2012-06-11

Testing Deception Tactics in Response to Cyberattacks

Frederick, Erwin E.

Monterey, California. Naval Postgraduate School

Proceedings of the National Symposium on Moving Target Research, Annapolis, Maryland,

Testing Deception Tactics in Response to Cyberattacks

Erwin E. Frederick, Neil C. Rowe, and Albert B. G. Wong
Department of Computer Science
U.S. Naval Postgraduate School
Monterey, California, United States
ncrowe@nps.edu

Abstract—Deception can be a useful tool in defending computer systems against cyberattacks because it is unexpected and offers much variety of tactics. It is particularly useful for sites of critical infrastructure for which multiple defenses are desirable. We have developed an experimental approach to finding deceptive tactics for system defense by trying a variety of tactics against live Internet traffic and seeing what responses we get. These experiments are easiest to do on a honeypot, a computer system designed solely as an attack target. We report on three kinds of experiments with deceptive honeypots: one with modifying attack packets using Snort Inline, one with scripted responses to attacks using Honeyd, and one with a fake Web site. We found evidence of responses to our deceptions, sometimes in the form of increased session lengths and sometimes by disappearance of attackers. Some benefit was obtained by varying the deceptions over time. These results are encouraging for developing more comprehensive automated deception strategies for defending computer systems, and provide a new experimentation methodology for systematically developing deception plans.

*Index terms—*deception, cyberattacks, honeypots, tactics, Honeyd, Snort Inline, packets, scripts

This paper appeared in the Proceedings of the National Symposium on Moving Target Research, Annapolis, Maryland, USA, June 11, 2012.

I. INTRODUCTION

Attempts to provide a moving target for cyberattacks are often hampered by the limitations of honesty. If a computer system is to truly provide a variety of faces to attack, it would need to change itself frequently. Change is generally bad for computer systems. Bugs are more likely to occur when software and hardware changes, and testing is more difficult when there are many possible configurations. So real changes to computer systems are costly and to be avoided. It would thus seem advantageous to merely simulate change rather than actually do it.

Cyberattacks almost necessarily involve deception by an attacker on a computer system. This is because computer systems are built to withstand a wide range of attacks and only succumb to unexpected ploys. Since deception is equally useful to both sides in conventional warfare, deception might be useful in protecting computer systems as well [1, 2] as a form of active response [3]. A wide range of deceptive tactics are available to confuse defenders. Defensive deception generally raises fewer of the ethical concerns that offensive deception does because it is used for self-defense.

An appealing way to develop good deception tactics is to systematically test them out against live attackers. We can easily receive attacks just by creating a new Internet site. These will be predominantly automated attacks that tend to be predictable. So we can implement a variety of deception tactics to provide unexpected responses to upset attacker plans, and see how attackers respond. This could be a first step towards planning more elaborate deceptions such as with game theory [4].

II. PREVIOUS WORK

A. Honeypots

Honeypots are a good platform for implementing deceptions against cyberattackers as they permit considerable flexibility. A honeypot is a computer system set to detect, analyze, or in some manner counteract attempts of unauthorized use of information systems [5, 6]. The value of a honeypot comes from its unauthorized or illicit use because it is not designated as part of an information infrastructure and any interaction with it is suspicious.

Honeypots can be low-interaction (emulating services) or high-interaction (allowing mostly full access to operating systems). Honeypots can also be deployed as attack-discouraging or attack-encouraging systems [7]. As attack-discouraging systems, normally in an enterprise network, honeypots can serve to prevent, detect, bait and respond to attacks. As attack-encouraging systems, they serve to collect information on threats for analysis, study, and security enhancement. Honeypots can also be either physical or virtual. Physical honeypots are expensive to maintain and install, making them impractical to deploy for large address spaces. Virtual honeypots use one real machine to run one or more virtual machines that acts as honeypots, often with VMware and User Mode Linux. They can emulate multiple systems with different operating systems on a single hardware device.

Honeypots tools are available to monitor, log, collect and report intruder activity. Data can be system log files, firewall log files, keystroke logs, system snapshots, and packets from key events. To be useful, honeypot monitoring needs to function stealthily, so the intruder cannot detect it and defeat it. An example data capture tool used in honeypots is Sebek which intercepts specific system calls at the kernel level and takes sophisticated measures to conceal itself.

B. Anti-honeypot technology

Cyberattackers have reacted to honeypots by creating tools to detect or disable them [8, 9]. Techniques and tools useful for footprinting or analyzing systems can be reused or adapted. Some tools can detect possible features of a honeypot environment like virtual machines, keyloggers, and debuggers.

