain of information about an attack and the attacker." - R. Baumann, C. Plattner

A honeypot therefore is a system which is acting as a potential target for an attacker. The system itself though isn't of much value to the operator as no valuable information or important services are located on that machine – it's the opposite. All services running on a honeypot aren't used in the productive environment. The services aren't promoted and so there shouldn't be any productive traffic going for these systems. Due to this fact, all traffic heading and reaching a honeypot is of potential value and should be analyzed. A honeypot doesn't need to deal with false positives like an intrusion detection system as there are simply no false positives – all traffic is suspicious as there shouldn't be any traffic because nobody knows of the system, no productive services are running and the system is not involved in "normal" activities.

Two Honeypot Categories

Two categories of honeypots have evolved – research and productive honeypots. Research honeypots are used primarily for research activities like detecting new kind of attacks, retrieving new hacker tools or to get a better knowledge about the attackers, their background, activities and goals. Research honeypots are valuable for developing new IDS signatures, analyze new attack tools or detect new ways of hidden communications or distributed denial of service (DDoS) tools. Research honeypots normally have great logging capabilities to log a hacker's activity once the attacks started or he gained root access.

The other category, the productive honeypots, is mostly used to distract an attacker from the real target. A honeypot is used as a bait to bind his attacking attempts as long as possible to the unproductive honeypot in order to gain time and protect the productive environment in the meantime. A productive honeypot is primarily not interested in gaining new knowledge about the blackhat community – its main interest is the protection of the real servers. Productive honeypots sometimes are also used to gather enough evidence for a successful prosecution of a hacker – But this application is still controversial and the legal side of such procedures is also not clear.

Level of Involvement

Besides the two usage categories of honeypots we already seen, there are also three different technical implementations of honeypots. The essential factor to distinguish here is the "level of involvement". A honeypot is acting like a "normal" server to the attacker – he offers certain services on different ports and could have certain vulnerabilities. Depending on the usage of a honeypot, having some real services on that machine is not always desired or even needed. It could be enough to have a simple listener bound to a port which just writes all incoming packets to a file and never answers to the received request. For catching an infected Microsoft Internet Information Server this is enough, no real IIS is needed. On the other hand, to study a hacker's social network and ways of communicating it could be necessary to "offer a real shell" and allow the attacker to gain root privileges. Once a hacker is root on a system it could be very interesting to see what he's going to do and for what he does need his newly gained system. These different honeypots can be described with the level of involvement

- Low involvement: They are listening on a certain port for incoming connections. All packets are logged. No answer to the request is sent. Low involvement honeypots have no interaction with the attacker. No traffic is ever leaving the honeypot – It's a simple logging machine.
- Mid involvement: Mid involvement honeypots also listen on different ports. But in contradiction to low involvement they send information back to the attacker. A request is answered and the attacker has the possibility to issue commands. Normally, mid involvement honeypots don't use real daemons, instead scripts or small programs are used to imitate the behavior of a service. The provided functionality depends on the script – in most cases, the provided commands are very limited. The big advantage of using such scripts is their logging capabilities and the circumvention of possible vulnerabilities of real services.
- High involvement: High involvement honeypots are the most advanced honeypots. They use real daemons and provide the full set of functionality. An attacker can do whatever he could do to a productive system – no limitations in functionality, vulnerability or behavior. Unfortunately, logging all attempts with high details isn't always easy and the risk of a compromise is growing. Mostly, high involvement honeypots are used when a compromise of a system is desired.

The following table (source "Honeypots" by R. Baumann, C. Plattner) provides an overview of the different honeypot and their level of involvement

	Low Inv.	**Mid Inv.**	**High Inv.**
Degree of involvement	Low	Mid	High
Real operating system	-	-	x
Risk	Low	Mid	High
Information gathering	Connections	Requests	All
Compromised wished	-	-	x
Knowledge to run	Low	Low	High
Knowledge to develop	Low	High	High
Maintenance time	Low	Low	Very high

Even if the idea of honeypots is not that new, honeypots are still in its infancy. Especially the combination of honeypots with intrusion detection systems and firewalls could open some new possibilities in the fight against the blackhat community. It is possible that the

combination of honeypots and intrusion detection could be used to dynamically update firewall rules and actively prevent attacks before they can be targeted at a productive system.

This short description of what a honeypot is, what kind of honeypots there are and how they are setup technically should provide enough background to go on with honeyd.

