

Honeypots: Covert Network Security

Joel Spriggs

jspriggs@franklincollege.edu

Michael Lavengood

mlavengood@franklincollege.edu

Information Technology Services

Franklin College

101 Branigin Blvd.

Franklin, IN 46131

(317) 738-8148

Introduction

The concept of a honeypot is fairly simple, it does nothing. The honeypot exists on a system that is not used by anyone for any purpose. So, if the system has no users other than someone checking logs, then there should be no substantial network activity aside from communication with routers for ARP table entries from time to time. Therefore, any activity on the system is genuinely suspect in nature. Looking at logs of any and all network activity with the non-production system can give an idea to what happened. There is a possibility that some piece of equipment is malfunctioning on the network and accidentally communicates with the honeypot, but that can be easily recognized, traced, and fixed. Any other traffic can be labeled as a malicious activity. Any traffic leaving the honeypot can be a direct indication that the honeypot itself has been compromised and the attacker is using it to possibly stage other attacks. So, you can expect nothing from the system, and anything you get will be directly helpful. This gives us a few advantages with using honeypots. They are simple to build and deploy, they don't require many hours of development and management. The return on investment can be seen easily in the value of the data. Any data collected will be of value to the organization since it will give direct information about the current security procedures and threats. Also, you see it directly and it is clear. With a firewall, you know it's probably deterring some attacks, but it's not really clear the number or breadth of attacks being repelled. With a honeypot you will see exact attacks and what is happening.

All that glitters is not gold though, and there are disadvantages to honeypots. For instance, you can gain a view of your network and its security, but it will be fairly narrow. The honeypot will only capture activity directed at the honeypot and nothing else, although the more honeypots you have on the network, the more probability you will have of capturing possible attacks. With that, though, you have the disadvantage of costs becoming higher and more time in maintaining and monitoring the honeypots. Also, there is a fair amount of risk associated with a honeypot. There is the risk that an attacker could fingerprint the system and decide to retaliate by destroying it and all data the honeypot has collected or other systems on the network. Another risk is that the attacker could compromise the system and use it to launch attacks against other systems in or outside of our organization. Due to these disadvantages, honeypots should not be used to replace other network security tools such as firewalls or intrusion detection systems, but should instead be used in conjunction with these devices to bolster security and allow covert views of activity, like that of a mole in an organization.

Types of Honeypots

Honeypots are rated generally in three categories: low, medium and high interaction. These interaction categories refer to how much interaction the hacker will have with the actual operating system and how much of the operating system is simulated, and will dictate how much risk there is of the attacker being able to take control of the system. Typically, how much information can be gathered from the honeypot will also relate directly to the interaction level.

Most low interaction honeypots will only give some basic information about what the hacker is doing, and the more interaction you give them, the more information you can gather. For instance, a low interaction honeypot will generally have a hands-off approach with hackers. This can be characterized as a port scanner or key monitor. The value of a low interaction honeypot is that it will let you know what ports are being scanned and are at risk on your network. This can be an effective way of tracking script kiddies and worms that are working through the network. A worm can be tracked using a low interaction honeypot by only having to monitor ports on the system and log where the incoming traffic is coming from. With this information we can deduce what type of worm was most likely attacking the honeypot by the port it was scanning, which the port can typically be traced back to a general vulnerability that the worm is looking to exploit. We also know where the worm came from and now know another system on the network that has already been compromised by that worm. We can now fix that system and move on to looking for other systems that are compromised.

Medium levels of interaction with honeypots will give attackers some basic options with the operating system and services that are being used as traps. Using a medium level honeypot, you will generally get more information on your attacker. At this point you can start easily discerning script kiddies from black hats and capturing what they do. You will learn more about what the attacker is doing, because they have a larger ground to work with than with the low interaction honeypots. It is also at this level, that you can start working with customizing your honeypot to appear more like exact types of computers and systems. For instance, you can create an emulation of a web server that has some but not all functionality. This gives you the ability to have more lure for the attacker to stay around while you trace him/her, but the safety of knowing they don't have as many options as they think. You will also have some added risk though. There is a possibility they can break through your barriers and figure out a way to take control of the system anyway.