Most software used to build and run honeypots has distinguishable characteristics that give clues to attackers such as recognizable directory and file names [10]. User Mode Linux, VMware, and even Sebek can be detected this way. For example, it is possible to detect Sebek by measuring execution time of the read() system call; excessive delays in the execution of some processes or higher load than normal in a CPU are also good hints. Specific requests and responses to corrupted packets could give a clear warning to the attacker of the presence of Honeyd or more active responses like attempts to slow down attacks.

There is also commercial software available. An example is the Send-Safe Honeypot Hunter, a honeypot-detection tool. This tool opens a local fake e-mail server on port 25 (SMTP) and asks each proxy to connect back to itself. If the proxy claims that a session is OK when there is no related incoming session, a possible honeypot is detected.

C. Deception in honeypots

Researchers have begun realizing that honeypots can be more effective if they use various kinds of deception. Sophisticated redirection schemes can be used to foil attempts to recognize them [11]. And considering the dislike of cyberattackers for honeypots, fake honeypots could deceive attackers [12]. These are real production systems with the signatures, anomalies or behaviors of honeypots. An attacker using anti-honeypot/honeynet technology during scanning and reconnaissance may conclude that a system with these characteristics is a honeypot, and may avoid exploitation or more interaction with the system. So in defending a computer system with deception, we have two mutually exclusive options: Convince the attacker we do not have a honeypot when we do, or convince them we have a honeypot when we do not. The experiments we describe explored both options.

More generally, deception can keep attackers interested in a site and sending packets to it. An example is a fake Web site with fake information that deceives attackers on several levels [13]. Attackers can also be automatically diverted from neutral sites to honeypots [7]. Earlier experiments of ours showed that simple deception methods led to measurable differences in the interaction between an attacker and a honeypot [14]. However, these differences were quite general, and the experiments were insufficiently systematic to provide guidance for deception planning. That became the goal of the current work.

III. EXPERIMENT 1: PACKET MANIPULATIONS

We first conducted experiments with a high-interaction honeypot running Snort Inline, a variation on the Snort intrusion-detection system that permits limited forms of packet modification. We started from the honeypot setup used in [14] and made modifications. This was a computer system on the Internet outside our organization's firewall. It used VMWare virtualization to embed two victim systems within a real system. We directed it to modify single bits in incoming packets (TCP and UDP protocols only) using the Snort Inline product from www.snort.org. The hope was that since most computer commands are very sensitive to errors, this would not only likely make the attack ineffective but would provide unexpected challenges to the attacker which they could not address. Here is an example Snort Inline rule we used. This says to change the 50th byte of the incoming data portion(payload) of a TCP-protocol packet to an upper-case X.

```
alert tcp $EXTERNAL_NET any -> $HONEYNET any
(msg:"Exp2-TCP-offset-50"; content:"|00|";
offset:50; flags: P+; replace:"X";classtype:exp2;
priority:10; sid:9002001;rev:1;)
```

We compared attack durations with and without packet bit modifications and found that attacks with the modifications took longer on the average. Much of this was in the form of retries of the same commands. We tried systematic modifications of bits at different locations in the packets but did not see much variation in effect. We conclude then that rare random bit modifications will tend to waste attacker time. Byte modifications provide roughly the same amount of information as normal so the attacker cannot tell that anything is unusual with the packets from their sizes and other statistics. Because of the need for consistency with higher-level protocols, it is difficult with Snort Inline to increase or decrease the size of a packet, so byte modifications are the preferred tactic.

Deceptions also changed the frequencies of attacks we saw. Table 1 shows counts of Snort attack classes with first a control experiment (no packet modifications) and then changes to packets at offsets into the packet of 10, 20, 100, 20, and 20 respectively; the last two experiments changed the bytes to "W" rather than "X", and the last ignored the flags, to see if these things made a difference. Each experiment was run for two days. While the ICMP Ping and MS-SQL traffic was consistent through the experiments, and the FTP attacks (generally password guessing) were sporadic, the NETBIOS and SHELLCODE attacks show some effects in Experiments 1A, 1C, and 1E.

	Control	Exp. 1A	Exp. 1B	Exp. 1C	Exp. 1D	Exp. 1E
FTP	0	0	0	68794	0	3735
ICMP Ping	155	162	198	239	194	186
MS-SQL	48	32	34	50	44	30
NETBIOS	76	19	15	96	22	173
POLICY	0	2	1	0	0	1
SHELL-CODE	74	57	33	38	65	148
WEB	0	0	0	1	35	0

Table 1: Number of attacks during Experiment 1 with Snort Inline.

This methodology was thus effective at finding useful deceptions. These experiments did take time. Two days was a minimum to obtain useful frequency differences, and there are billions of potential deceptions. We could have speeded testing of different deceptions by deploying a large number of the honeypots simultaneously, but even then we would still need to run for two days to get a good sample of traffic on each. In fact, we should run for more than two days to get a good sample because additional random variations are added by differences in the machine addresses. This means it will be hard for this approach to develop deceptions quickly.