Honeyd – A Virtual Honeypot

Honeyd is a freely available framework for setting up virtual honeypots. With honeyd it is possible to setup honeypots with different personalities and services on one machine. Honeyd emulates the different operating system's IP stack and binds certain script to a desired port to emulate a specific service. Honeyd is able to fool network fingerprinting tools to think they are dealing with a real operating system ranging from a Windows NT to an AIX box. Even different router's IP stacks can be emulated. Honeyd relies on the nmap fingerprinting file which is used to characterize different kind of operating systems and their IP stacks. Before honeyd is inserting a packet into the IP stream, the personality of the packet is adjusted according to the desired operating system and the corresponding TCP/IP flags. With honeyd it is even possible to emulate complex network architectures and their characteristics. Virtual routing topologies can be defined including different brands of routers, the latency of a network connection as well as the packet loss. When using tools to map the network (like traceroute), the network traffic appears to follow the configured routers and network connections.
The setup of virtual machines is very easy. A configuration file is used to tell honeyd what kind of operating system is desired, how it does respond to closed ports and what kind of service is listening on which port. Honeyd is capable of binding a script to a network port. The script can be a standard shell script which simulates a certain service. Most scripts are built as state machines where a command triggers a certain response or advances to a new state with new possibilities. Scripts for the most popular well known services like SMTP, HTTP and telnet are available at several locations on the Internet.

Honeyd Installation and Configuration

The honeyd installation is straight forward and no problem at all. All you have to get first are three libraries

- libevent - an asynchronous event library
- libnet – a network library
- libpcap - a packet capture library

Installing all three libraries is very easy; just go the usual way of installing a UNIX application with

```
$ ./configure; make; make install
```

Honeyd itself is also installed in the same way. After successfully installing all components it is a wise idea to store all honeyd scripts in a central place.

As mentioned, honeyd is configured via a simple text file where all virtual honeypots as well as the virtual network topology is specified. Each system is first created with a

`create` command. The system then is further specified and configured with `add` and `set` commands. With the `set personality` command, a personality is assigned to a created system. It is further possible to choose the default action for the supported network protocols like `block`, `reset` or `open`. If the default value is set to be `open`, all ports for the desired protocol are in a listening state. The value `reset` defines all ports to be closed while `block` is used to drop all packets for the designated protocol.
Adding services, therefore binding scripts to a certain port, is done by using the `add` command. Instead of binding a script to a port it is also possible to forward the traffic to another IP by using the keyword `proxy`. Honeyd defines four variables which can be very handy: `$ipsrc` for the IP source, `$sport` for the source port, `$ipdst` for the targeted IP and finally `$dport` for the destination port. With these variables it is possible to pass parameters to the scripts or to forward traffic based on one of these values.
The created systems are then assigned to an IP with the `bind` command. The following example configuration file shows most of these commands as well as a basic example for defining a network topology.

```
route entry 192.168.1.1
route 192.168.1.1 link 192.168.1.0/24
route 192.168.1.1 add net 192.168.2.0/24 192.168.2.1 latency 45ms
loss 0.2
route 192.168.1.1 add net 192.168.3.0/24 192.168.3.1 latency 10ms
loss 0.1
route 192.168.2.1 link 192.168.1.0/24
route 192.168.3.1 link 192.168.2.0/24

create router
set router personality “Cisco 7206 running IOS 11.1(24)”
set router default tcp action  reset
add router tcp port 23 “scripts/router-telnet.pl”
bind 192.168.1.1 router

create linux
set linux personality "Linux 2.2.12 - 2.2.19"
add linux tcp port 23 "sh scripts/telnet.sh"
add linux tcp port 22 open
set linux uptime 112211
set linux default tcp action reset
set linux default udp action reset
bind 192.168.1.2 linux
```

The example configuration script generates a simple network topology as well as a Cisco router with telnet running on port 23 as well as a Linux operating system with telnet on port 23 and an open ssh port. The Linux system is set to have an uptime of 112211 seconds.
As it can be seen here, adding new systems, modifying existing ones and even construct virtual networks is straight forward and easily achieved.

Honeyd in Action

To see honeyd in action, I decided to give it a try and configure three different systems with different services with honeyd. The following chapters will describe how honeyd was installed and configured and most importantly, what kind of attacks did hit honeyd during a two weeks testing period.