High interaction honeypots are very challenging to work with. These honeypots give the attacker basically a full system to work with. This will give you the greatest insight to what attackers are doing with your system and captures everything they do. With this, the attacker won't be necessarily warned by features not working correctly and may think it is a standard production machine. There are many difficulties with this however. Creating exact replicas of systems for hackers to break into while maintaining security of the machine and monitoring capabilities can be extremely difficult. Also, there is an even greater risk of the attacker being able to break through the barriers and take control of the machine, at which point they could use it again as a staging ground for attacks on other systems in the organization or outside on the internet.

Generally, these interaction levels are taken into account when deciding what you want to do with your honeypot. If you want a honeypot that has a value in production with the organization, then you would most likely work with a low interaction or medium interaction honeypot. The reason is that these will give an adequate measure of security in prevention and detection of attacks while maintaining a minimal risk to the rest of the organization. These production honeypots will appear to be other production machines within your organization and will prevent attacks that could have been directed to other systems had the honeypot not been there. They will also act as a detection method by letting you know what is roaming around the network. On the other hand, if you want to research hackers and any kind of attack being perpetrated you would want a higher medium interaction honeypot or a high interaction honeypot to get as much as you can out of attacks. With this research honeypot, you can collect data to make generalizations of what to expect from attackers and where they are going. More often, the research honeypots are not used in corporate environments as they usually cost more in time to operate and maintain. Most research honeypots are being used in academic environments or with security companies that want to know what is on the horizon of data security.

For our purposes, we will be looking into customizable honeypot software that can create a low, middle, or high interaction honeypot. Honeyd, the software we will be using in our honeypot, is usually installed on a Unix based system and can emulate any service, nearly any operating system and monitor any port.

Our System

For our Honeypot, we will be using a Compaq Armada E500 with a 1Ghz processor, 256 MB of ram and a 40GB hard drive. This should be more than sufficient for the small amount of software our honeypot will actually be running. I have installed Red Hat Linux 9.0 on it for the operating system. The reason for this is that Honeyd currently only has stable releases for BSD, GNU/Linux and Solaris systems. The computer has been given a static IP address.

Honeyd is a daemon written for Unix based operating systems by Niels Provos. Honeyd will create virtual hosts on a network. In other words, when Honeyd is deployed on a network it can emulate any number of separate computer systems, as well as emulating many different operating systems. It emulates each service based on script sets that are programmed and referenced in the configuration files. The emulation of the operating system and the services are only as detailed as the scripts are written to make them.

Here is an example of one script:

```
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 /etc/honeyd/scripts/iis/main.pl"
add windows tcp port 25 block
add windows tcp port 23 proxy real-server.tracking-hackers.com:23
add windows tcp port 22 proxy $ipsrc:22
set windows uptime 3284460
```

```
#bind specific IP addresses to specific templates  
bind 192.168.1.200 windows
```

The example above was one created by Lance Spitzner in his book *Honeypots: Tracking Hackers*. I use this example because it shows a really basic setup for a Honeyd configuration and is easy to learn from. Going line by line in this script, the first will create the template of what you are trying to emulate. In this case, this is creating a template called windows. The next line sets the personality of windows to Windows NT 4.0 Server, specifically the release of service pack 5 and 6. Setting the personality will tell the system how to behave at the IP stack level of the operating system. This is necessary because every operating system operates slightly differently at the IP stack level. Due to this slight difference, a system can be fingerprinted and uniquely identified by looking at the structure of the IP stack. Attackers can use a tool called Nmap, which will scan ports and the IP stack of a target system to determine what services are running and it checks against a database to see what operating system is on the system. Needless to say it would look a little suspect to have a Linux system emulating a Windows service when it is fingerprinted against Nmap, thus alerting the hacker that what they are about to encounter is most likely a honeypot. To counter this, Honeyd uses the same Nmap databases to determine how to structure the IP stack when it is sending and receiving packets. It relates the personality to the Nmap database, and will now interact using the IP stack for a system running Windows NT 4.0 Server instead of one running Red Hat Linux 9.0.

