Defending Cyberspace with Fake Honeypots

Neil C. Rowe
U.S. Naval Postgraduate School, Code CS/Rp, Monterey, California, USA
Email: ncrowe at nps.edu

E. John Custy and Binh T. Duong
U.S. Navy SPAWAR Systems Center, San Diego, California, USA
Email: custy at spawar.navy.mil, binh.duong@gmail.com

***Abstract*—Honeypots are computer systems designed for no purpose other than recording attacks on them. Cyber-attackers should avoid them since honeypots jeopardize the secrecy of attack methods and it is hard to launch attacks from them. This suggests that a computer system might pretend to be a honeypot to scare away attackers, reducing the number of attacks and their severity. This could be done on ordinary computer systems as a kind of "vaccination" of those systems, to create what we call "fake honeypots". After some background, we examine this idea from three perspectives. We develop a mathematical model of what would make an attacker go away. We report experiments with deliberate distortions on text to see at what point people could detect deception, and discover they can respond to subtle clues. We then report experiments with real attackers against a honeypot. Results show that attacks on it decreased over time (which may indicate that attackers are being scared away), irregular outages of the honeypot stimulated attacks, and other changes occurred in response to our manipulations. We conclude with some speculation about the escalation of honeypot-antihoneypot techniques.**

***Index Terms*—honeypots, deception, intrusion-detection systems, defense, signatures**

I. Introduction

Honeypots are computer systems designed to be attacked to enable defenders to learn about attack methods [1, 2]. When they are connected to the Internet, any users other than their system administrator are unauthorized by definition and probably malicious. They yield much richer log and intrusion-detection data for attacks than is possible by monitoring ordinary computer systems or networks. Honeypots are often connected in networks called "honeynets" to see how attacks exploit networks.

Unfortunately, smarter attackers are now becoming aware of honeypots, as mentioned in the Honeynet Project's book [1]. They have been observed trying to avoid honeypots, as they should for several reasons [3]. First, their activities will be monitored on a honeypot, and likely in several different ways so it will be difficult to escape monitoring. Attackers do not like monitoring because their activities are generally illegal. They also jealously guard their attack methods to gain status with other attackers and to preserve them for critical moments. Second, attackers want to avoid honeypots because attacks on them are unlikely to achieve much. Honeypots are designed by people that are knowledgeable about security threats. While honeypots appear easy to subvert, important features like logging facilities are not. Careful controls generally prevent propagating the attack to other machines, and the number of machines subverted is an important measure of the success of a hacker. So since honeypots are rare, attackers finding one should do better attacking other machines.

Although most attackers who discover a honeypot will go away, a few may want to stay and fight it. Since deception often stimulates emotions, they may like a challenge, or they may want revenge, or they may be angry [4, 5]. But then the honeypot is still useful because disabling the monitoring is hard, and in the meantime the attacker ploys will be recorded. So hackers that overreact to honeypots can also benefit the defender.

A simple defensive ploy on the usual attacker who is predisposed to avoid honeypots might be to make ordinary systems appear to be honeypots ("fake honeypots"). Unlike most security measures, this would work best against smarter attackers, and could be an effective defense in asymmetric warfare [6] against surprise attacks. Attackers usually deceive victims in some way, so defensive deception in return would appear to be ethical. We could plant clues in the secondary or primary storage of a system that would suggest a honeypot, like names of known honeypot tools, nonstandard system calls in key security-critical subroutines, and mysterious running processes; and we could reduce the number of "normal" files, by storing them elsewhere, so the system appears little-used. We could also make a system respond in ways typical of honeypots, such as dropping or modifying packets. These clues could provide a kind of "vaccination" for computer systems that would reduce the length and severity of the attacks, much as how biological vaccines reduce the length and severity of biological infections. In

Based on "Fake Honeypots: A Defensive Tactic for Cyberspace", by Neil C. Rowe, E. John, Custy, and Binh T. Duong, which appeared in the Proceedings of the 7th IEEE Workshop on Information Assurance, West Point, New York, June 2006. © 2006 IEEE.

addition with a fake honeypot, an attacker cannot find any monitoring software to disable, and will be more likely to ignore the honeypot clues on the next real honeypot they find. Fake honeypots could be effective defenses if used occasionally on critical systems. But even if fake honeypots were widespread and ineffective, the effectiveness of true honeypots would increase in compensation.