Chosen Setup

Honeyd was installed as described in one of the earlier chapters. To be able to catch and categorize most attacks I also installed Snort in its latest version available at the time of writing (version 1.9.1) as well as MySQL 4.1 for logging all Snort events to the database for easier analysis. Describing the installation of all involved pieces of software would break the content here, but installing them shouldn't be a problem for a UNIX user/administrator.

One physical system was configured with honeyd to host three virtual machines. Each should listen on its own IP address. Honeyd is using libpcap to listen on a network interface and to capture the traffic. To have traffic for the configured IP's passing our network interface, we have to answer the corresponding ARP requests. For this purpose, arpd was installed. This little tool listens on an interface and answers ARP requests for some desired IP addresses. With the help of arpd it was ensured that the traffic for our virtual honeypots did get to our physical interface.

For honeyd, the following configuration file was used:

```
annotate "AIX 4.0 - 4.2" fragment old
create aix
set aix personality "AIX 4.0 - 4.2"
add aix tcp port 80 "sh scripts/web.sh"
add aix tcp port 22 "sh scripts/test.sh $ipsrc $dport"
set aix default tcp action reset
bind 10.0.0.2 aix

create linux
set linux personality "Linux 2.2.12 - 2.2.19"
add linux tcp port 23 "sh scripts/telnet.sh"
add linux tcp port 22 open
set linux uptime 112211
set linux default tcp action reset
set linux default udp action reset
bind 10.0.0.3 linux

create windows
set windows personality "Windows NT 4.0 Server SP5-SP6"
set windows default tcp action reset
set windows default udp action reset
add windows tcp port 80 "perl scripts/iisemulator-0.95/iisemul8.pl"
add windows tcp port 139 open
add windows tcp port 137 open
add windows udp port 137 open
add windows udp port 135 open
set windows uptime 42002
bind 10.0.0.4 windows
```

For Snort, the following configuration file was used

```
#--------------------------------------------------
#   http://www.snort.org     Snort 1.9.1 Ruleset
#     Contact: snort-sigs@lists.sourceforge.net
#  Ruleset for honeyd field project
#--------------------------------------------------
# NOTE:This ruleset only works for 1.9.1 and later
#--------------------------------------------------
#
##################################################
# Step #1: Set the network variables:
```

```
#
var HOME_NET 62.2.201.201/28
var EXTERNAL_NET !62.2.201.201/28
var DNS_SERVERS $HOME_NET
var SMTP_SERVERS $HOME_NET
var HTTP_SERVERS $HOME_NET
var SQL_SERVERS $HOME_NET
var TELNET_SERVERS $HOME_NET
var HTTP_PORTS 80
var SHELLCODE_PORTS !80

# Path to your rules files (this can be a relative path)
var RULE_PATH /etc/snort/rules

##################################################
# Step #2: Preprocessors
#
preprocessor frag2
preprocessor stream4: detect_scans, disable_evasion_alerts
preprocessor stream4_reassemble
preprocessor http_decode: 80 unicode iis_alt_unicode double_encode
iis_flip_slash full_whitespace
preprocessor rpc_decode: 111 32771
preprocessor bo: -nobrute
preprocessor telnet_decode
preprocessor portscan: $HOME_NET 4 3 portscan.log
preprocessor conversation: allowed_ip_protocols all, timeout 60,
max_conversations 32000

##################################################################
# Step #3: Configure output plugins
#
output log_tcpdump: tcpdump.log
output alert_full: snort-alerts.txt
output database: log, mysql, user=snort password=snort dbname=snort
host=localhost

##################################################################
# Step #4: Customize your rule set
#
# The standard rules where used… but removed here to save space
```

