

Honeynets

Chris Brenton
Dartmouth College Institute for Security Technology Studies (ISTS)

ABSTRACT

Over the last year, network-based intrusions have increased exponentially, due to the popularity of scripted or automated attack tools. This increase in intrusions has rekindled interest in honeypot systems, which can be used to trap and decode the attack methods used by the black hat community. This paper will review the current state of honeypot technology as well as describe methods of deploying entire honeypot networks.

Keywords: honeypot, honeynet, network security, network intrusions, network defense, network protection

1. DEFINITION OF HONEYPOTS AND HONEYNETS

Within the realm of network security, a *honeypot* is a computer system that is specifically designed to capture all activity and files generated by a criminal who has gained unauthorized access to the system. While honeypots can perform a production service in support of the local environment (domain name services, mail, etc.), they are usually systems dedicated to the task of catching would-be perpetrators. Honeypots differ from regular networked systems in that greater attention is placed on logging all network activity. A greater level of access control in file manipulation may also be implemented. This allows the administrator to easily recover any intrusion tools the perpetrator may place on the system and then attempt to remove in an effort to cover their tracks.

Honeypots typically receive a different level of firewall protection than other network-based systems. For example, if a network is protected by a firewall, the honeypot would usually be placed outside the firewall or on some minimally-protected service network. This allows perpetrators on the Internet to receive full access to any services being offered by the honeypot. Remember that the idea is to record the perpetrator's activities, not to prevent them from gaining access to the honeypot.

There are four different methods of honeypot deployment. These include *Deception Services*, *Weakened Systems*, *Hardened Systems* and *User Mode Servers*. Each of these deployment methods are described below.

1. Deception Services

Deception Services are applications that are specifically designed to listen on an IP service port and respond to network requests like some other application. For example, a deception service could be configured to emulate Sendmail. When a perpetrator connects to port TCP/25 on the honeypot, they receive a banner that identifies the service as being some version of Sendmail. If the perpetrator is fooled by the deception and a Sendmail attack is part of their arsenal, they may attempt to gain access to the system through what they think is the Sendmail service. This allows the system administrator to log the particulars of the attack in an effort to safeguard their other systems that may actually be running Sendmail. This log may also be submitted to CERT, the vendor, or law enforcement for review in order to ensure that a proper fix can be created.

Deception services are probably the earliest form of honeypot deployment, Fred Cohen's Deception Toolkit being an excellent example.

There are a number of problems with using deception services for honeypot deployment. To start, it can be extremely difficult to emulate a service to enough of a level that a perpetrator will be fooled. For example, the perpetrator may try a variety of e-mail addresses and check for expected responses, as well as try a number of control commands. Unless the deception service is capable of passing this level of advanced testing, the perpetrator may become wise to the trap and never actually launch the attack.

Another problem is that a deception service is only capable of collecting a limited amount of information. You see the initial attack, which attempts to gain root access to the machine, but you see nothing else. It could be useful to see what the perpetrator would do if the attack were actually successful. A successful attack could yield additional information, such as other systems compromised by the perpetrator, clues to the perpetrator's identity, or even some of their tools. Since the

deception service should not provide the perpetrator access to the machine, additional forensic information cannot be collected.

Finally, it is theoretically possible that the deception service itself could have some form of vulnerability that could provide the perpetrator unexpected access to the system. For example, if I record all log entries on the honeypot itself, and a perpetrator finds a way to break the deception service, I could be caught off-guard, allowing the perpetrator to remove all evidence of their attack. Even worse, the honeypot itself could potentially become a tool in the perpetrator's arsenal for attacking other systems.

2. Weakened Systems

A weakened system is a honeypot deployment that typically uses a stock installation of some operating system that has known vulnerabilities. For example, I could install an older version of SunOS, knowing that RPC, Sadmin, mountd, etc. would be vulnerable to remote attacks. The theory behind this type of installation is that it is now easier for a perpetrator to break into the system, thus allowing us to collect data on the attack. Some form of external logging is usually implemented such as syslogd and/or an intrusion detection system in order to ensure that evidence is not lost if the perpetrator deletes log entries from the honeypot itself.

One benefit of the weakened system approach is that we are using the actual services that the perpetrator is trying to exploit. This solves the major problem with the deception service method that only allowed a limited amount of information to be collected. Once a perpetrator exploits one of the honeypot's services, we can continue to log their activity to see what else they may do. This could yield information on the perpetrators themselves, or the methods or tools they are using. For example, if we watch the perpetrator create a directory named /tmp/bob in which to store their tools, we now know we should check all of our production systems for this same directory. If the directory is found, chances are that system has been infiltrated in the same manner.