Honeypots and fake honeypots are part of a long tradition of deception methods in civilian and military settings. [7] distinguishes six kinds of deception (masking, repackaging, dazzling, mimicking, invention, and decoying), and fake honeypots exemplify invention of a new kind of object that is not what it seems. [8] distinguishes nine military-relevant kinds of deception (concealment, camouflage, disinformation, lies, displays, ruses, demonstrations, feints, and insight), of which camouflage and disinformation are good descriptors for fake honeypots. More specific are the 32 semantic-case deceptions of [9], of which the most relevant are deception in content and instrument (the clues left on the fake honeypot mislead the attacker as to its hidden operations), deception in supertype (the fake honeypot is not a honeypot), and deception in purpose (the fake honeypot is to scare away attackers, not record their actions). In addition, fake honeypots can exploit a "second-order" deception effect [9] in that once an attacker recognizes a honeypot they will tend to think they have outsmarted the defender and be less able to see subsequent more-subtle deceptions; here the second-order deception concerns the ineptness of the administrators of the system containing the fake honeypot.

This paper will first examine the possible clues to honeypots. It will then present a decision-theoretic analysis of the advantages to the attacker of terminating an attack when a machine is suspected of being a honeypot. Results of experiments with human subjects are then reported that indicate people can detect subtle deception in otherwise normal text, suggesting that it is not hard for attackers to recognize clues to honeypots. Finally, experiments are reported with a real honeypot indicating that attackers do respond to some simple deceptions designed to encourage them to go away. We conclude with some speculation about the future evolution of honeypots and fake honeypots in information warfare.

II. CLUES TO HONEYPOTS

To make a good fake honeypot, we must understand how attackers detect honeypots. Just as with intrusion-detection systems, honeypot detection can be either signature-based or anomaly-based. So we should exaggerate signatures and anomalies for a fake honeypot. A package of modifications implementing these for an operating system is analogous to a vaccine.

Signature-based detection looks for features of an installation of an operating system that would suggest a honeypot. Signatures have been used to detect honeypots for spam email (which invite email and collect statistics on it) in a commercial tool "Honeypot Hunter" designed to help spammers [10]. [11] suggests some specific clues for low-interaction honeypots like HoneyD. The Sebek keystroke-monitoring honeypot tool of the Honeynet Project modifies some kernel system activities and tries to conceal them. Hence [12] suggests a number of specific clues this kind of modification can create, although some have since been removed from Sebek. Although most of a honeynet's monitoring activities are concealed, there are unavoidable clues to them in the form of additional software code, additional data left in main memory, and extra time delays.

A fake honeypot can leave these clues too; those in [2] are a start. [13] mentions several default values that indicate honeypots since they are difficult to modify, such as parameter values for the hardware devices and various "magic numbers" like that for the VMWare backdoor. So a fake honeypot should deliberately set these to the values indicative of a honeypot. [13] also points out that traces of Sebek are easy to find in memory, and mentions a hacker tool "module_hunter.c" that looks for them, so the designer of a fake honeypot should create such traces. They also note that the system read and write calls in Sebek are not to adjacent addresses, so a fake honeypot should ensure this. Finally, fake honeypots could also usefully delay responses to a wide range of commands since these suggest the overhead of surreptitious monitoring, perhaps by modifying the system read calls much as Sebek does. These fake clues can be implemented by modifying operating systems in just a few places.

Anomaly-based detection of honeypots compares metrics on an operating system to those of typical computer systems [14]. This works because honeypots have atypical users and usage. This does require familiarity (perhaps intuitive) with typical systems, and can be implemented by inspecting randomly-chosen files or directories to see if they look "normal". [11] suggests timing tests to detect low-interaction honeypots, and [15] suggests looking for a revealing anomaly in the number of attacks that one can launch from a honeypot, since low numbers definitely suggest a honeypot rather than a typical system.

Fake honeypots should thus try to make their obvious metrics different from those of normal computer systems. [14] developed 72 useful metrics and an implementation for comparing file systems. Of these, the most useful in fake honeypots are the number of files and subdirectories per directory since even an inattentive attacker can see their abnormalities. It is easier to make the anomaly for a fake honeypot in the direction of too-few files and subdirectories. Also useful and easy to control are the fraction of files that are system files (make this too high), the number of filenames that are English words (make this too few), the standard deviation of the length of a filename (make this close to zero), and the range of dates of last modification of files (make these nearly the same and well in the past). With enough modifications like these, it should be apparent that something is wrong to even an attacker who is not focused on detecting a honeypot.

III. MATHEMATICAL MODELING OF ATTACKER REACTIONS TO HONEYPOTS

A. Analyzing manual attacks

We can analyze mathematically what could make an attacker go away because they think a system is a honeypot. Our approach is similar in philosophy to [16] for analysis of deception in coalitions of agents, but our assumptions are specific to information security. First let us consider a manual attack where the attacker watches the progress of their attack, though perhaps only checking periodically. Such an attacker will generally be intelligent and it requires some planning to deceive them, as observed in recent research on deception in software agents [17]. Information warfare specialists [18] exemplify this kind of attacker.