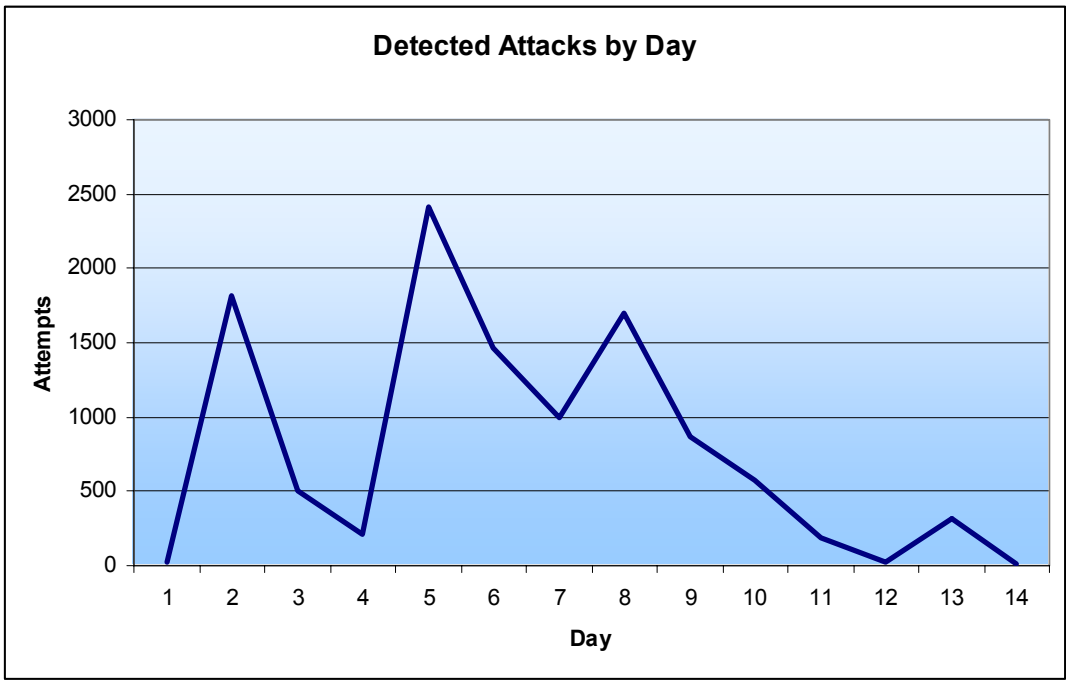
After starting Snort and arpd, the system was ready to host three new virtual honeypots. Honeyd was then started

```
$ honeyd -p nmap.prints -f honeyd.conf -x xprobe2.conf
-a nmap.assoc –l /usr/local/honeyd/logs 10.0.0.2-
10.0.0.4
```

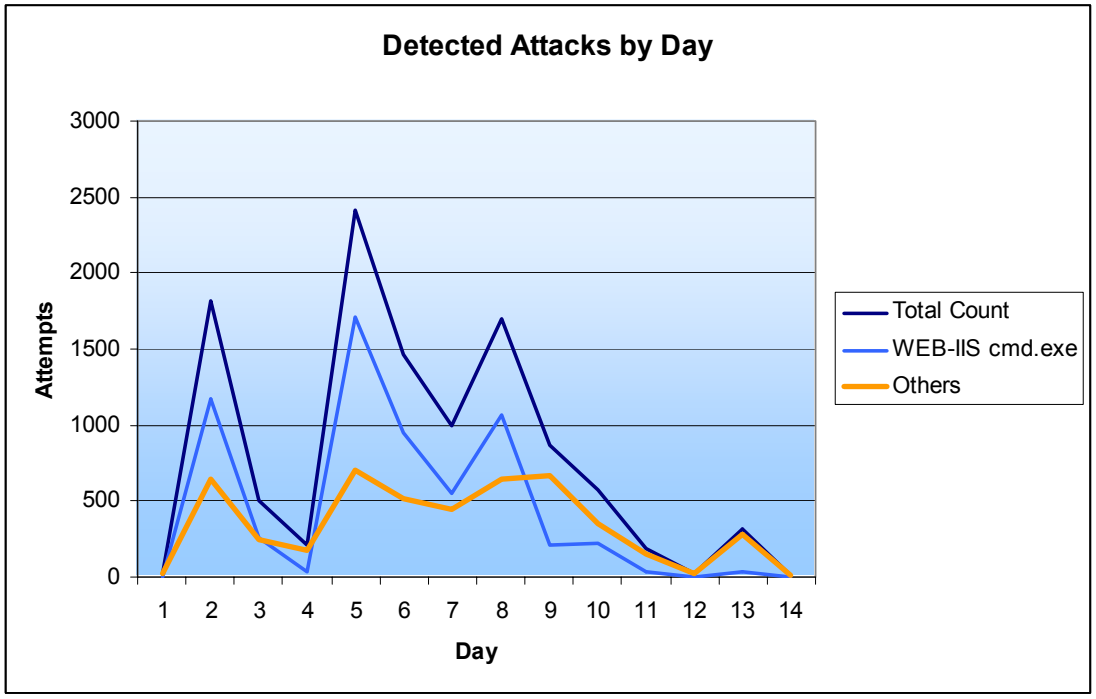
After checking the readiness of our honeypots, the system was left alone for 14 days to gather and log attacks destined for its victims.