A more fundamental weakness of using this approach with Snort Inline, however, is that it is making rather crude changes at the byte level to the packets at a low level of protocol. Though it is possible, it is unlikely that random changes to bits will discover a good new kind of deception, any more than random changes to the bits of a program executable will create a better program. And even if we do discover better performance from a particular

byte substitution, it is hard to determine what principle is involved so that we could generalize. This suggests working with deceptions more at the application layer of network protocols.

IV. EXPERIMENT 2: DYNAMICALLY CHANGING HONEYD

Honeyd [15] is an open-source tool to deploy low-interaction honeypots. It allows a user to set up and run multiple virtual hosts on a network with services and specific operating systems running. It permits scripted responses of various kinds of cyberattacks, providing some opportunities for deception.

A. Honeyd

Honeyd (Honey daemon) allows a user to create virtual hosts to mimic several different types of servers and network configurations. The hosts can be configured to run arbitrary services, and their personality can be adapted so that they appear to be running certain operating systems. Honeyd enables a single host to claim multiple IP addresses. In this way, Honeyd deters adversaries by hiding real systems in the middle of virtual systems.

The daemon software offers several interesting features: It is possible to ping the virtual machines, or to run Traceroute on them to find their forwarding packets. Any type of service on the virtual machine can be simulated according to a simple configuration file. Instead of simulating a service, it is also possible to proxy it to another machine, even to the source. The different personalities simulate operating systems at TCP/IP stack level; this configured personality is the operating-system fingerprint that scanning tools like Nmap [16] or Xprobe would return. Although Honeyd is considered a low-interaction honeypot, it has powerful features to run services through scripts that could be configured to go beyond simple port listening and give responses to intruders. It can create virtual honeynets, and supports the creation of a virtual network topology including dedicated routes and routers. The protocols can simulate latency and packet loss to make the topology seem more realistic.

Honeyd software provides two types of logs that are useful to analyze. Network packet-level logging provides data of what kind of traffic the honeypots receive, and system logging provides host-centered information about the ongoing traffic. Honeyd can be used to either distract potential hackers or catch them in a honeypot. Either way, the hackers will be slowed down and subjected to analysis.

Honeyd software could be detected in several ways. One method is to flood a honeypot with pings or other CPU-intensive process. Then all other virtual honeypots running on the hardware will be slowed and this can be noticed. Another possible method is to time responses on the simulated systems since a honeynet will always be a little slower than a real system, and its times could be more consistent than on a real network where hardware is different for different machines. Another detection method is to analyze the responses of the machines to some uncommon packets and note discrepancies. For Honeyd, this happened previously when a TCP packet with SYN and RST flags set was probed to an open port; Honeyd sent a reply, and most other machines did not. Finally, Honeyd can be attacked through packet fragmentation. Honeyd checks the source address, destination address, and identification number of fragmented packets but not the protocol number. An adversary could send a carefully prepared fragmented packet mixing protocols that when reassembled by Honeyd could produce a reply packet or execute some attack, where a normal operating system would just discard them.

To prevent detection of Honeyd, countermeasures have been added to new versions of the software. For example, versions starting from 0.8 solved the clock timestamp problem by providing a different clock skew to each operating system and each virtual honeypot, and the wrong replies to TCP packets with SYN and RST flags are now patched.

B. Other tools used

Our experiments used VMware software, which provides a virtualized set of hardware to the guest operating system (info.wmware.com). VMware software virtualizes the hardware for a video adapter, a network adapter, and hard disk adapters to give the appearance of an x86 hardware platform. The virtualization capabilities of VMware simplify deployment of honeypots.

It is relatively easy to detect a VMware machine if one has access to the operating system. By default, the MAC address of NIC will be 00:0C:29, 00:50:56 or 00:05:69, the MAC addresses assigned to the vendor VMware by the IEEE, the MAC address will be immediately detected if the attacker is in the same network. The names of IDE

and SCSI devices (HD and CDROM) are clearly related to VMware: VMware Virtual IDE Hard Drive, NECVMWar VMware IDE CDER10 and VMware SCSI Controller. The PCI vendor string and device ID of video adapter, VMware Inc PCI Display Adapter, is visible. Finally, the I/O backdoor in port 0x5658 (22104 in decimal) that can be used to configure VMware during runtime is visible. However noting these features requires access to the operating system, something Honeyd does not offer.

To prevent an attacker from easily detecting the VMware machine or a virtual environment, the VMware binary file "vmware-vmx.exe" could be edited. Virtual IDE Hard Drive and Virtual IDE CDROM could be reassigned names more appropriate to hide the VMware application. In Linux, this can also be done automatically using scripts for patching. An appropriate configuration of the OS could prevent the VM system from being fingerprinted and detected. For example, we need to give each virtual machine enough main memory to be credible, for example 512 MB or above. This change could be done through the VMware Virtual Machine Control Panel. Also, we should change VMware MAC address because the default MAC address assigned by VMware always starts with 00:0C:29, 00:50:56 or 00:05:69. Operating systems or VMware provide a way to change the MAC address, but we need to be careful to match the numbers to an existing vendor that also is related with changed names of other devices.