We assume an attacker will give up when it becomes clear that it will require significantly more effort to compromise a computer, since the attacker thinks it is a honeypot, than to start over with another random computer on the Internet and compromise it. Most amateur attacks ("hackers") and oftentimes professional attackers treat all computers as equally valuable to compromise. That is because their most common goals are impressing other hackers, building "botnets" or arrays of controllable computers, or collecting salable intelligence about networks and their configurations, so there is no disadvantage besides in wasted time to trying another computer if one is too hard to attack. Assume:

c_r is the average cost perceived by the attacker (perhaps mostly in time) to reconnoiter or "footprint" a randomly chosen system;

c_c is the total average cost perceived by the attacker to subsequently compromise a randomly chosen system;

c is the remaining cost estimated by the attacker to compromise a randomly chosen system at some time;

b_c is the benefit to the attacker of compromising a machine, expressed in the same units as the c values;

p_h is the probability the system is a honeypot as perceived by the attacker at some time, based what they have seen the system do and their a priori probability of a honeypot;

r_h is the rate of damage (additional time incurred in the future) to the attacker, as believed by the attacker, by an average honeypot due to its learning of the attacker's attack methods; and

v_h is the ratio of the attacker's perceived value of compromise of a honeypot to their perceived value of compromise of a non-honeypot. This should be less than 1 for most attackers, since honeypots have ways to prevent attacks from spreading and are thus not as useful to an attacker, but this could be greater than 1 for "honeypot-hating" attackers. When the attacker is not sure a computer is a honeypot, the simplest model is to assume a linear weighting of $v = 1 - p_h(1 - v_h)$ where v is the relative value to the attacker of a machine that is uncertain to be a honeypot.

When an attacker discovers clues that indicate they are attacking a honeypot, they must choose whether to continue trying to compromise it or try to attack another computer. They will then need to do reconnaissance and incur the full cost of compromise on the other computer, but that could be a good tradeoff if they can avoid an obvious honeypot. Honeypots are currently rare, so we will assume that the second computer is not a honeypot. Changing the attack target can be evaluated in terms of the rate of compromises per unit time; if the attacker compromises a machine of relative value v to a normal machine, they need to do $1/v$ of these to get the benefit of compromising one normal machine. Hence the attacker should give up on a machine and start over on another if $(c + r_h p_h c)/v > c_r + c_c$ or if $c > v(c_r + c_c)/(1 + r_h p_h)$ or after substitution if $c > (1 - p_h(1 - v_h))(c_r + c_c)/(1 + r_h p_h)$.

To illustrate the inequality, consider some typical values where costs are measured in minutes. Suppose a typical reconnaissance is 1 minute, the cost to compromise a typical computer system is 30 minutes because of all the downloads, the attacker's perceived relative value of compromise of a honeypot is 0.2, and the rate of damage to an attacker by being on a honeypot is 0.03 (this factor does not affect the result much so it need not be precise). Then if an attacker thinks a machine is a honeypot with probability 0.5, the right side of the inequality becomes (1+30) * (1-0.5*0.8) / (1+0.03*0.5) = 18.6, so an attacker should give up with this machine if they have more than 18.6 minutes to go in the expected time of 31 minutes for the attack, else keep at it. This is encouraging for design of fake honeypots because it is not hard to design them so attackers will be more than half certain a machine is a honeypot after 60% of their attack.

B. Applications and extensions of the model

We can encourage an attacker to believe that the inequality holds if the perceived value of a honeypot is small, the cost of reconnaissance is small, the rate of damage to the attacker is large, or the probability of a honeypot is large. A fake honeypot should try to encourage belief in these conditions. A high probability of a honeypot can be suggested by clues in the operating system, and belief in the others by obvious clues that the attack is being monitored such as delayed responses to suspicious actions.

Some of the benefit of implementing fake honeypots can be obtained merely by disseminating stories that they are being used even if they are not, since increasing values of p_h make it more desirable for an attacker to go away sooner. So disinformation campaigns can be designed to encourage belief that we use fake honeypots, much as how disinformation about capabilities like troop strengths can be useful in conventional warfare.