Results

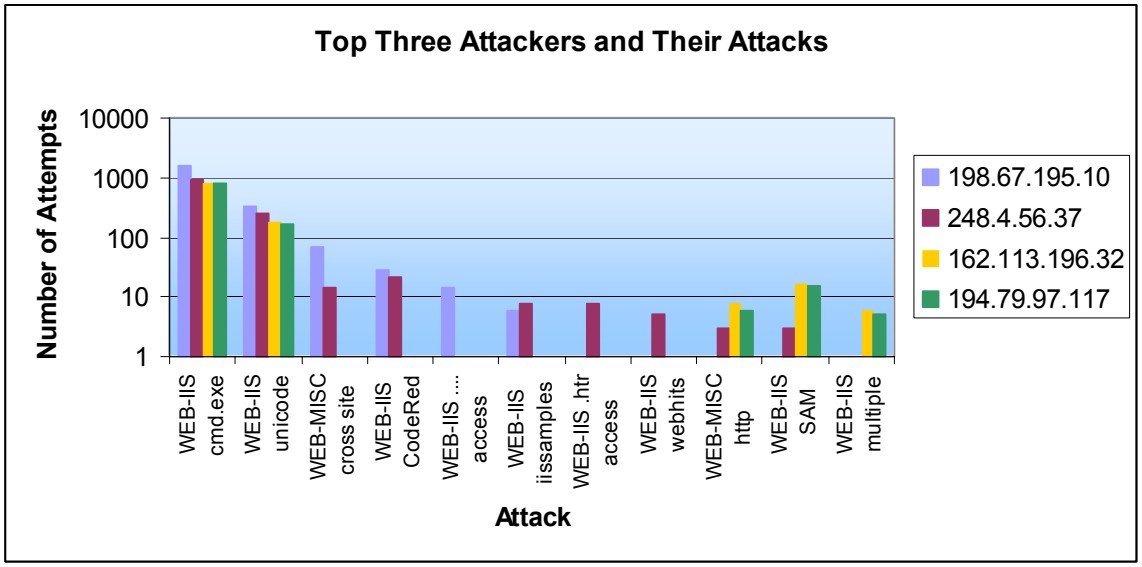
The honeyd configuration was running for exactly 14 days. All events were stored in a MySQL database for easy statistical analysis. In the two weeks, 11121 alerts were generated. It would be interesting to see if we have about the same amount of attacks each day.



As it can be easily seen, we have huge differences in the number of attacks detected each day. By having a look at the alerts itself we can see that the signature “WEB-IIS cmd.exe access” triggered most of the alerts (more than 50%). Doing another chart where we explicitly show the “WEB-IIS cmd.exe access” alerts leads to some new conclusions. As it can be seen, all other alerts don’t lead to such strong peaks which we had on day 2, 5 and 8. We still have some kind of “top-scorer days” but they are not as significant as others.



The next analysis which we were interested in was if we had top attackers and if so what they did. It's interesting to see that three attackers generated about 35% of all alerts. By examining these three attackers a little bit closer we can see that the mostly launched web attacks.