Another tool used in our experiments was the Snort intrusion-detection system from www.snort.org. We used the latest VRT rules available free for registered users, which were an average of 30 days old when released. We also used the Wireshark packet analyzer (www.wireshark.org), the Microsoft Log Parser 2.2 [17], and Security Onion (securityonion.blogspot.com) for installing, configuring, and testing the intrusion-detection system. We used an operating system based on Red Hat Linux Fedora 14, the latest version available at the start of the experiments.

C. Experiments with Honeyd

We set up experiments on a small network at our school that is not protected by the organizational firewall, different from the network in Experiment 1. We ran a group of honeypots created with the mentioned software, and we tested them in different runs with different configurations, analyzing them week by week in trying to find the best ones for attack-discouraging and attack-encouraging purposes. We tried to test as many features of Honeyd as possible, like simulation of open, close and filtered ports, emulation of operating systems at the TCP/IP stack level, and service scripts associated to certain well-known ports. We also tested small details like changes in the MAC addresses, set drop rates, set uptime, and the use of proxy and "tarpit" capabilities to create a credible set of virtual machines. In general, we tried to maximize the interaction with possible attackers -- which we defined as maximizing the number, variety, and duration of attacks -- since then the honeypots are more successful and more difficult to detect or avoid.

Experiment 2 was run with one laptop running Windows XP as host; it also ran first a Fedora 14 Virtual Machine and later a Security Onion VM, both using VMware, as guests. We used seven IP addresses for this experiment, all of them monitored and working as honeypots. We used a hub to connect to the network because the experiment coexisted with other tests and honeypots. One of the latter is a real host running Windows XP which can be considered part of our experiment, although it is production system. On the Windows machine we installed Snort 2.9, configured to log and create Tcpdump files for the alerts which could be read with Wireshark. On the Fedora virtual machine we installed Honeyd 1.5c that generated its own packet and system logs. At the beginning of the experiment and between every run, both machines were updated and patched to harden them against possible attacks. Snort was also updated when a new set of rules became available.

D. Configurations tested

Experiment 2 tried different configurations of Honeyd, each which ran for approximately one week. In general, we went from simpler to more complex; we discarded configurations that were unsuccessful judged by the amount of traffic and number and variety of alerts; we included some random changes to provide variety; and we included control experiments without Honeyd.

- In week 1 we ran only the host computer with Snort to get a baseline for the normal traffic.
- In week 2 we ran Honeyd in the guest virtual machine, mistakenly by default, for a long weekend. Honeyd claimed all 32 addresses of the network from other users (Honeyd is aggressive in capturing IP addresses) and the Ethernet connection was closed by our information-technology department. We discarded data from this week.

- In week 3, we ran Honeyd on the guest virtual machine claiming five addresses: three simulating Windows hosts and two simulating Linux hosts. Each host had two open ports for the operating system, for example NetBios ports (137, 138, 139) for Windows hosts or to certain services, like port 80 to HTTP or port 25 to SMTP. We added a virtual machine running Honeyd and one as the host for VMware.
- In week 4 we configured a more elaborate deception that includes simulated services (some included in the software for Honeyd and others downloaded from the web page). In addition, we kept most of the open ports and used a more credible ports status. Two Windows OS computers were simulated and three Linux OS. Every computer had several TCP and UDP open ports, and some emulated services like SMTP, HTTP, FTP, SSH, telnet, NetBIOS, POP3, IMAP. We also used the proxy function of Honeyd in a couple of ports to redirect the attacks to its source.
- In week 5, we used a similar configuration to the previous week but without the "production" host.
- In week 6, after noticing a decrease in the traffic and number of alerts of the previous configuration, the IP address of the VM host was switched with that of one of the honeypots, and this machine was changed to a Security Onion suite instead of Fedora 14 because it was supposed to have better capabilities of monitoring and to be hardened against possible attacks. The rest of the configuration was similar to the previous week.
- In week 7, although the number of alerts had not increased significantly, we continued with a similar configuration as the previous week with several scripts running on each host.
- In week 8, due to the clear difference in the amount of interactions with emulated Microsoft Windows and Linux operating systems, we emulated only Windows. Analysis of the results showed us that not-credible emulated services were worse than simulated open ports. Hence we set only open ports with no services running.
- In week 9 we continued with a similar port configuration using four of what appeared to be the most successful scripts.
- In week 10 we ran without Honeyd to check again the normal behavior of the network.
- In week 11 we ran without Honeyd and even without the guest virtual machine (like week 1 but with better tools for analysis), to check again the traffic in the network.
- In week 12 we ran Honeyd with the same configuration as week 9.
- In week 13 we continued the same previous configuration adding a Perl script with a fake Telnet server in port 23 on host .79 and using the new created personalities for Windows Server 2008 and Windows XP Service Pack 3.
- In week 14 we did another control run without Honeyd running.
- In week 15 we used the same configuration as week 13 but we replaced the fake Telnet server with a fake internal web server in maintenance status and we included the proxy function in port 445 in one honeypot, to send back to the source everything the virtual host receives in this port. Also, we switched the IP addresses of four honeypots.