The next lines:

```
set windows default tcp action reset  
set windows default udp action reset
```

simply tell Honeyd that when it encounters any tcp or udp connections it will just reset the connection and not allow any access, emulating a closed port. The next line tells Honeyd that any traffic that comes over port 80, the default port for HTTP traffic, will be sent to the script main.pl in the folder for IIS. Main.pl in this case is a perl script that emulates an IIS web server on Windows NT. This gives the hacker a web server that is as functional as the perl script will make it.

The next line will take any packets received going to port 25 and drop them, this emulates a firewall port. The next line tells Honeyd to take all traffic coming to port 23 and redirect it to a server called `real-server.tracking-hackers.com:23`. The server can be any server, it doesn't have to be named, it could just be the IP address of that particular server, and the ":23" will direct the traffic to port 23 on that server. The last add line in the configuration will do another proxy, but this time it will redirect all traffic coming in on port 22 back to the original source of the traffic. In other words, Honeyd will send the exact same packet back to the sending machine on port 22. The last setting in the windows template sets the uptime. A system can look suspicious if it appears to have just come online recently. This setting will tell Honeyd to make the system appear to have been on for 3,284,460 seconds, or 38 days and 21 minutes. If the uptime is not specified, the program will assign a random uptime between 0 and 20 days.

The last two lines in the configuration are a comment and a bind command. The bind will tell Honeyd to take any traffic that is destined for the IP of 192.168.1.200 and use the windows template. Using bind commands in this fashion and many separate templates, a single honeypot with Honeyd can take on thousands of IP addresses, split them up to emulate different operating systems and services for specific IP ranges.

At this point it would be best to describe how Honeyd is able to emulate so many different machines at one time. The set up for how Honeyd interacts with networks and packets is somewhat network specific; there is no one exact way to set it. The method we will be using is termed as blackholing.

In an Ethernet setup, packets of data are sent from the clients, servers and the internet to the routers and switches, which then disperse the packets to their destinations. The Franklin College network is setup a little bit differently. In our network, the clients and servers are all addressed depending on where they are physically located. In this situation, sub-networks are created so that traffic is able to be redirected faster to where it is destined. For example, when a router gets some packets that are being sent to a computer with an IP address starting with 10.23, it will know that they are going to Elsey and route them to the routers in Elsey. Each subnet has the potential of over 64,000 unique IP addresses.

For our honeypot's purposes, Michael Lavengood, Network Security Administrator at Franklin College, and Jim Riggle, Network Administrator, have set up a new subnet in the routing scheme. The subnet they have set up will create the 10.96 network for our honeypot. In this case, what the router will do is simply redirect all traffic destined for an IP address starting with 10.96 into the honeypot running Honeyd. In this sense, we will now have a potential of more than 64,000 virtual hosts all running simultaneously. If all were running at once, it is assumed that Honeyd won't have any issues, as it has been tested by its creator to take on 65,536 in a simulation, though it is doubtful all 64,000 will be hit at one time.

The router has been configured so that the honeypot will never see the internet. The router has been set this way to narrow the scope of what the honeypot sees only to the campus network so we can study just the situation of the network as it is and for legal reasons that will be discussed later. This setup will create a black hole in the network, where no other computers will take up any real existence.

