

Open Source Honeyd, Part Two: Deploying Honeyd in the Wild

Lance Spitzner 2003-03-12

This is the second part of a three-part series looking at Honeyd, an open source solution that is excellent for detecting attacks and unauthorized activity. In the [first paper](#), we introduced honeypots and discussed what they are, their value, and the different types of honeypots. We then went into detail about the Honeyd, . In this paper we take a closer look at Honeyd. Specifically, we will deploy Honeyd on the big, scary Internet for one week and watch what happens. The intent is to test Honeyd by letting real bad guys interact with and attack it. We will then analyze how the honeypot performed and what it discovered.

The Configuration and Deployment

Before jumping in and covering how Honeyd performed and what it found, let's take a moment to discuss how I configured and deployed the honeypot. For the purposes of this paper, I deployed Honeyd version 0.5 on a secured Linux Red Hat 7.3 system. This version of Honeyd has new features. Most notably, it now allows us to specify multiple IP and IP ranges for the honeypot to monitor at the same time. This gives us more flexibility in what the honeypot monitors. Also, version 0.5 has new logging features. Not only are connections logged to `/var/log/messages`, but Honeyd now supports an optional logging file where all connections are logged in greater detail, which is excellent for post-processing. We will be demonstrating these new features, and others, later in this article.

The network I selected is a small business connection to the Internet. This network is not firewalled, as such it is exposed to all types of attackers, worms, and other malicious activities that are regularly found on the Internet. This is an excellent breeding ground for testing a honeypot. On this network I had five IP addresses that were not in use. I configured Honeyd to monitor four of the IPs with virtual honeypots. As you may remember, Honeyd works by monitoring unused IP space, then interacting with any activity sent to the unused IPs. I used the fifth IP address to remotely administer the honeypot. This IP is a real IP assigned to the physical computer running the honeypot software. You can get a better idea in the IPTables configuration shown below. Using IPTables, I then secured my honeypot. The firewall allows any inbound connection to the four virtual honeypots. However, the firewall strictly controls access to the system itself, in this case I only allowed SSH access from my administration system. To further secure your deployment, you may want to `chroot()` the Honeyd process or `systrace()` it if you are using OpenBSD. Below is the IPTables configuration I used to access control the

honeypot system and the four virtual honeypots.

```
$IPTABLES -A INPUT -i lo -j ACCEPT

# Allow the following inbound inbound
$IPTABLES -A INPUT -p tcp --tcp-flags ALL SYN --dport 22 -j LOG --log-
prefix "Inbound SSH Connection: "
$IPTABLES -A INPUT -p tcp --tcp-flags ALL SYN -s 192.168.1.1 -d
192.168.1.100 --dport 22 -j ACCEPT

$IPTABLES -A INPUT -d 192.168.1.101 -j ACCEPT
$IPTABLES -A INPUT -d 192.168.1.102 -j ACCEPT
$IPTABLES -A INPUT -d 192.168.1.103 -j ACCEPT
$IPTABLES -A INPUT -d 192.168.1.104 -j ACCEPT

# Maintain state of inbound connections.
$IPTABLES -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT

### Set Policies
$IPTABLES -P INPUT DROP
$IPTABLES -P FORWARD DROP
$IPTABLES -P OUTPUT ACCEPT
```

Once I had my platform firewalled and identified the IPs I wanted Honeyd to monitor, the next step was to configure the honeypots themselves. I decided to test Honeyd by using different operating systems and services. In this case I created the following three honeypots; Windows NT 4.0 Server, Linux 2.4.X, and a Cisco router. The fourth honeypot, IP 192.168.1.103, was not given a configuration file. I was curious to see how Honeyd would react monitoring a specific IP with no config. Below is the configuration file I used.

```
### Windows computers
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/iis-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 3284460
bind 192.168.1.101 windows

### Linux 2.4.x computer
create linux
set linux personality "Linux 2.4.16 - 2.4.18"
set linux default tcp action reset
set linux default udp action reset
add linux tcp port 110 "sh scripts/pop3.sh"
add linux tcp port 25 "sh scripts/smtp.sh"
add linux tcp port 21 "sh scripts/ftp.sh"
set linux uptime 3284460
bind 192.168.1.102 linux

### Cisco router
create router
set router personality "Cisco IOS 11.3 - 12.0(11)"
set router default tcp action reset
set router default udp action reset
add router tcp port 23 "/usr/bin/perl scripts/router-telnet.pl"
set router uid 32767 gid 32767
set router uptime 1327650
bind 192.168.1.104 router
```