The only problem with the weakened system approach is that such systems tend to be high-maintenance with very little return. Having a perpetrator launch a three-year-old exploit against your honeypot is of little value to you if you already know of the exploit's existence and have already patched your production systems against this kind of an attack. In short, a weakened system is so vulnerable that you may end up analyzing and cleaning up many different attacks before you find one that produces cutting-edge (referred to as "o-day" in the black hat community) techniques or attack tools.

3. Hardened Systems

Hardened systems take the approach of a weakened system one step further. Instead of deploying a known-to-be-vulnerable system, the honeypot administrator will apply all known security patches to the base operating system in an effort to make each of the exposed services as secure as possible. The line of thinking here is that if a perpetrator is capable of breaking through a "thought-to-be-secure" service, forensic evidence collected from the attack will be useful for defense, as well as for law enforcement purposes. Deploying a hardened honeypot is the best way to ensure that maintenance time is minimized while the value of the data collected is maximized.

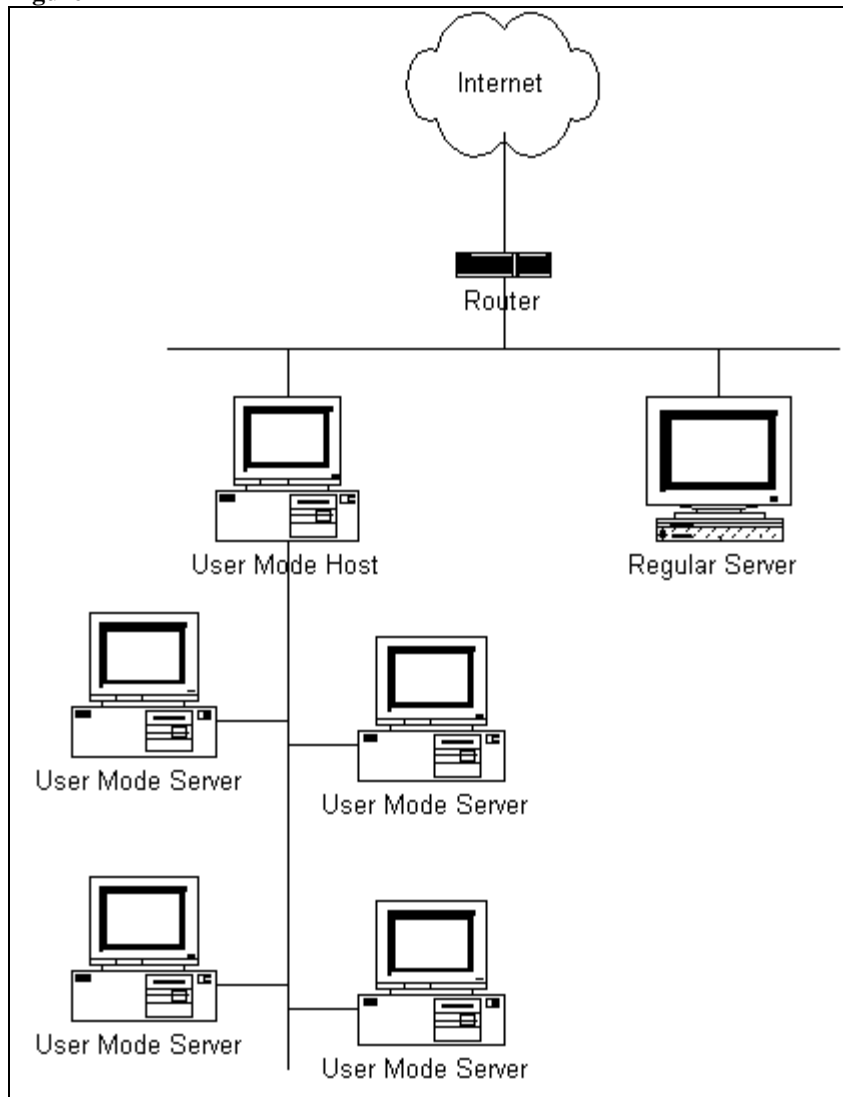
The only real weakness with using a hardened system as a honeypot is that this method relies as the system administrator being of the same or greater skill level than the person attempting to infiltrate the system. If the attacker has a greater skill level, then it's possible that they may be able to wrestle control of the system from the administrator, cover their tracks, or even worse, use the honeypot to launch attacks against other systems. I've run into a few situations where a highly-seasoned security administrator has had their honeypot infiltrated and has not identified the activity for a couple of days. While additional steps can be taken to ensure that the administrator does not lose control of the system (covered in the deployment section of this paper), it's a danger that must be addressed.