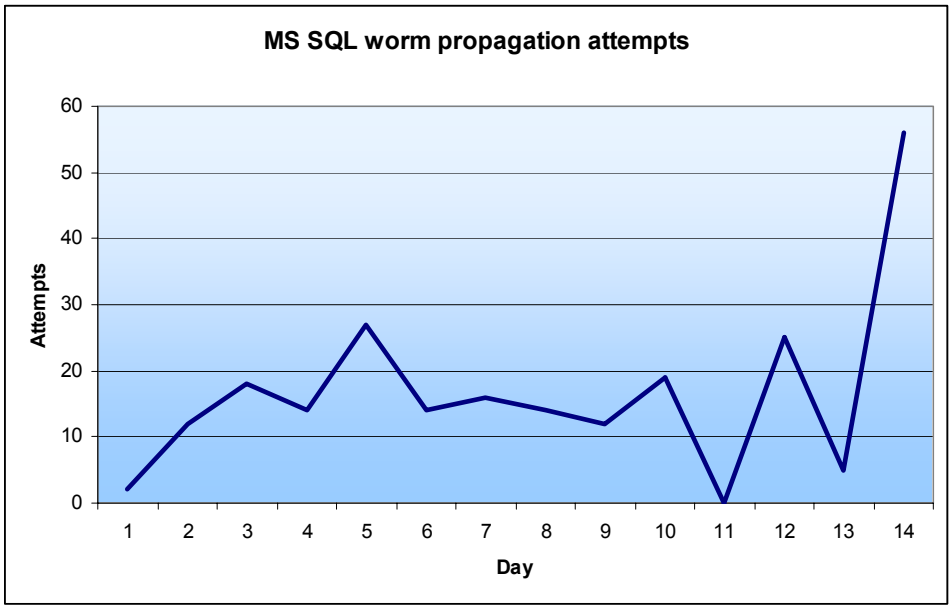
Querying the MySQL database about the alerts reveals the following table

```
mysql> select s.sig_name, count(*) as count from event
e, signature s where e.signature=s.sig_id group by
e.signature order by count desc;
```

sig_name	count
WEB-IIS cmd.exe access	6240
WEB-IIS unicode directory traversal attempt	735
WEB-PHP content-disposition	456
WEB-IIS ISAPI .ida attempt	432
WEB-IIS unicode directory traversal attempt	354
WEB-IIS unicode directory traversal attempt	336
SCAN SOCKS Proxy attempt	319
WEB-IIS unicode directory traversal attempt	235
MS-SQL Worm propagation attempt	234
ATTACK RESPONSES 403 Forbidden	223
WEB-MISC robots.txt access	160
WEB-IIS view source via translate header	153
SCAN Proxy (8080) attempt	150
WEB-IIS CodeRed v2 root.exe access	142
SCAN Squid Proxy attempt	135
ICMP Large ICMP Packet	124
ICMP PING speedera	115
WEB-MISC cross site scripting attempt	85
ATTACK RESPONSES http dir listing	64

```
| ICMP PING CyberKit 2.2 Windows                                |    45 |
| WEB-CGI formmail access                                       |    39 |
| WEB-MISC http directory traversal                             |    38 |
| WEB-IIS SAM Attempt                                           |    36 |
| ICMP superscan echo                                           |    29 |
| ICMP PING NMAP                                                |    25 |
| ICMP Destination Unreachable (Communication Admini...)        |    24 |
| WEB-IIS .htr access                                           |    22 |
| WEB-IIS iissamples access                                     |    17 |
| WEB-MISC bad HTTP/1.1 request, Potentially worm attack        |    15 |
| WEB-IIS webhits access                                        |    15 |
| WEB-IIS .... access                                           |    14 |
| (spp_stream4) STEALTH ACTIVITY (NULL scan) detection          |    12 |
| WEB-CGI formmail arbitrary command execution attempt          |    12 |
| WEB-IIS ISAPI .idq access                                     |    12 |
| WEB-IIS multiple decode attempt                               |    11 |
| WEB-MISC http directory traversal                             |     7 |
| WEB-FRONTPAGE /_vti_bin/ access                               |     6 |
| WEB-IIS Unicode2.pl script (File permission canonicali...)    |     6 |
| WEB-IIS encoding access                                       |     4 |
| WEB-IIS ISAPI .printer access                                 |     4 |
| WEB-IIS ISAPI .idq attempt                                    |     4 |
| ICMP Source Quench                                            |     3 |
| WEB-IIS /iisadmpwd/aexp2.htr access                           |     3 |
| (spp_stream4) TTL LIMIT Exceeded                              |     2 |
| (spp_stream4) STEALTH ACTIVITY (XMAS scan) detection          |     2 |
| WEB-MISC /etc/passwd                                          |     2 |
| WEB-IIS _vti_inf access                                       |     2 |
| WEB-FRONTPAGE _vti_rpc access                                 |     2 |
| WEB-IIS cmd32.exe access                                      |     2 |
| (spp_stream4) STEALTH ACTIVITY (SYN FIN scan) detection       |     2 |
| WEB-MISC .htpasswd access                                     |     1 |
| MISC Tiny Fragments                                           |     1 |
| SCAN myscan                                                   |     1 |
| ATTACK RESPONSES id check returned root                       |     1 |
+---------------------------------------------------------------+-------+
54 rows in set (0.11 sec)
```

Looks like most of our top ten attacks are well known web attacks. Nothing that interesting, except the “MS-SQL worm propagation”. We encountered 234 attempts of an MS-SQL worm propagation attempt. This would result in more than 15 attacks each day. Let’s check if we have a uniform distribution.