Attackers will not necessarily do a careful decision-theoretic analysis of their costs and benefits. Some logs from honeypots [1] show attackers repeatedly failing because they are committed to a single approach. Nonetheless, we can still model this mathematically as a probability that the attacker will give up rather than a threshold. We hypothesize that this will occur with a probability that is a monotonic function of the strength of the inequality, i.e.:

$$p_{giveup} = f[c(1+r_h p_h)/(1-p_h(1-v_h))(c_r + c_c)]$$

Good candidates for f are sigmoidal functions such as the logistic function, the hyperbolic tangent, and $f(x) = x/(K+x)$.

We can fit these mathematical models to observed data for an attacker or class of attackers. They allow us to predict how much a system needs to do to scare off an intelligent attacker. Of course, an attacker may behave irrationally, but if they do they are mostly hurting themselves because they will be wasting time on unsuitable attack targets; feedback from the outcomes of attacks will indirectly train them to estimate a cost to attacking a honeypot. Note that it costs defenders themselves little to try to scare an attacker, since the methods we will propose here are mostly simple additions to an operating system.

This analysis also applies to automated attacks constructed by manual testing, so if we can fool an attacker during testing, or even get them to anticipate being fooled, we may get the deceptive effect many times over. There is anecdotal evidence [13] that some automated attacks do check for honeypots, which means we could make the attack repeatedly waste effort every time it is used. But we need not be too concerned with automated attacks when we are being deceptive because they can usually be stopped by any unexpected response, such as asking for additional confirmation before executing a command. Most automated attacks are so "brittle" (lacking conditionals in their execution scripts) that mild unexpected deceptions can stop them.

C. The cost of testing for a honeypot

So far we have assumed that the attacker goes about their business of attacking a machine without specifically trying to determine whether it is a honeypot, just noticing any clues they encounter. But a more systematic attacker can test for a honeypot before attacking a machine, using the clues mentioned in [13]. Then their value of p_h (the degree to which they think their victim machine is honeypot) would be either 0 or 1. However, this testing entails additional cost to the attacker and may not be cost-effective for them.

To analyze this, assume c_t is the cost of testing for a honeypot, p_{h0} is the a priori probability that a random machine is a honeypot, the test for a honeypot is perfectly accurate, the attacker leaves if the test indicates a honeypot, and the attacker does the test at the first opportunity after reconnaissance since there is no advantage to delay. Then we assume if the attacker detects a honeypot, they will find another machine at random and test it, continuing until they find a machine that the test confirms is not a honeypot. Then the average cost per full compromise when not testing will be greater than the average cost per full compromise with testing if:

$$(1-p_{h0})c_c + p_{h0}(c_c(1+r_h p_{h0})/v)$$
$$> c_t + p_{h0}(c_r + c_t) + p_{h0}^2(c_r + c_t) + ... + c_c.$$

The right side contains an infinite geometric series that can be simplified to $c_c - c_r + (c_r + c_t)/(1-p_{h0})$. So the inequality simplifies to:

$$c_t < p_{h0}[-c_r + c_c(1-p_{h0})(((1+r_h p_{h0})/v) - 1)]$$

Or if we substitute for v:

$$c_t < p_{h0}[-c_r + c_c(1-p_{h0})$$
$$(((1+r_h p_{h0})/(1-p_{h0}(1-v_h))) - 1)]$$

For the example data values given above, the right side of the inequality is 0.5*(30(1-0.5)((1.015/(1-0.5*(1-0.2)))-1)-1) =4.5, so any completely accurate test less than 4.5 minutes long is desirable. The tests described in [13] should fulfill this.

This formula can be used as a design criterion for a fake honeypot. We want the tests that confirm it is a fake honeypot (like inspection of main-memory traces) to be significantly more expensive than the standard tests (like counting the number of files on the machine or observing large delays on system responses) that suggest it is a honeypot. We should design the fake honeypot so that its testing cost exceeds the right side of the inequality, while testing cost for standard honeypot is less than the right side.

We can summarize the game-theoretic aspects of introducing fake honeypots by Table I below. This gives the expected cost to the attacker after an initial reconnaissance and selection of a victim machine. Here c_{tf} is the cost of testing whether an apparent honeypot is in fact a fake honeypot, p_{h0} is the probability of encountering a true honeypot on a randomly chosen machine, and p_{f0} is the probability of encountering a fake honeypot on a randomly chosen machine. We assume now that a test for a fake honeypot is an expensive test that is always accurate, and that the test for a honeypot will err when given a fake honeypot but not a true honeypot.

Note that, from the point of view of the defender, the fake-honeypot strategy is better than the normal-machine strategy since for two of the attacker's strategies it is equal and for one it is more costly. Similar, the true-honeypot strategy is better than the fake-honeypot strategy. However, the true-honeypot strategy requires considerably more setup cost to the defender which is not included in this comparison.