4. User Mode Servers

The use of user mode servers as a method of deploying honeypots is a relatively new idea. In short, a user mode server is a fully functional operating system running as a user process on a host machine. As an analogy, think of how you use a typical desktop system. From your desktop you may have a word processor, spreadsheet, and e-mail program all running at the same time. Now think of each of these applications as being a unique instance of an operating system with its own IP address and set of services and you'll get the idea. User mode servers are simply fully-functional servers that have been nested within the application space of the host operating system. When an Internet user transmits a request to the IP address of one of the user mode servers, the host accepts the request and routes it to the proper user mode instance.

Figure 1 shows how a user mode server deployment would look from a network perspective. Note that to a user on the Internet, the user mode host would appear to be some form of router or firewall. Each of the user mode servers would appear to be unique hosts running on a subnet located behind this router or firewall. Since it is not uncommon for hosts to be protected behind a firewall, this configuration would look perfectly normal to a would-be attacker. Note that this is only one possible implementation of user mode servers. It's also possible, using proxy ARP, to make the user mode host and servers look like they are all connected to the same logical network segment. This would allow you to hide your honeypots on a network segment that also contains production systems.

Figure 1



How well an attacker would be fooled would greatly depend on the implementation of the user mode servers. If they are properly configured, a perpetrator would have little to no clues that they are connecting to a user mode server rather than a standard dedicated server. This helps to ensure that the perpetrator does not suspect that their activities are being logged.

One benefit of user mode honeypots is that each of the user mode servers runs as a regular user process. This means that in order for a perpetrator to gain control of the machine they need to first break out of the user mode server and then find some way of elevating their privileges on the host system. This helps the administrator maintain control of the system when faced with an adversary of a higher skill level. This also aids in forensics and cleanup, since each user mode server is typically a single file located on the host system. To clean the compromised honeypot, simply kill the user mode server process on the host machine and launch a new file. When it comes time for forensics, simply transfer the user mode server file to another machine, launch the file, and logon and check out the file system.

Another benefit of user mode servers will become evident when we begin discussing the deployment of honeypots. In order to properly log and control perpetrators who attack the honeypot system, it's a good idea to deploy support systems such as a firewall, intrusion detection system and a remote logging server. This requires multiple servers and networking hardware to connect all the pieces together. With a user mode deployment, all these pieces can be deployed within a single computer.

The biggest drawback of user mode servers is that they are not available for every operating system. To create a user mode server you must start with a regular operating system and port it so that it can be run as a user application. At the time of this writing, there are user mode servers for Linux and NT, but not for some of the more obscure operating systems, such as HP-UX or AIX. This severely limits your operating system choices when deploying your honeypots.

So the concept of a *honeynet* is simply the deployment of multiple honeypot systems. This could be a dedicated network segment that contains only honeypot systems, or you could choose to spread out multiple honeypot systems across your entire environment. Remember that deploying honeypots is not all that different than playing roulette. You are betting that a perpetrator will infiltrate one of your honeypots prior to hitting one of your production systems. There are methods of increasing your odds, which I will cover in the deployment section of this paper.

2. WHY USE A HONEYPOT?

Some of the reasons for deploying honeypots were already touched on in the last section, so I will only cover them in brief here. Honeypots or honeynets should not be considered a replacement for any other security precautions such as firewalls or regular system audits. Rather, they are tools for augmenting existing security, as well as a means of learning the tools and tactics being used by the black hat community. I've personally been involved with honeypot projects that have been responsible for identifying and tracking a group of black hats responsible for many Web site defacements as well as the discovery of an (at the time) unknown Distributed Denial of Service tool.

One nice thing about automated attack tools is their indiscriminant evaluation of IP address space. This is a good thing from a honeypot perspective, as it means that a potential perpetrator is just as likely to check and attempt to penetrate a honeypot system as they are a regular production system. By meticulously recording all activity going to and from your honeypot, the odds are in your favor that any documented attack traffic is probably being sent to your regular production systems as well. As mentioned in the last section, this can act as an early warning system for the network administrator so they can more effectively identify what types of attacks are being logged against their network. This information can then be used to better protect their environment. As an additional bonus, if this information is shared with law enforcement and other security-minded professionals, it has the potential to minimize the damage caused by newly-developed methods of attack.