E. Methodology for analysis of results

As we learned what worked and what did not, we used different logs, scripts, tools and software to better analyze the information captured. This caused changes in the methodology and log formats, so there was more work and information available toward the end of the experiments.

Every week we collected the Snort summary (displayed on the screen) in a text file, Snort alerts in CSV format, Snort alerts in PCAP format, Honeyd packet logging in text format, and Honeyd system logging in text format. We used some code samples from Giuseppini [5]. We created a batch program using Microsoft Log Parser which generates several files and folders by calling six SQL query scripts that produce results in HTML template format. The results of the queries are details of the alerts, sources, and destinations, and indexes to this information. We created another batch program for Log Parser that generates graphs for several kinds of information: top alerts, top source IPs, top destination IP, alerts per hour, top source port, top destination port, and top protocol. To analyze Honeyd logs, we created a script that parses the text file "honeyd.log" related to Honeyd packet logging and produces

a file useful for seeing how relatively effective the honeypots were in attracting traffic so we could compare configurations.

Attackers could easily scan our experimental network from outside to identify its ownership using Traceroute or similar tools. However, we saw no evidence of scanning. Foreign adversaries executing a state-sponsored cyberattack would be interested; this information could deter some of their attacks or encourage them, depending on the identity of the site. But concealment of a site's ownership is not difficult if it is desired.

F. Overall statistics

Analysis from the data obtained in weeks 1, 10, 11 and 14, without Honeyd running, estimated the baseline behavior of the network, the normal network levels of traffic and alerts. Table 2 shows the results. We saw a 40% increase in the number of packets in the network when Honeyd was running. Also, the total number of alerts increased several times and the number of different alerts was greater. Given these statistics, we can say that Honeyd generally increases both the amount of traffic and the amount of malicious traffic on a network.

Another way to analyze how successful Honeyd and its different configurations are at changing attacker behavior is by comparing the number of conversations (the traffic between two specific endpoints) produced by Honeyd, as obtained from its packet log. The numbers for weeks 4-13 were 46124, 14078, 5226, 13676, 26827, 49818, 40950, and 35244. So what we did in weeks 5-8 substantially discouraged attacks. We saw that some honeypots had significantly more conversations than others. For instance, in week 4, honeypots 73, 74, 77, and 79 had 99, 121, 371, and 94 ICMP packets respectively; in week 6, 5, 68, 0, and 118 respectively. This illustrates why it is important to run deceptions on a number of machines to decide which are most effective, because target choice by attackers is often multistage where first a machine is selected, then a considerable number of attacks are tried against it before the attacker tries another.

G. Snort alerts

During the experiment Snort generated alerts for the attacks. The most common alerts were related to NetBIOS. These alerts appeared in bursts and only in half of the weeks the experiments were running. An important alert that appeared almost every week was the Shellcode NOOP. This alert was associated with several different Snort signature identification numbers, which means different types of buffer overflow attacks were injecting null operations. Other important alerts were the attempts to connect to the hosts with the remote desktop protocol and constructed bit-string heap corruption.

The ports less than 1024 that were most probed and attacked were 445, 80, 135, 139, 53 and 22. Port 445, Microsoft-DS (Directory Services) Active Directory, was by far the port most attacked during the experiment, with 95.32 % of the TCP protocol alerts. In a distant second place appears port 80, the HTTP (Hypertext Transfer Protocol) port with 3.25% of the same kind of alerts. Also appearing were port 135, Microsoft End Point Mapper (also known as DCE/RPC locator service) with 0.84% and port 139, NetBIOS Session Service, with 0.34%. Thus it is recommended that port 445 be open in any Windows honeypot configuration, and also ports 80, 135, and 139.

