

Open Source Honeypots: Learning with Honeyd

Lance Spitzner 2003-01-20

Honeypots are an exciting new technology. They allow us to turn the tables on the bad guys, we can take the initiative. In the past several years there has been growing interest in exactly what this technology is and how it works. The purpose of this paper is to introduce you to honeypots and demonstrate their capabilities. We will begin by discussing what a honeypot is and how it works, then go into detail using the OpenSource solution Honeyd.

What is a Honeypot?

This question is more difficult to answer than it sounds. Many people have their own definition of what a honeypot is, or what it should accomplish. Some feel its a solution to lure or deceive attackers, others feel its a technology used to detect attacks, while other feel honeypots are real computers designed to be hacked into and learned from. In reality, and hence the confusion, they are all correct.

A honeypot is a security resource whose value lies in being probed, attacked, or compromised. The key point with this definition is honeypots are not limited to solving only one problem, they have a number of different applications. To better understand the value of honeypots, we can break them down into two different categories: production and research. Production honeypots are used to protect your network, they directly help secure your organization. Research honeypots are different; they are used to collect information. That information can then be used for a variety of purposes, such as early warning and prediction, intelligence gathering, or law enforcement.

Neither solution is better than the other, it all depends on what you want to achieve. For the purpose of this paper we are going to focus on production honeypots. To learn more about research honeypots, you may want to start with the article [Know Your Enemy: Honeynets](#), by the Honeynet Project.

To better understand the value of production honeypots, we will use Bruce Schneier's model of security, specifically the three layers of prevention, detection, and response. Honeypots can apply to all three layers. For prevention, honeypots can be used to slow down or stop automated attacks. For example, the honeypot [LaBrea Tarpit](#) is used to "tarpit" or slow down automated TCP attacks, such as worms. Against human attackers, honeypots can utilize

psychological weapons such as deception or deterrence to confuse or stop attacks. You can learn more about deception at [Fred Cohen's site](#).

Honeyd can also be used to detect unauthorized activity. Traditional detection solutions can overwhelm organizations with alerts, yet only a few of the alerts signal valid attacks. Also, many of today's technologies are not designed to detect unknown attacks. Honeyd help resolve both of these problems. Honeyd generate very few alerts, but when they do you can almost be sure that something malicious has happened. Honeyd can also detect and capture unknown attacks as well as known attacks.

Finally, production honeyd can be used to respond to an attack. If an attacker breaks into your organization, and one of the systems they broke into was a honeyd, then information gathered from that system can be used to respond to the break-in. Honeyd can also be used to smoke out and identify an attacker once he or she is in your organization. For a more indepth look at honeyd, you may want to consider the author's book [Honeyd: Tracking Hackers](#).

Honeyd

Now that you have a better understanding of honeyd technologies, lets actually build one. In this case, we are going to cover the OpenSource solution [Honeyd](#), which was created and maintained by Niels Provos. It's designed to be used on Unix-based operating systems, such as OpenBSD or Linux; however, it may soon be ported to Windows. Since this solution is OpenSource, not only is it free, but we also have full access to the source code, which is under the BSD license.

The primary purpose of Honeyd is detection, specifically to detect unauthorized activity within your organization. It does this by monitoring all the unused IPs in your network. Any attempted connection to an unused IP address is assumed to be unauthorized or malicious activity. After all, if there is no system using that IP, why is someone or something attempting to connect to it? For example, if your network has a class C address, it is unlikely that every one of those 254 IP addresses is being used. Any connection attempted to one of those unused IP addresses is most likely a probe, a scan, or a worm hitting your network.

Honeyd can monitor all of these unused IPs at the same time. Whenever a connection is attempted to one of them, Honeyd automatically assumes the identity of the unused IP

addresses and then interacts with the attacker. This approach to detection has many advantages over traditional methods. Any time Honeyd generates an alert, you know it most likely is a real attack, not a false alarm. Instead of hammering you with 10,000 alerts a day, Honeyd may only generate 5 or 10. Furthermore, since Honeyd is not based on any advance algorithms, it is easy to set up and maintain. Lastly, it not only detects known attacks, but unknown ones as well. Anything that comes its way is detected, not only that old IIS attack, but also that new RPC 0-day attack no one knew about.