3. PROBLEMS WITH HONEYPOT DEPLOYMENT

The biggest problem with running a honeypot can be summed up in a single word: "Administration". It takes time and diligence on the part of the security administrator to ensure that any penetrations into the honeypot are quickly identified and control of the system is not lost to a perpetrator. While much of the process can be configured to use automated alerting, the security administrator must plan on spending at least an hour or two per day reviewing honeypot activity.

Justifying the time required to properly maintain a honeypot or honeynet environment to your superiors can be a difficult task. These tools provide no direct security protection, such as keeping intruders out of your customer database, so the high administration labor can be a difficult sell. If deployed properly, however, honeypots can actually result in lower overall

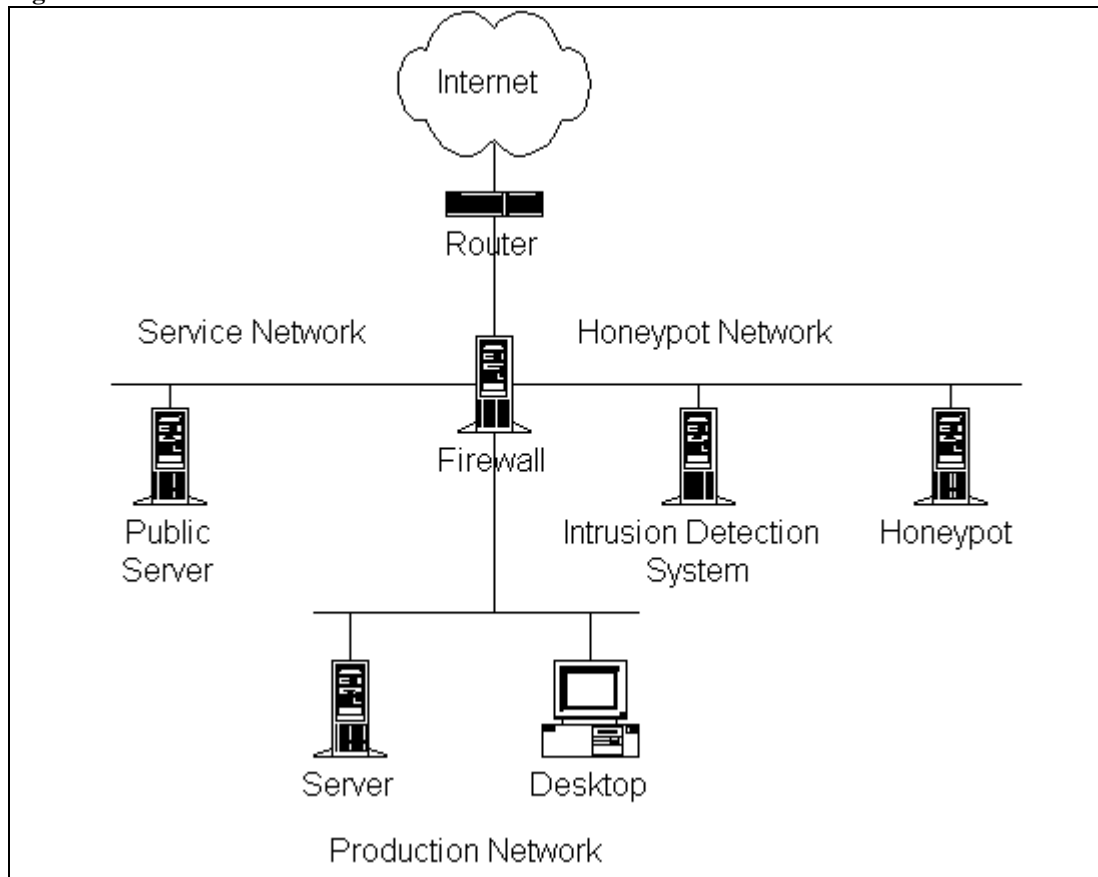
administration. Instead of meticulously reviewing all activity for every system on your network, you can do a peripheral review of your production systems while focusing most of your attention on the honeypots. If the honeypots are more exposed than your regular production servers, chances are that any attack activity directed at your network will first be seen on the honeypots. Keep in mind, however, that this is somewhat of a gamble and you are playing the odds that the honeypots will be hit first. This may not always be the case.