Week	Honeyd running?	Number of packets	Number of alerts	Different alerts	ICMP alerts	TCP alerts	UDP alerts
1	No	438661	388	4	388	0	0
3	Yes	1191410	8589	24	8366	2185	5
4	Yes	1313693	259776	36	255744	4016	16
5	Yes	701771	2525	12	1940	584	1
6	Yes	906893	2823	17	2176	647	0
7	Yes	740769	6686	11	2990	3696	0
8	Yes	897552	3386	14	2144	1242	0
9	Yes	951556	2957	19	2651	306	0
10	No	618723	1325	13	757	568	0

11	No	541740	756	16	476	270	10
12	Yes	995235	2526	10	2270	256	0
13	Yes	807712	3711	15	3445	266	0
14	No	518659	488	5	488	0	0
15	Yes	1066743	4694	14	3082	1612	0

Table 2: Overall alert statistics for Experiment 2.

Given that the operating systems emulated by Honeyd were not detected correctly by fingerprinting tools like Nmap, we cannot say that emulated Windows hosts received more attacks than emulated Linux hosts because of their operating system. Rather, the port most attacked, 445, is a Windows operating-system service.

One of the most interesting features of Honeyd is its capability to run scripts associated with ports as specified in the configuration file. These scripts can provide a variety of responses to make it harder to attackers to recognize the honeypot. The degree of success of our scripts was wide due to two factors. First, there was some effect related to the appearance of "new" services (something similar occurred with "new" hosts) that were quickly probed and scanned, illustrated in the large number alerts in week 4. Within a short time, between one and two weeks depending on the script, this novelty effect was lost, as we can see in weeks 5 and 6. Similar phenomena were observed in our first work with honeypots [14] where there was considerably more variety in the attacks in the first few weeks a honeypot appeared on the Internet. We conclude that a script will have a decreasing degree of success if it is continued past a few weeks.

Secondly, services that were not credible because they were old, had errors, or were not logical according to the host's configuration were quickly recognized by attackers who then avoided the honeypot, as we can see in week 6 where the same configuration was running for three weeks. This identification process took less time compared with the novelty effect. Hence it was better to keep the port open instead of running a uncredible script, as in weeks 8 and 9 (weeks with a small number of scripts but many ports open), for both attack-encouraging and attack-discouraging purposes.

It is true that service scripts could make an operating system running Honeyd more vulnerable to attacks, but we saw no evidence of such exploitation in our experiments, either in the host or the guest operating system. The routine updates done to the operating systems and antivirus software probably helped.

H. Honeyd as an attack discourager

To analyze the usefulness of the Honeyd framework as an attack manipulator, we can compare the alerts in production hosts (normal users of the network) with the alerts in the honeypots. This includes the Windows XP running Snort and VMware and the virtual machine running Honeyd. We sought evidence that a significant number of alerts migrated from production hosts to members of the honeynet.

Table 3 shows the percentage of alerts in dividing honeypots into the host running Windows XP and VMware, the guest running Fedora 14 or Security Onion and the honeypots created by Honeyd. We saw a significant decrease in the percentage of alerts on production hosts with Honeyd running. There is a notable exception in week 4, where a new configuration with many ports open and dozens of service scripts running was very interesting for attackers, and it was repeatedly probed and attacked. So week 4 was good for attack encouragement. The rest of the weeks appeared promising for attack discouragement, showing that more than 80% of the alerts occurred on honeypots instead of production systems.

Week	Host machines	Guest machines	Honeyd honeypots	Production hosts
3	72.7	16.8	6.1	4.4
4	4.3	1.5	1.6	92.7
5	9.8	38.9	41.1	10.2
6	12.4	56.4	21.9	9.3
7	4.2	48.6	42.4	4.8
8	7.3	30.9	50.2	11.6

9	10.2	11.0	62.8	16.0
12	10.6	26.7	53.5	9.2
13	8.4	40.8	43.0	7.8
15	4.6	49.9	40.0	5.7

Table 3: Percentage of alerts in Experiment 2 by week and type of machine.

I. Discussion

Honeyd increased about 40% the amount of traffic in the network, and by several times the number of alerts to be studied. It simulated well the status of the ports (open, closed, or filtered) and appeared to associate them properly to scripts. However, Honeyd failed to create virtual hosts with credible emulated operating systems because it was designed for the first-generation Nmap software and has not been updated. The attempts to mitigate this problem were only partially successful; therefore, we had to rely only in the port configuration to simulate different operating systems. This is a critical weakness of Honeyd that needs to be fixed.

Scripts varied in their effectiveness. Some were only initially effective at attracting attackers and then lost their effectiveness in subsequent weeks, but others continued to be interesting. It is likely that the less successful scripts were insufficiently elaborate and could not provide enough of the right kind of responses to encourage attackers to probe further. The low numbers of attacks during weeks 5-8 suggest that the configuration then was a good one for scaring away attackers, and was thus a useful protective deception.