The other common method of setting Honeyd onto a network is very tricky and uses a technique known as ARP Spoofing. Routers store all addressing information into a table called an ARP (Address Resolution Protocol) table. The ARP table basically references IP addresses to MAC (Media Access Control) addresses. In the situation of ARP spoofing, Honeyd will fool the routers ARP tables into letting it take up the entire unused network. This method is used in a situation where there are systems used for production and systems not used for production (honeypots). What Honeyd will do, is use a program called Arpd to keep track of the ARP tables. It will look at the structure of the current ARP table and decide what all IP addresses are available. When traffic comes to the router, it will look to see if the traffic is destined for any of the real entries (actual production systems) in the table. If the packets are meant for an address not in the ARP table, it will send them directly to Honeyd for it to take over and acts like a target production system.

While this method can cover a lot more IP addresses, it is very dangerous and can cause problems with other users getting into the routing table. By not using this system, but creating a

black hole, we will short circuit a long list of potential problems while still catching what we are looking for, wandering probes that can attempt attacks on non-existent machines on a network. With our black hole, the network can now be viewed like Figure 5.

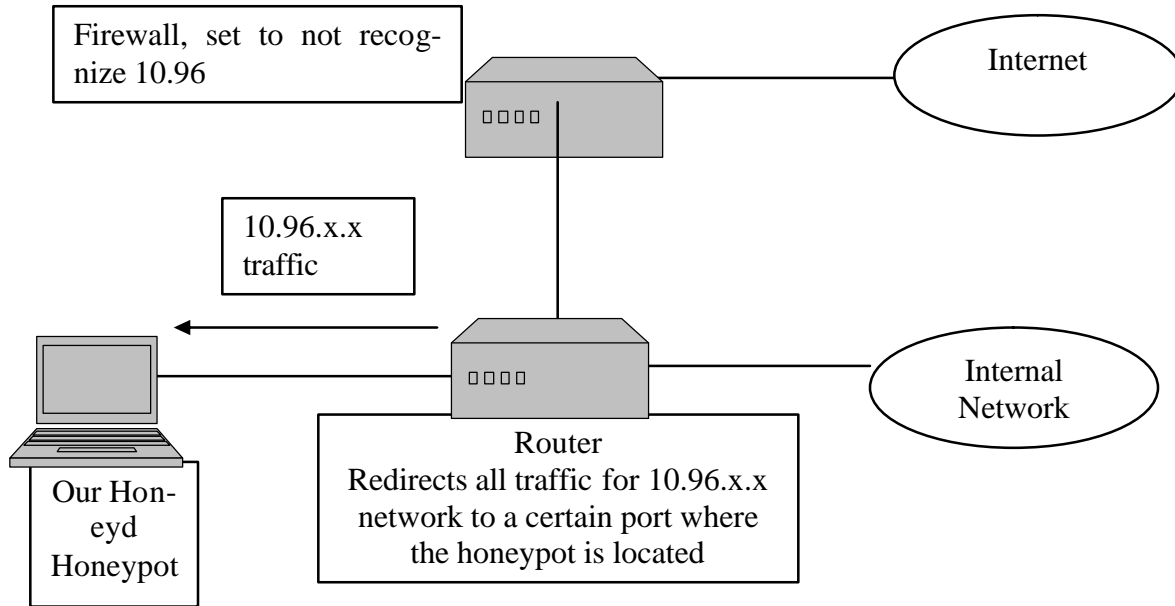


Figure 5: Simple Network Diagram with our Honeyd

This system will now emulate any possible operating system and service we want to program. For our tests, we will be emulating common services that are found on the internet and can become targets of hackers. These services will include a HTTP server, pop3 and SMTP mail services, and Telnet. While Honeyd will help us emulate these services, it won't, however capture the packets or the keystrokes of the attacker. For that purpose we must use a sniffer that will look at and capture all activity with the honeypot. For this, we will most likely use Snort, another open source piece of software that can capture all activity related to clear text-based connections such as FTP, HTTP and Telnet.