4. SUGGESTED DEPLOYMENT EXAMPLES

There are a number of things we will want to keep in mind when deploying our honeypot systems. To start, the systems should be isolated (when possible) from any production systems. This way if our honeypot becomes compromised, the attacker cannot use the system as a launch point for attacking the rest of our network. We will also want to place our honeypots as close to the Internet as possible. This way we do not need to expose networks that lie between the honeypots and the Internet. Additionally, we will want to take steps to log all traffic to and from the honeypots in such a manner that an attacker cannot delete log entries in order to cover their tracks. Finally, we will want to implement some form of firewall solution in order to control the traffic flowing to and from the honeypots.

Figure 2 shows an example deployment that meets all of the above criteria. We have added a fourth network card to an existing firewall in order to isolate our honeypot systems from the rest of our network. This also covers the requirement of having the systems as close to the Internet as possible. An intrusion detection system (IDS) has been placed on the honeynet in order to record all traffic flow. Since an IDS can record all packet information, including the contents of the payload, we should be able to log every step of a perpetrator's activities. Since logging is being performed on a system other than the honeypots, a penetrated system does not give the perpetrator access to log information. In fact most IDSs can function with no IP address bound to the system. This makes the IDS completely invisible to anyone monitoring network activity.

Figure 2



The firewall in Figure 2 not only isolates the honeynet from all production systems, but allows us to control all traffic to and from the honeypots as well. For example, we will probably want to allow unrestricted inbound access to the honeypots. While we will want to log all inbound traffic in order to correlate this information against the IDS logs, we want our honeypots to appear exposed to the Internet, so little inbound traffic control is recommended. If you wanted to get tricky, however, you could in fact filter out access to most inbound services. This would make it appear to a would-be attacker that the firewall is in fact attempting to “protect” the honeypot systems. This provides a much more realistic imitation of how most systems are connected to the Internet.

The only thing left is how to handle outbound traffic originating from the honeypots. Remember that our honeypots are just sitting idle. You should not see a communication session initiated from the honeypot unless a perpetrator has penetrated the system and is now using the host to contact another system. Obviously your firewall rules should prevent the honeypot from communicating with any internal systems. You may, however, want to allow the honeypot to communicate with hosts on the Internet. This allows the perpetrator to verify that they “own” the system, as well as access their tool archive in order to pull down a root kit. If we are looking to perform a complete analysis, we will want to allow the perpetrator to copy down and install the root kit so that it can be included in our analysis.

This raises a problem, however. What if the perpetrator copies down a Denial Of Service (DOS) tool as well as the root kit and proceeds to use the honeypot to attack other systems? Obviously we will want to prevent this kind of activity. The easiest way is to simply limit the number of outbound connections you allow to originate from your honeypot on a regular basis. For example, you could configure your firewall such that if 10 outbound connections originate from your honeypot in a 24-hour period of time, all subsequent outbound attempts are denied. This would provide the perpetrator with a sufficient number of connections to pull down their tools, while preventing them from using the honeypot as an effective platform for launching attacks.

One problem you may encounter is that many firewalls will not allow you to define a rule based on the number of daily connection attempts. If this is the case, you may be able to use an additional tool such as SWATCH to monitor the firewall log files and make dynamic rule changes when outbound activity is detected. Obviously, this ability is very firewall-specific, and is beyond the scope of this paper. The author has personally had very good luck implementing this type of functionality using a combination of SWATCH and Linux’s Netfilter firewall (sometimes referred to as IPTables).

So Figure 2 has all the pieces we need to run an effective honeynet. The only additional component you may wish to add is a syslogd server. This would allow you to send all log entries recorded on the honeypot to a remote system. This provides you with a backup of all of the system logs. While the IDS will record all traffic to and from the honeypot, it’s possible that system log entries could provide additional clues to the perpetrator’s activities.

The deployment discussed so far is fine if your main concern is outsider threats. Recent surveys, however, claim that up to 80% of all intrusions are launched from within a network’s perimeter. This raises the question, “Can honeypots be deployed internally in order to monitor for internal attacks?”