The ports most attacked should be supported in any honeypot configuration for an adequate level of interaction with attackers. Ports 445, 135, and 139 are related to the Windows operating system, and hence produced more attacks for virtual hosts simulating Windows than the Linux operating system. The more common attacks received in the honeypots were related to NetBIOS and to Shellcode NOOPs (buffer overflows). We did not find symptoms or evidence of successful attacks or compromise in the host running Windows XP or in the guest system running Fedora 14 or Security Onion Ubuntu. In general, we found that the honeypots received only basic and not advanced attacks.

V. EXPERIMENT 3: A WEB SERVER

The previous experiments suggested that it would be especially useful to simulate Web pages since attackers seemed interested in them even in the absence of a Web server, and HTTP responses are not difficult to configure. We adapted the “iis.sh” web service script written by Fabian Bieker for interactions with the adversaries, which we chose over “iisemul8.pl” because it is smaller and easier to understand and modify. It was important before these experiments to reduce protection violations, a key difficulty in using Honeyd. This required some tedious search and manual modification of directory properties.

We developed a dynamic homepage for the honeypot to keep intruders interested. It has three frames presented in two rows. A left column consists of a number of hard links designed to keep the interest of the attacker to explore the variety of resources. The right column is the view frame where the selected resource is displayed. When the homepage is requested, three pages are automatically downloaded to the browser -- label.html, nps.html, and research.html -- to populate the frames. Additional pages such as faculty.html, staff.html, student.html, community.html, and event.html are links to fill the view frame when selected.

We also varied the errors returned on HTTP request, hoping to confuse attackers and entice them to respond differently from otherwise. This is done by adding a simple shell script named code_sequencing.sh that runs concurrently with Honeyd. This script defines an array containing a few selected error codes and steps through the array in a circular fashion. At each step, a loop index containing the error code is written into a file called httperr.txt. It sleeps for an arbitrary number of seconds (the default is one). The httperr.txt file is then read by iisnew.sh and posted as the return code.

As before, we ran Snort on Windows XP to capture attack packets and IDS alerts and we ran honeyd on Xubunte. We used honeyd15.conf and modified the configuration file in simulated Windows NT4 web server to run the iisnew.sh script at port 80. Microsoft Log Parser 2.2 was used for analyzing the study results. Two programs process Aerts.csv from Snort and one processes honeyd.log from Honeyd. Processing was done on a separate system.

The report from honeyd.html showed the distribution of connections shown in Table 4.

Honeypot address	Hits	Configuration
..*.77	15,412	2003 Server with ftp and smtp service scripts
..*.70	9,900	Windows XP with no servicescript
..*.74	9,626	SQLserver with standard iis.sh script
..*.73	7,963	NT4 Web Server with iis.new.sh script
..*.79	7,668	Windows Xp with no service script

Table 4: Virtual machines used in Experiment 3 and their traffic volume.

Table 5 shows comparative results over time and rates per day. In December, we established a connection with one Web page; in January, we added Web pages; and in February, we sent back various HTTP error messages part of the time. It can be seen that the error messages in February definitely increased both the rate of interaction with the honeypot and the rate of attacks on it. This was surprising since it appears that this attempted discouragement of attackers may actually encourage them. But on the other hand, the overall rate of Web interactions went down in February in response to the increased number of error messages.

	December	**January**	**February**
Number of days running	12	21	9
Rate of all Honeyd log entries	3389	3913	5695
Rate of Snort alerts	176	208	1069
Rate of all Web log entries	16.1	74.1	30.9
Rate of GET commands	5.7	34.9	10.2
Rate of OPTIONS commands	8.9	35.9	18.7
Rate of HEAD commands	1.4	2.2	1.1

Table 5: Rates per day for the fake Web server in Experiment 3.

Altogether, Web page requests were 3.3% of the total traffic in the last five days, so most attacks were on other targets than the Web pages. Apparently most of the "background radiation" of attacks is indiscriminate about matching its attacks to software targets. We could not tell yet if attackers were responding differently to different error codes, so we are continuing to test this now.

The strings received as arguments in Web page requests were quite interesting as indicators of what attackers were up to. Table 6 shows the most common ones appearing in either raw arguments or in email addresses with site address that of the honeypot. These names should provide excellent sources for preprogrammed responses – if attackers want to use an argument "yes", we should give them something different when they do, and this will likely keep them interested for a longer time. Similarly, if they keep asking to talk to test@honeypot.nps.edu, that address should respond.

String	**Count in February**	**User Name**	**Count in February**
yes	717	test	98
no	492	info	76
admin	98	admin	76
test	61	sales	46
info	41	web	36
david	18	contact	32
michael	18	postmaster	32
mike	15	office	24

richard	15	spam	24

Table 6: The most common strings received in Web page requests in February of Experiment 3.

Similarly, we found particular Web pages that attackers were repeatedly asking for (Table 7). These should be prime targets for future development of fake pages.