As you can see in the configuration file, each operating system was assigned respective services, which you would expect to find on the respective systems. The Windows system had NetBIOS ports open (both UDP and TCP), the Linux system had mail and FTP enabled, while finally the Cisco router had a standard Telnet port enabled. Once I had the honeypots configured, I was ready to start my honeypots by running Honeyd. With the new version 0.5 we have several additional options, including the config files `xprobe2.prints`. This file is used to spoof [Xprobe](#) fingerprint attempts. I also used Honeyd's new logging feature but logging attacks to a second additional file, `/var/log/honeyd`. Below you can see how I started my honeypot.

```
honeyd -p nmap.prints -f honeyd.conf -x xprobe2.prints -a nmap.assoc -l  
/var/log/honeyd 192.168.1.101-192.168.1.104
```

In addition to Honeyd's native logging capabilities, I added additional logging functions with [Snort](#). Snort was configured to capture the packets and packet payload of all activity with the four virtual honeypots. Also, Snort generated alerts on any detected malicious activity. I used a [Snort configuration file](#) similar to what the Honeyd Project uses with Honeyd. Combined, Honeyd and Snort proved to be a powerful logging solution.

The Findings

First, keep in mind, as Honeyd has no production activity, nothing should be interacting with the four IP addresses it is monitoring. This makes analyzing the information it captures extremely simple, particularly as anything it captures is most likely hostile in intent. Second, the combination of Honeyd and Snort logs proved extremely useful. The Honeyd logs give you the overall picture, specifically what systems are probing what ports, and when. Version 0.5 now supports two logging formats, so you can select the data that best supports your analysis needs. Once you have identified the traffic you are looking for, Snort can then be used to analyze the packet captures for more detailed information.

Not surprisingly, Honeyd demonstrated that the Internet is an extremely hostile environment. If vulnerable systems are deployed on the Internet, they will be quickly identified and compromised. Worms, autorooters, and active attackers are constantly scouring the Internet for anything they can find. Honeyd quickly and easily demonstrated this. During a one-week period in February 2003, Honeyd was scanned on the average by 67 unique systems per day. Let's take a look at a typical day and see the activity that Honeyd captured.

February 12, 2003 was a particularly busy day. During that day 82 different computers probed our honeypots. None of these systems had any reason, nor had any authorization, to interact with our honeypot. As such we will assume that they were hostile. We begin by first looking at `/var/log/messages`. This is where Honeyd logs all the detected activity in simple, human readable format. Below we see several different scan attempts, with Honeyd explaining what is going on in each logged probe.

- The first scan attempt is made against our Windows NT honeypot. The attacker probes

UDP 1978, which the logs tell us is a closed port. Our honeypot would have responded with an ICMP ERROR Port Unreachable message.