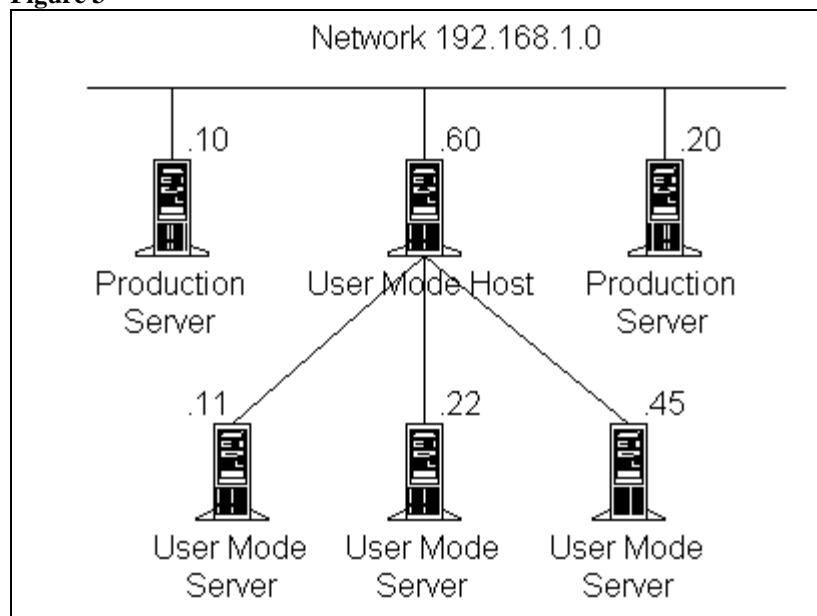
The dynamics change slightly when dealing with insiders. To start, an insider usually has some level of knowledge about which servers are on the network and what information is stored on each. This means that it is unlikely that an insider will stumble upon your honeynet and attempt to penetrate the system. The exception would be in the case of very large networks such as ISP’s or universities. The sheer volume of users in these environments increases the odds that a honeypot can be used as an effective security tool.

One of the problems with running internal honeypots is that if we isolate the systems on their own logical network, a perpetrator may be tipped off that these systems provide a special purpose and will ignore the systems. In order to deploy an effective internal honeynet, we need to meet the same guidelines as the network discussed in Figure 2, minus the requirement of being close to the Internet. We also need some method of spreading out the addresses assigned to the honeypots so they can appear at random points within our network.

A possible deployment is shown in Figure 3. In this example, we are leveraging the proxy ARP ability of a user mode setup in order to make the virtual servers look like they are a part of the main network. Note that all systems appear to be part of the 192.168.1.0 network. Also note that the IP addresses assigned to each of the user mode servers are spread out within the production network. This keeps us from having to group all of the honeypot IP addresses within a specific range, thus making

it more difficult to distinguish between the honeypots and the regular production systems. When a communication request is sent to one of the user mode servers, the user mode host takes care to direct the traffic to the appropriate application.

Figure 3



Note that in Figure 3 we have the ability to run both firewall and IDS software on the user mode host. This allows us to monitor and control all traffic headed to and from all of the user mode servers. Of course, the additional benefit is that we only need a single physical computer to host all of our honeypot software.

5. FUTURE EVOLUTIONS

It is clear that user mode servers hold great potential for honeypot and honeynet deployment. The ability to deploy what appears to be multiple systems on a single physical box allows a wide range of servers and services to be simulated. For example, most environments have a Web server, mail server, FTP server, and one or more DNS servers. Using user mode servers, this entire configuration can be simulated on a single computer. This not only reduces hardware costs when deploying honeypots, but provides a single point of administration as well.

As user mode servers are developed for honeypot use, they may very well find application within production servers as well. For example, a user mode server could be used to provide DNS for an organization. The user mode host could monitor all activity going to and from the DNS server. If an intrusion is suspected, the user mode hosts could shut down the user mode server, notify the administrator of the suspected intrusion, and restart a new "known to be clean" version of the user mode server running DNS. The number of potential back servers is only limited by the amount of disk space on the host machine. This would allow DNS services to continue to function while the administrator investigates how the perpetrator penetrated the system.

6. SUMMARY

Deploying a honeynet can be an excellent way to augment your existing security, as well as an excellent learning tool. While proper preparation is required prior to deployment, a honeynet can act as an early warning system, and can assist law enforcement in identifying perpetrators as well as collecting tools of their trade. Analysis of these tools can also lead to a better security posture for everyone attempting to protect their network.

7. ACKNOWLEDGMENTS

This work was supported by the National Institute of Justice.

8. REFERENCES

Lance Spitzner's Whitepapers
<http://www.enteract.com/~lspitz/>

Deception ToolKit
<http://www.all.net/dtk/index.html>

User Mode Linux
<http://user-mode-linux.sourceforge.net/>

vmware
<http://www.vmware.com/>

SWATCH
<http://www.stanford.edu/~atkins/swatch/>

Netfilter
<http://netfilter.kernelnotes.org/>

Snort
<http://www.snort.org/>