From the point of view of the attacker, comparing the expected cost of the second and third attacker strategies gives us a criterion for when an attacker should find it advantageous to do additional tests to determine if an

apparent honeypot is a fake honeypot. Then the attacker should test for a fake honeypot if:

$$c_{tf} < (c_t + c_r)[-p_{h0} + (p_{h0} + p_{f0})(1 - p_{h0}) / (1 - p_{h0} - p_{f0})]$$

or $c_{tf} < (c_t + c_r) p_{f0} / (1 - p_{h0} - p_{f0})$. Note if the probability of a fake honeypot is less than 10% this will be virtually impossible to satisfy. For the example values above when the probability of a fake honeypot is 0.1, the right side becomes 0.43, so we should test for a fake honeypot then only if the test takes less than 0.43 minutes. It would be unlikely that so fast a test could be devised if the fake honeypot is designed well; good tests like counting the number of attacks that can be launched from the machine [15] will likely take much more time than this.

TABLE I. EXPECTED COST IN THE GAME OF ATTACKER VERSUS DEFENDER WHEN BOTH HONEYPOT AND FAKE HONEYPOTS ARE POSSIBLE.

Attacker strategy (rows) / defender strategy (columns)	**Victim machine is a normal machine (probability $1 - p_{h0} - p_{f0}$).**	**Victim machine is a honeypot (probability p_{h0}).**	**Victim machine is a fake honeypot (probability p_{f0}).**
Attacker does not test the machine.	c_c	$c_c(1 + r_h p_h) / v_h$	c_c
Attacker tests whether the machine is a honeypot, and tries another random machine if it is.	$c_t + c_c$	$c_t + c_c + (c_r + c_t) / (1 - p_{h0} - p_{f0})$	$c_t + c_c + (c_r + c_t) / (1 - p_{h0} - p_{f0})$
Attacker tests both whether the machine is a honeypot and a fake honeypot, and leaves and tries another random machine only if it is a true honeypot.	$c_t + c_{tf} + c_c$	$c_t + c_{tf} + c_c + (c_r + c_t + c_{tf}) / (1 - p_{h0})$	$c_t + c_{tf} + c_c$

Thus fake honeypots appear to be very useful tactically if they represent 10% or less of all machines. But if the proportion of fake honeypots rises to 50% or more, the inequality will be difficult to satisfy and attackers will find it advantageous to test for them.

IV. EXPERIMENTS WITH TEXT DECEPTION DETECTION

Another key issue in designing either a honeypot or a fake honeypot is whether an attacker will notice it. People in general are not especially good at detecting deception [19]. There is considerable research on detecting deception because of its relevance to criminology, law, business, psychology, and the performing arts. [20] provides a good introduction to the psychological aspects, and [21] provides useful ideas about cooperative autonomous methods. People are especially poor at detecting online deception as witnessed by the greater effectiveness of many traditional scams on the Internet than in the real world [22].

A. Deception channels

Some researchers have suggested the difficulty of online detection is due to the reduced capacity of the communications channel in interacting with computers, since on a narrow channel it is hard to detect subtle clues [23]. So we have explored the theory that deception is a surreptitious information-narrowing in a communications channel. A spy or an attacker needs to obtain information about your computer systems; a defense could be to reduce the information flow to them by inserting "noise" in the form of errors of either commission or omission. Very little noise is necessary to invalidate documents like computer programs; more noise is necessary for text messages.

However, effective deception requires the deceivee to believe they are receiving the same amount of information that they expect. People are good at detecting significantly different levels of information flow in a channel, as when the noise on a telephone line increases or a person deliberately makes nonsensical statements to see if anyone is paying attention. Thus a good strategy for honeypots is to provide the same rate of information as real systems, but information that is significantly qualitatively different. Information can be defined [24] as the summation over all possibilities i of $I = -\sum_i p_i \log(p_i)$.

Maintaining similar information flow rate during deception has several implications for communications with honeypots, by what we can call "honeychannels". It is desirable that the information content of the honeychannel be similar to that of typical victim machines in the words and characters it transmits and their rate of transmission. This means it will be important to simulate the frequencies of words and data in the normal messages of a real computer system as

much as we can during deception. But it also means that the semantics of information transmitted must preserve a similar information content. This means that fake information we present to a user must have roughly the same number and kinds of internal logical connections as real information.