- In the second connection we see an attacker connection to TCP 25 on our Linux honeypot. Not only is this port open but we have a script bound to it, in this case the script smtp.sh, which emulates a mail server. This script logs the interaction with the emulated mail server, which we see in smtp-.log. In this case the attacker is most likely probing for open mail relays. Spammer!
- Following that we see an attacker probe TCP 80 on our Linux honeypot. This port is closed and the honeypot responds with a RST.
- Finally in the last log we see an attacker probe UDP 137, most likely a NetBIOS scan. What is odd is the attacker is using a Private RFC 1918 IP.

```
/var/log/messages
```

```
Feb 12 23:06:33 laptop honeyd[30948]: Connection to closed port: udp
(210.35.128.1:1978 - 192.168.1.101:1978)
Feb 12 23:23:40 laptop honeyd[30948]: Connection request: tcp
(66.136.92.78:3269 - 192.168.1.102:25)
Feb 12 23:23:40 laptop honeyd[30948]: Connection established: tcp
(66.136.92.78:3269 - 192.168.1.102:25) <-> sh scripts/smtp.sh
Feb 12 23:24:14 laptop honeyd[30948]: Connection dropped with reset:
tcp (66.136.92.78:3269 - 192.168.1.102:25)
Feb 12 23:34:53 laptop honeyd[30948]: Killing attempted connection:
tcp (216.237.78.227:3297 - 192.168.1.102:80)
Feb 12 23:39:14 laptop honeyd[30948]: Connection: udp (10.5.5.71:1026 -
192.168.1.101:137)
Feb 12 23:39:14 laptop honeyd[30948]: Connection established: udp
(10.5.5.71:1026 - 192.168.1.101:137)
```

```
/tmp/honeyd/smtp-.log
```

```
Wed Feb 12 23:23:40 UTC 2003: SMTP started from Port
EHLO relay.verizon.net
MAIL From:
RCPT To:
```

In the log file /varlog/honeyd, we see the same information logged, but in a different format. This information is designed to be read and processed by scripts or tools. Also, notice to the far

right of the connections. These numbers show the number of total bytes in the connection attempt.

```
/var/log/honeyd
2003-02-12-23:06:33.0633 udp(17) - 210.35.128.1 1978 192.168.1.101
1978: 69
2003-02-12-23:23:40.0600 tcp(6) S 66.136.92.78 3269 192.168.1.102 25
2003-02-12-23:24:14.0940 tcp(6) E 66.136.92.78 3269 192.168.1.102 25:
98 342
2003-02-12-23:34:53.0969 tcp(6) - 216.237.78.227 3297 192.168.1.102 80:
48 S
2003-02-12-23:39:14.0008 udp(17) S 10.5.5.71 1026 192.168.1.101 137
2003-02-12-23:39:14.0194 udp(17) - 10.5.5.71 1026 192.168.1.102 137: 78
```

The Honeyd logs prove very effective in telling us what is going on. I normally start there to determine what activity is occurring. If any connections are made to ports with scripts bound to them, the scripts often have their own additional logging features built into them, as we saw with the smtp.sh script. The only disappointment was that I could not get the IIS Web server script to work with Honeyd.

In addition to the Honeyd logs, every one of these connections was also captured by Snort. We can analyze the packets and packet payloads of every attacker for additional information, such as passive fingerprinting. If an actual attack had been attempted, Snort would detect the attack and generate an alert. For February 12th, Snort detected 48 attacks, including FTP CWD overflow attempts, Socks Scans, and WEB-IIS unicode directory traversal attempts. All in all, just a typical day on the Internet.

The real value of Honeyd, and honeypots in general, lies not only in what they detect, but also what they don't detect: false positives. Honeypots virtually eliminate false alerts. Almost all of the information they collect indicates malicious activity. Since the data sets are small, you can quickly identify and react to threats. Tools such as Honeyd make an excellent technology to complement your IDS systems. Imagine if you deployed something like this on your internal network, what do you think you would find?

Conclusion

Honeyd demonstrates the true detection capabilities of honeypots. Keep in mind that this tool is in its infancy: you can expect many new and exciting features to come with Honeyd. If you would like to learn more about Honeyd, or honeypots in general, you may want to start with the site <http://www.tracking-hackers.com>, or with the author's book [Honeypots: Tracking Hackers](#).

To read the next installment to this article, click [here](#).

Relevant Links

[Open Source Honeypots: Learning with Honeyd](#)

Lance Spitzner, SecurityFocus

[Privacy Statement](#)

Copyright 2006, SecurityFocus