An interesting feature about Honeyd is that you don't have to use only one configuration. You can actually split up an entire network over any number of configurations by using the bind commands to tell Honeyd which template to use on what IP address. In our tests, we are setting only a certain number of servers on the initial setup. These will be the addresses from 10.96.2.3 through 10.96.2.23, making 20 servers available with the services listed above. The rest of the 10.96.2 network is going to be set with another template. By naming the template default, it will fill any addresses that Honeyd finds that nothing else is bound to. The default template will look read out:

```
create default
set default personality "Microsoft Windows XP Professional"
set default default tcp action reset
set default default udp action reset
set default default icmp action open
set default uid 32767 gid 32767
add default tcp port 1080 "/honeyd-files/mydoom.pl -l /honeyd-files/mydoom"
add default tcp port 3127 "/honeyd-files/mydoom.pl -l /honeyd-files/mydoom"
```

```
add default tcp port 3128 "/honeyd-files/mydoom.pl -l /honeyd-files/mydoom"  
add default tcp port 10080 "/honeyd-files/mydoom.pl -l /honeyd-files/mydoom"
```

This will set the virtual hosts to look like standard Windows XP Professional machines with default actions of resetting connections over tcp and udp, while leaving icmp open. I left icmp open so we can ping the machines and still receive a response. The tcp ports left open are meant to replicate the vulnerability exploited by the myDoom worm. Once a system connects to our honeypot using any of these hosts on these ports, it is sent to a script that logs where the probe came from, what time, and at which virtual host it was directed.

To get Honeyd to install, you must first install three libraries it depends on, libevent, libdnet, and libpcap. These libraries replace some system settings in Linux to make sure packets are transmitted correctly for Honeyd to work. It also requires a perl compiler to get perl scripts to run correctly. A final element necessary in getting Honeyd to run properly is a program called Arpd. Arpd will reconstruct the arp tables the Linux kernel uses to determine what packets it should handle.

Once these are installed and the configuration file created, there are only a few commands to run in terminal. For our first run with Honeyd, we enter `arpd 10.96.2.0/24`. This recreates the arp table to let the system take in any data sent to any IP address on the 10.96.2 network. Next, we need to add a routing reference to tell Linux where to route these packets. By running the command `route -n add -net 10.96.2.0/24 eth0`, we tell Linux to take any data sent to Ethernet adapter 0 on the 10.96.2 network. Next, to start Honeyd, enter the command `honeyd -p /honeyd-files/nmap.prints -f /honeyd-files/honeyd.conf -l /honeylogs -u 00000 -g 00000 10.96.2.0/24`. This will tell Linux to run Honeyd with a certain nmap file, what configuration file to use, where to save the logs, that the user in the process log should be root, and the network to watch. Finally, to start Snort to capture all data coming down the pipe, run the command `snort -de -l /log -c /snort-2.1.1/rules/snort.conf`.

Our Agenda for the Honeypot

We have two main agendas with our honeypot: hackers and virus detection. We're not sure how successful we will be at either of these agendas, as this whole project assumes that there is a threat. The threat that is not as likely, but more hazardous, is hackers. Our honeypot is set to log any and all activity going in and out of it, including keystrokes and remote addresses. We chose to emulate telnet, IIS, and ftp servers because those are ones that traditionally seem to get hacked. It seems doubtful that any network would be full of servers, which is why only 20 have been bound to IP addresses.

As far as virus detection, the other virtual hosts that fill up the network will allow us to view that activity. Our main concern at the moment is the myDoom worm that has been running across networks. All virtual hosts that are not servers are emulating machines that would be vulnerable to the myDoom worm. There is no risk here, since myDoom only affects windows systems and the ports are only open to do one thing: log. When a system connects to the virtual host on any of the ports vulnerable, a script runs that logs the day and time of the probe as well as the virtual host being attacked and the attacking system. This will give us an idea of who on the college network is infected so we can fix the problem before other systems are infected.