But honeychannels are additionally challenging to design because maintaining information flow rates is only one design requirement. We must also make sure that the honeychannel transmits enough information to deceive the attacker. This requirement conflicts to some extent with the information-constancy requirement because many useful deceptions, like giving an excuse that the network is down, lower the information flow in the honeychannel. One tactic is to ensure that such deceptions do not last long, as when they encourage the attacker to go away so they do not see a consistently lower information flow rate. Another tactic is to conceal the information content in something unreadable, as when transmitting compiled code, but this is not always possible.

With fake honeypots as opposed to true honeypots, we should try to be noticeably deceptive, but not so deceptive that it would be obvious that we are being deceptive. This is analogous to the formula for testing for honeypots in the last section. We need to transmit information a rate between two levels: one at which the attacker notices they are being deceived, and one at which the attacker will know we know we are being deceptive.

B. Experiments with distorted text files

A simple experiment we can do that bears on these issues is to study to what extent users notice distortions in text and file listings. [25] pioneered in automated construction of fake online documents, but did not study their effectiveness. [26] developed sophisticated methods for detecting fake documents, but these are beyond the abilities of humans to duplicate. So we set up a test with computer-science students and staff at our school.

We gave subjects pairs of text where one item of text was deliberately distorted and one item was not. Starting text came from two sources: A requirements document for the JMPS (Joint Mission Planning System) for aircraft mission planning, and file directories from our school's Web site as used for the honeypot file directories in [14]. The source requirements we used were chosen to exemplify technical text, since attackers will be examining technical documents in their inspection of honeypots; they referred to terms unfamiliar to the subjects, but which could be categorized from context. The fake file directories were created from names of real Web-page directories at our school by choosing randomly selected filenames and appending them to a single directory name.

The distorted requirements were two-sentence paragraphs created in two ways. Some were created by substitution of sequences of words in the document with other sequences of words, under the constraint that the first and last words of both sequences matched as well as one word in the interior of each sequence. The substituted sequences were 3 to 13 words that appeared elsewhere in the document, which helped preserve coherence. An example result of this method is:

"In the event of a network failure of electronic transmission failure, JMPS shall provide the capability to output graphics for printing in the error log."

The other fake requirements were created by chaining together overlapping strings where one to three words on the end of each segment matched the words at the front on the next segment. The same source strings were used as in the other method. An example is:

"The generic UPC shall provide implementations of abstractions which require UPC extension to be useful to assist other developers and so that the JMPS system can be exercised in a realistic fashion."

With both methods, we manually discarded results that were ungrammatical or accidentally correct specifications. An example question about requirements was:

Circle the statement that appears most likely to have come directly from a real requirements document.

a) "JMPS shall overlay a geophysical data over any GI&S background or layers. JMPS shall provide the capability for the user to adjust the intensity of the GI&S background in order to fade-out or fade-in the background so that geophysical data is easier to see."

b) "JMPS shall display facilities query results overlaid on any GI&S background or layers using user-defined symbology. JMPS shall display the mean number of charts per page, and the page borders of strip charts relative to sea level."

An example of a question given our subjects about directories was:

Circle the listing that appears most likely to have come from a real computer system.

(a)
physics/gaitan/bubble/specv
physics/gaitan/bubble/other
physics/gaitan/bubble/rp
physics/gaitan/bubble/gilmore
physics/gaitan/bubble/int
physics/gaitan/bubble/trash
physics/gaitan/bubble/old.imsl
physics/gaitan/bubble/flynn

(b)
physics/cooper/irtool/examples
physics/cooper/irtool/gif_files
physics/cooper/irtool/idlold
physics/cooper/irtool/wizard
physics/cooper/irtool/114564-01
physics/cooper/irtool/template
physics/cooper/irtool/target_n_horiz
physics/cooper/irtool/ass2

In both examples the first choice is real and the second is fake. Clearly, these were difficult questions that

required good intuition that something was "not right" to solve.

C. Results

We conducted experiments with 14 subjects, 12 pairs of text (4 with the first method of modification, and 8 with the second), and 4 file-directory listings. Text examples had an average of three modifications per example. Subjects scored 64.3% correct on the text and 53.6% on the listings. These scores were significant at levels of 3.87 standard errors and 0.54 standard errors respectively, so the text was significantly detected while directories were not (though this may reflect too few of the latter questions). Scores on individual questions ranged from 0.857 to 0.286. Results for the substitution obfuscation method for text were 0.636, which was not a statistically significant difference from that for the chaining method of 0.670. There was a mild correlation of performance with the number of characters in the problem; subjects scored 54.8% for the 8 shorter questions versus 62.5% for 8 longer questions (text varied from 108 to 411 characters in length, and directories from 8 to 18 lines). Tables II and III show the complete set of experimental data, with the final column and final rows representing averages. Correct answers are indicated by "1"s.