Page	Count in February	Page	Count in February
//phpmyadmin	27	//admin	4
/	8	/w00tw00t.at.ISC.SANS.DFind:)	2
//pma	6	//sql	2
//mysql	5	//phpmanager	2

Table 7: Web pages that occurred more than once in GET requests in February of Experiment 3.

This suggests a methodology for letting attackers themselves guide us in developing deceptive Web servers: Record what attackers are asking for in GET commands, give it to them as constructed pages, and repeat. If the result is too top-heavy with administrator pages, we can always supplement it with some real Web pages. PUT and OPTIONS commands are ignored by many servers, so a deceptive server need not do anything with them.

VI. CONCLUSIONS

From these experiments, it is clear that honeypots can be configured to serve either to encourage or discourage attacks, but careful design of their configuration is important to obtain either effect. For instance, configurations with a sufficient number of open ports and running scripts were useful as encouragement but not as discouragement; they also needed to have a limited number of scripts and open ports, enough to make them only mildly attractive while avoiding making their entire local network attractive to attackers. But all deceptions, as in conventional warfare, have a limited useful life, and our experiments suggest they need to change at least every two weeks.

Our experiments showed we have much work to do before we can use, as is hoped by some, game theory in real time to create deceptions for attackers. Cyberattacks are bursty and inconsistent phenomena, and using them for systematic configuration development requires a significant amount of data. Our honeypot network was insufficiently large to catch all the phenomena of interest in a reasonable time window, as honeypot "farms" are necessary. Nonetheless, our experiments demonstrated that careful testing of attacker responses can be very useful in designing deceptions for an Internet site, and had definite advantages for learning to manipulate attacks over a passive approach of choosing one configuration and staying with it.

VII. ACKNOWLEDGMENT

This work was supported in part by the National Science Foundation under grant 0429411 and by the Air Force Research Institute. The views expressed are those of the authors and do not represent those of the U.S. government.

VIII. REFERENCES

[1] N. Rowe, "Deception in Defense of Computer Systems from Cyber-Attack," chapter 13 in L. Janczewski and A. Colarik, Cyber Warfare and Cyber Terrorism, pp. 97-104, 2007.
[2] S. Vadialis and Z. Kazmi, "Security through Deception," *Information Systems Security, (16)*, 1, pp. 34-41, January 2007.
[3] M. Rash, A. Orebaugh, G. Clark, B. Pinkard, and J. Babbin, Intrusion prevention and active response. Rockland, MA: Syngress, 2005.
[4] G. Wagener, R. State, A. Ulaunoy, and T. Engel, "Self-Adapative High Interaction Honeypots Driven by Game

Theory," Proc. 11th Intl. Symposium on Stabilization, Safety, and Security of Distributed Systems, pp. 741-755, Berlin: Springer-Verlag, 2009.
[5] R. Grimes, Honeypots for Windows, Apress, Berkeley, CA, 2005.
[6] The Honeynet Project, Know Your Enemy: Learning about Security Threats, Second Edition, Boston, MA: Addison-Wesley, 2004.
[7] N. Provos and T. Holtz, Virtual Honeypots, from Botnet Tracking to Intrusion Detection, Boston, MA: Addison-Wesley, 2008.
[8] B. McCarty, "The Honeynet Arms Race," IEEE Security and Privacy, pp. 79-82, November/December 2003.
[9] N. Krawetz, "Anti-Honeypot Technology," IEEE Security and Privacy, pp. 76-79, January/February 2004.
[10] T. Holz and F. Raynal, "Detecting Honeypots and Other Suspicious Environments," Proc. 6th SMC Information Assurance Workshop, West Point, NY, pp. 29-36, June 2005.
[11] L.-M. Shiue and S.-J. Kao, "Countermeasure for Detection of Honeypot Deployment," Proc. Intl. Conference on Computer and Communication Engineering, Kuala Lumpur, Malaysia, pp. 595-599, 2008.
[12] N. Rowe, E. Custy, and B. Duong, "Defending Cyberspace with Fake Honeypots," Journal of Computers, 2 (2), 2007, 25-36.
[13] N. Rowe, "Measuring the Effectiveness of Honeypot Counter-counterdeception," Proc. 39th Hawaii International Conference on Systems Sciences, Poipu, HI, January 2006.
[14] N. Rowe and H. Goh, "Thwarting Cyber-Attack Reconnaissance with Inconsistency and Deception," Proc. 8th IEEE Information Assurance Workshop, West Point, NY, pp. 151-158, July 2007.
[15] N. Provos, "Developments of the Virtual Honeyd Honeypot", www.honeyd.org, retrieved July 1, 2011.
[16] Nmap.org, "TCP/IP Fingerprinting methods Supported by Nmap", nmap.org/book/osdetect-methods.html, retrieved July 8, 2011.
[17] G. Giusepinni, Micrrosoft Log Parser Toolkit, Syngress Publishing, 2005.