Honeypots have an interesting quirk when it comes to viruses that spread through worms or auto routers that depend on vulnerability. They can fix the vulnerability and remove the virus. It's not particularly hard either. All you need to know is the vulnerability, how to fix it, and how to remove the virus. Once you know these, you can write a small script that runs a few programs that will do all of this. Then, find out what ports the vulnerability relies on and set the configuration file of Honeyd so that any time it receives a probe on a port related to that vulnerability, it replies by sending the fix programs and script back to the attacking system through the vulnerable ports. From there, the system will update and fix itself from any future issues. There are some major legal issues at stake here, which we will cover soon.

Legal Issues

There are a number of legal issues concerning honeypots. In his book, *Tracking Hackers*, Lance Spitzner dedicates most of a chapter to discussing these legal issues, and Government Computer News Daily writer William Jackson echoes Spitzner's legal warnings in an article he wrote in 2002. First we should consider a right to privacy. Do people, hackers included, have a right to their data being private? There is no exact legislation over this matter yet, but current laws lean to the concept that any criminal trespass will negate a right to privacy by the intruder. Certainly, an attack on a computer system is a trespass. Does capturing the data about the hackers constitute an illegal search and seizure protected by the 4th amendment? This does not apply since the 4th amendment deals with searches performed by law enforcement. The Electronic Communications Privacy Act makes it illegal to intercept data and disclose it to others. With honeypot operators, however, we are capturing a copy of everything, and the ECPA provides legal groundwork for how to turn that over to the authorities if needed for prosecution, be aware though, there are specially provisions for honeypot operators that are Internet Service Providers.

The Wiretap Act makes it illegal to collect information streaming through systems. The Pen/Trap Statute makes it illegal to monitor the routing and addressing of that data. These two federal laws could make it sticky to operate a legal honeypot, but there are exceptions to the law. One is consent. Most services and systems have login banners when someone accesses the system. If you bury somewhere in that banner an agreement that states by using or accessing the system or even logging in, the user agrees to his activity being monitored, then you can be operating legally. Also, the laws provide for an exception when the monitoring is done to protect the property of the system provider.

The primary legal issue any honeypot operator needs to address is that of liability. The question will arise that if the attacker managed to break through your protections and take control of the honeypot, then does damage to a computer with that honeypot, can you be held liable in a lawsuit for the damages? This can be a very broad topic, especially considering all of the people involved. The natural response would be that the hacker is responsible. What if, however, the courts ruled that by operating a honeypot you invited the hacker in and are therefore responsible for his/her subsequent damage? Also, you have to consider the fact that this could all take place in different areas. The attacker could be in one state, you in another, and the person whom got attacked by your compromised honeypot in yet another. Now you have three sets of state legislation to deal with to decide who is responsible for what. Not to mention if this all takes place in

different countries, then you have federal and state laws in completely different areas to be worried about.

This is the reason why we decided to block our honeypot from the internet and focus only on the internal network issues. We are covered for our liability with student and faculty systems, since everyone agrees to an acceptable use policy before ever logging onto the network. This policy covers our monitoring the traffic into and out of the honeypot. This does not cover us damaging student systems, which is why we aren't activating anything on the honeypot to automatically fix and remove viruses from student computers. There is the remote possibility that something would go wrong, and we would be held accountable for the resulting damage. However, we are legally covered to monitor the information being passed through the network.

References

- 1) Spitzner, Lance. Honeypots: Tracking Hackers. Boston, MA. 2003.
- 2) Provos, Niels. A Virtual Honeypot Framework. Michigan, 2003. Last Accessed: April 22, 2004. URL: <http://www.citi.umich.edu/techreports/reports/citi-tr-03-1.pdf>
- 3) Cheswick, Bill. An Evening with Berferd: In which a cracker is lured, endured and studied. AT&T Bell Laboratories. Last Accessed: April 22, 2004
URL: <http://www.tracking-hackers.com/papers/berferd.pdf>
- 4) Jackson, William. Dangers in Luring Hackers with Honey. Government Computer News Daily. Last Accessed: April 22, 2004. URL: http://www.gcn.com/vol1_no1/daily-updates/19506-1.html