From these experiments we conclude that randomly modified text files can be detected even if the clues are subtle, despite the difficulty in general that people have in detecting deception, and randomly constructed file directories may be detected. This is because attackers by their very nature should be more suspicious than our student subjects and better at detecting clues. We can also see that some subjects are significantly better than others at detecting manipulation. This suggests that even subtle clues in fake honeypots could be noticed by a perceptive attacker. It is interesting that we are measuring deception "intuition" because intuition has seen a recent revival of interest in psychology [27].

TABLE II. EXPERIMENTAL DATA SHOWING SUBJECTS (ROWS) VERSUS TEST-MODIFICATION QUESTIONS (COLUMNS).

1	0	0	1	1	0	1	1	0	1	1	1	**0.667**
1	1	1	0	1	1	1	1	1	1	1	1	**0.917**
1	1	1	1	1	0	1	1	1	1	1	0	**0.833**
1	1	0	0	0	0	1	0	1	1	1	1	**0.583**
1	1	1	1	1	1	1	1	0	1	0	1	**0.833**
1	0	1	0	1	1	0	1	1	0	1	0	**0.583**
1	1	1	1	1	1	1	1	1	1	0	0	**0.833**
0	1	1	0	0	1	0	0	0	0	1	0	**0.333**
1	0	0	0	1	0	0	0	1	0	1	1	**0.417**
1	0	1	1	0	0	0	1	0	0	0	1	**0.417**
1	1	1	0	1	1	1	1	1	1	0	1	**0.833**
1	1	1	1	0	1	1	0	1	0	1	0	**0.677**
0	1	0	1	0	1	1	1	1	0	0	0	**0.500**
1	0	1	0	1	1	1	0	0	1	1	0	**0.583**
.857	**.643**	**.714**	**.500**	**.643**	**.643**	**.714**	**.643**	**.643**	**.571**	**.643**	**.500**	**0.643**

TABLE III. EXPERIMENTAL DATA SHOWING SUBJECTS (ROWS) VERSUS DIRECTORY-MODIFICATION QUESTIONS (COLUMNS).

0	0	0	1	**0.25**
0	0	0	0	**0.00**
0	1	1	1	**0.75**
0	0	1	0	**0.25**
0	0	0	1	**0.25**
1	0	0	1	**0.50**
0	1	0	0	**0.25**
0	1	1	1	**0.75**
0	0	1	1	**0.50**
0	1	1	1	**0.75**
1	1	1	1	**1.00**
1	0	1	1	**0.75**
1	1	0	1	**0.75**
0	1	1	1	**0.75**
.286	**.500**	**.571**	**.786**	**.538**

V. EXPERIMENTS WITH REACTIONS TO HONEYPOTS

A. Setup

To put our ideas together, we also conducted experiments with real honeypots and real attackers to assess whether malicious traffic could be influenced by clues suggesting a honeypot [28]. We used a three-computer configuration: a router, a data-capture computer, and a machine implementing two virtual honeypots in a virtual honeynet. All machines were Intel Pentiums with SuSE Linux version 10 as the operating system (the latest version). The router had the intrusion-detection system of Snort (version 2.4.3) with the public rules available. Snort alerts were sent through BASE (the Basic Analysis and Security Engine) to a Web front end running on an Apache Web server (which could be attacked too but rarely was). The honeynet machine used VMware 5.5 to simplify restart after compromise. One virtual machine on the honeypot was a Windows workstation and one was a Windows 2000 Advanced

Server for the World Wide Web. The data-capture machine stored Snort log data in a PostgresSQL database.

The machines were attached to an Internet connection that did not go through any firewall, unlike nearly all of our school's connections. Thus our machines were exposed to the full range of Internet attacks. We progressively left a number of anomalies and clues that indicated this was a honeynet. We made no attempt to conceal VMWare, Snort, and TCPDump utilities, known to be associated with honeypots. We also copied some innocent files to the honeypots, like course notes, to help populate their file systems. To encourage unauthorized visitors, at the start of experimentation we posted the IP addresses of the machines in some blogs with a notice to "check this out", to encourage both manual and automated mining of the address for attacks. We restarted the machines three times when they crashed or when the virus-checker found viruses, using VMWare to restore the initial system state.

B. Results

We tabulated Snort alert information over a period of 46 weeks. Figures 1-3 plot the number of alert clusters of various types per week. We used totals per week because we saw significant variation in alerts with day of the week: The rate of alerts on Saturday and Sunday was 15% less than the rate on the other days of the week [29]. Also, since there is much random variation in attack times because these attacks are analogous to “background radiation” of the Internet [30], an average over a week gives clearer trends.