As it can be seen, we have days with much more attack attempts and days which no attempt was detected at all. Let's see if all the attacks were from different IP addresses or if we are dealing with a handful of attackers (signature number 27 is our MS SQL worm propagation attempt).

```
mysql> select i.ip_src, count(*) as count from event e,
iphdr i where e.cid=i.cid and e.signature=27 group by
i.ip_src order by count desc;
```

```
+------------+-------+
| ip_src     | count |
+------------+-------+
| 2340602250 |    20 |
| 1093107862 |    13 |
| 3395763701 |     6 |
| 1029824543 |     5 |
| 3395788908 |     5 |
| 1078811399 |     5 |
| 3501812517 |     4 |
| 3256754466 |     3 |
| 1087796482 |     3 |
| 1061685509 |     3 |
...
```

The query reveals that we have two top attackers. Let's check with whom we are dealing here. Our top attacker 139.130.193.138 (2340602250) belongs to an Australian company named Telstra, who is some kind of ISP offering Internet connectivity to their customers. The other IP, 130.79.1.22, belong to a university in France. I informed the administrator of the network about a possible infection of one of his MS SQL servers. The third one is an ISP again, this time from China.

Another attack which draws attention is the "ATTACK RESPONSES id check returned root". Further investigation reveals that this must have been a false positive as it was an incoming packet destined at port 25 (SMTP). It looks as if a mail did contain information which triggered this alert.

Conclusion

Honeyd is a nice little tool which can be perfectly used to setup a low to mid involvement honeypot. The possibility to generate different virtual honeypots on one machine with even different simulated operating systems enhances the usability of this tool even further. It's great for simulating victims and collecting a lot of interesting information. Honeyd could be used as a early warning system in a productive environment to catch some attacks and trigger an alert. Honeyd could also be very useful in detecting infected machines in a network by just sitting there and playing victim. Finding infected web or MS SQL servers can be achieved. The mechanism of attaching a script to a certain port allows a very flexible setup with unlimited capabilities and opportunities for tuning. Honeyd is not as flexible as a high involvement honeypot – the information that can be collected by a high involvement honeypot is much higher. On the downside, the associated risk and the amount of time needed for a good implementation is much higher. Honeyd is simple and the associated risk is very low – A tool which is very handy and much fun to run it.

Software Download Locations

I will shortly present a list of download locations for the different tools. Unfortunately, I can't guarantee that they are still working at a later time. In case the source can't be found there, try google (http://www.google.com) to search for them.

Tool	Download Location
honeyd	http://www.citi.umich.edu/u/provos/honeyd/
arpd	http://www.citi.umich.edu/u/provos/honeyd/
honeyd scripts	http://www.citi.umich.edu/u/provos/honeyd/contrib.html
libevent	http://www.monkey.org/~provos/libevent/
libdnet	http://libdnet.sourceforge.net/
libpcap	http://www.tcpdump.org/

References

Baumann, Reto and Plattner, Christian. “Honeypots”. 2002
URL: http://security.rbaumann.net/download/diplomathesis.pdf (March 14 2003)

Multiple authors. “The Honeynet Project”. 2003
URL: http://project.honeynet.org/ (March 10 2003)

Neville, Alan. “IDS Logs in Forensics Investigations: An Analysis of a Compromised Honeypot”. 2003. URL: http://www.securityfocus.com/infocus/1676 (March 10 2003)

Provos, Niels. “Honeyd – A Virtual Honeypot Daemon”. 2003
URL: http://www.citi.umich.edu/u/provos/papers/honeyd-eabstract.pdf (March 11 2003)

Roesch and Green. “Snort User’s Manual Chapter 2: Writing Snort Rules”. 2003
URL: http://www.snort.org/docs/writing_rules/chap2.html (March 15 2003)

Spitzner, Lance. “Honeypots: Tracking Hackers Website”. 2003
URL: http://project.honeynet.org/ (March 11 2003)

Spitzner, Lance. “Open Source Honeypots, Part Two: Deploying Honeyd in the Wild”. 2003. URL: http://www.securityfocus.com/infocus/1675 (March 10 2003)