By default, Honeyd can detect (and log) any activity on any UDP or TCP port, as well as some ICMP activity. You do not need to create a service or port listener on ports you want to detect connections to, Honeyd does this all for you. However, with Honeyd, you have the additional option of not only detecting attacks, but also creating emulated services that interact with the attacker. These emulated services allow you to determine what the attacker is attempting to do, what they are looking for. This is done by creating scripts that listen on specific ports and then interact with attackers in a predetermined manner. For example, you can create an FTP script that emulates a `wu-ftpd` daemon on Linux, or a Telnet connection on a Cisco router.

These emulated services are limited in that they act in a predetermined behavior. The script can be written in almost any language, such as Perl, Shell, or Expect. At the time of publication of this paper, Honeyd had seven emulated services you could choose from. Since Honeyd is OpenSource, you can create and contribute your own services. Below is an example of a service emulating a Cisco router. In this case, when the attacker connects to the Honeyd honeypot, he will be deceived into thinking they have found a Cisco router.

```
attacker $telnet 192.168.1.150
Trying 192.168.1.150...
```

```
Users (authorized or unauthorized) have no explicit or
implicit expectation of privacy. Any or all uses of this
system may be intercepted, monitored, recorded, copied,
audited, inspected, and disclosed to authorized site,
and law enforcement personnel, as well as to authorized
officials of other agencies, both domestic and foreign.
By using this system, the user consents to such
interception, monitoring, recording, copying, auditing,
inspection, and disclosure at the discretion of authorized
```

site.

Unauthorized or improper use of this system may result in administrative disciplinary action and civil and criminal penalties. By continuing to use this system you indicate your awareness of and consent to these terms and conditions of use. LOG OFF IMMEDIATELY if you do not agree to the conditions stated in this warning.

User Access Verification

Username: cisco

Password:

% Access denied

This attack would have then been detected and logged. Notice how Honeyd logs the attack that was attempted, the login attempt, and when the attacker closed the connection. You also see the script used to emulate this service.

```
Jan 3 11:23:32 marge honeyd[22885]: Connection request: (192.168.1.10:2783 -
192.168.1.150:23)
Jan 3 11:23:32 marge honeyd[22885]: Connection established:(192.168.1.10:2783 -
192.168.1.150:23) <-> /usr/bin/perl scripts/router-telnet.pl
Jan 3 11:23:42 marge honeyd[22885]: E(192.168.1.10:2783 - 192.168.1.150:23):
Attempted login: cisco/cisco
Jan 3 11:23:47 marge honeyd[22885]: Connection dropped with reset:
(192.168.1.10:2783 - 192.168.1.150:23)
```

Honeyd also introduces another exciting feature for honeypots: emulating operating system at the kernel level. We just discussed how Honeyd can emulate different operating systems by using different scripts for different services. This capability is not unique, as many other honeypots use it. However, Honeyd can also emulate different operating systems at the kernel level. Attackers often remotely fingerprint operating systems by using such tools as [Nmap](#) or [Xprobe](#). Honeyd takes the same fingerprint database used by Nmap to spoof the responses of

any operating systems you are emulating. In the case of our Cisco router, if the attacker used the `-o` option of Nmap to fingerprint the IP address, the results would have told him it was a Cisco router. This helps increase the realism of the honeypot.

Configuring Honeyd

To implement Honeyd we need to compile and use two tools: Arpd and Honeyd. Honeyd cannot do everything alone and requires the help of Arpd. Arpd is used for ARP spoofing; this is what actually monitors the unused IP space and directs attacks to the Honeyd honeypot. Honeyd does not have the capability to direct attacks to it, it only has the capability to interact with attackers. The commands to start both are listed below. The networks in the command are the networks that Arpd will monitor and Honeyd will interact with. In our example, we want our honeypot to monitor all unused IP space in the 192.168.1.0/24 network.

```
arpd 192.168.1.0/24
```

```
honeyd -p nmap.prints -f honeyd.conf 192.168.1.0/24
```

So, based on the command above, the Arpd process will monitor any unused IP space on the 192.168.1.0/24 network. If it sees any packets going to unused IP's, it will direct those packets to the Honeyd honeypot using Arp spoofing, a layer two attack. It spoofs the victim's IP address with the MAC address of the Honeypot. As this is a layer two attack, it also works in a switched environment. For the Honeyd command, `-p nmap.prints` refers to the Nmap fingerprint database. This is the actual database that the scanning tool Nmap uses to fingerprint operating systems. Though this comes with the Honeyd source code, you may want to get the most recent fingerprint database from [Nmap](#) directly. The second option for the Honeyd process, `-f honeyd.conf`, is the honeypot configuration file. This determines how you want the honeypot you behave. Below is one 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
```

You start off with by creating different types of computers you want to emulate, or what Honeyd calls templates. These templates define the behavior of each emulated operating system. In this configuration file we have created two different emulated computers: *default* and *router*. The first thing we need to do in each template is assign the "personality"; this is what operating system will emulate at the IP stack level. You give it the OS type using the same description in the Nmap fingerprint database. In the template *default*, we have assigned the personality "Windows NT 4.0 Server SP5-SP6" and in the template *router* we have given it the personality "Cisco 4500-M running IOS 11.3(6) IP Plus". Note, the personality does not affect the behavior of the emulated services, it only modifies the behavior of the IP stack. For the emulated services, you have to select different scripts based on what type of OS you want to emulate. In other words, if your personality is Windows, don't bind an emulated Apache script to the HTTP port. Instead, you would bind an emulated IIS script to the HTTP port.

The next step is to define the behavior of each port. You can either assign specific ports specific behavior, or define general behavior. For example, in the template *default*, we have assigned all the TCP ports the behavior reset, so they respond with a RST to any connection attempts (for UDP, ICMP port unreachable). Other options are open (will respond with ACK, or for UDP nothing) or block (will not respond for both TCP and UDP). A fourth option is the use of scripts to emulate services. In the case of the template *default*, we have bound scripts to ports 21, 80, and 110. These are the actual scripts that are executed and interact with the attackers. You

also have the option to proxy connection attempts to other systems, or even the attacker. In the *default* template, we actually proxy all SSH connections back to the attacker. There are several other more advanced features of Honeyd, such as creating virtual, routed networks and spoofed timestamps, but a detailed explanation is beyond the scope of this paper.

Once you have created your templates, you have to decide which IP addresses are bound to which template. Using the `bind` command, as we do in the *router* template, you can bind the template to specific IP addresses. 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 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, at which they will get the Cisco router.

Before deploying, the author of Honeyd (Niels Provos) recommends running [Snort](#) on your honeypot to capture additional information. The advantage with Snort is that not only can it give you more information on attacks using its IDS alerts, it can also capture every packet and its full payload that comes to and from the honeypot. That information can be critical for analyzing attacks, especially unknown ones.

Normally it is resource intensive to capture all the network activity and full packet payload. However, keep in mind that honeypots get very little traffic, and what they do get is most often malicious or unauthorized activity. To help users better understand and work with Honeyd, Marcus Ranum and I have created a pre-compiled [Linux Honeyd Toolkit](#) for you. This toolkit has statically compiled Arpd and Honeyd binaries for Linux, with all the required configuration files and startup scripts discussed in this paper. The intent is that you can download the toolkit to your Linux computer and immediately begin working with Honeyd.

Conclusion

The purpose of this paper was to introduce you briefly to the concepts of honeypots and their value. We then discussed in detail how one such honeypot, Honeyd, works and how you could deploy your own. If you would like to learn more about Honeyd, or honeypots in general, you may want to start with the author's site <http://www.tracking-hackers.com>.

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

Relevant Links

[Know Your Enemy: Worms at War](#)

The Honeyd Project

[Know Your Enemy: Honeyd](#)

The Honeyd Project

[Know Your Enemy: Building Virtual Honeyd](#)

The Honeyd Project

[The Value of Honeyd, Part One: Definitions and Values of Honeyd](#)

Lance Spitzner with Marty Roesch

[The Value of Honeyd, Part Two: Honeyd Solutions and Legal Issues](#)

Lance Spitzner with Marty Roesch

[Intelligence Gathering: Watching a Honeyd at Work](#)

Toby Miller, SecurityFocus

[Privacy Statement](#)

Copyright 2006, SecurityFocus