Figure 1 shows the total number of alerts clustered in two ways, time clustering to show the variation in volume of the attacks, and packet-property clustering to show the variation in the variety of the attacks. Time clustering was done by identifying all identical alerts within 10 minutes as a single event. Results with windows of 3 minutes and 30 minutes were not significantly different; clustering by remote IP address was not as helpful since we had a significant number of similar attacks from multiple machines (probably due to spoofing). Packet-property clustering was done by taking 12 properties of the alerts and clustering them each day using a K-Means algorithm with a boolean (“city-block”) metric, adjusting the threshold to give a mean of 306.4 and a geometric mean of 71.3 clusters per day. The K-Means algorithm used 50 initial clusters each day, but changing this number to 5 or 500 made a difference of less than 10% in the number of clusters per day.

Attacks generally decreased in both number and variety over the 46 weeks. There was a sharp decrease in the first six weeks during which we gradually increased the obviousness of the honeypots and built the honeynet, but the decrease continued after a six-week gap in which the honeynet was disconnected from the Internet. To distinguish whether this was due to our advertising of the site, we also did similar advertising in week 41 resulting in less impressive effects (though a definite increase in attacks). This supports the hypothesis that new machines get attacked more frequently simply because they are new, and that most (but not all) attackers finish their reconnaissance of a new machine within a few weeks after it is connected. That suggests that a good deception for a normal (non-honeypot) machine is to reuse an existing IP address when attaching a critical machine to the Internet so that attacks will be lower at first. Also, the fact that attackers did not follow up on their initial attacks suggests either that they realized quickly that our machine will not succumb easily or that they are being scared away by the variety of clues that it is a honeypot; both explanations help us as defenders.

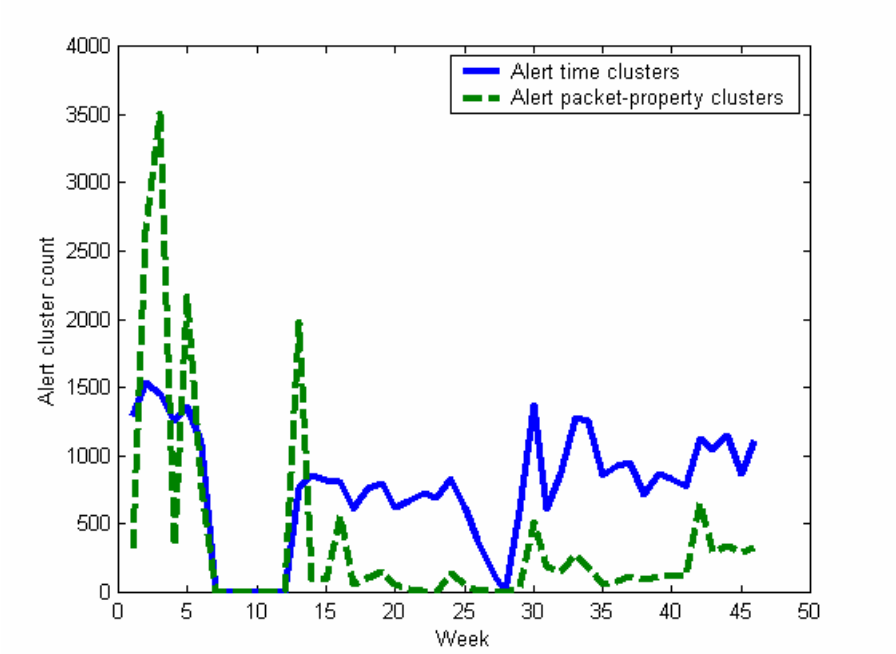

Figure 1. Total alert clusters per week for our honeynet.

Another important trend in Figure 1 is effect of the gaps in Internet availability of the honeypot for 42 days in weeks 7-12, 11 days in weeks 26 and 27, and 4 days in week 31; weeks 7-12 involved a deliberate turning off of the system, and the other weeks represented crashes due to our own setup problems. (In week 41 we started making configuration changes.) These gaps seemed to excite attackers even more, especially in the later periods. This suggests it is actually a good strategy for a honeypot to have an unreliable Internet connection since this will incite more attacks, and honeypots want to encourage attacks. Figures 2 and 3 show the rates for the 11 most common categories of attacks, and most of the service-interruption effect is due to ICMP attacks, with effects also on SCAN and INFO attacks. Most of these are easy to defend against, so the increase in the attack rate on the honeypot is keeping attackers tied up attacking useless targets (and providing more data to us).

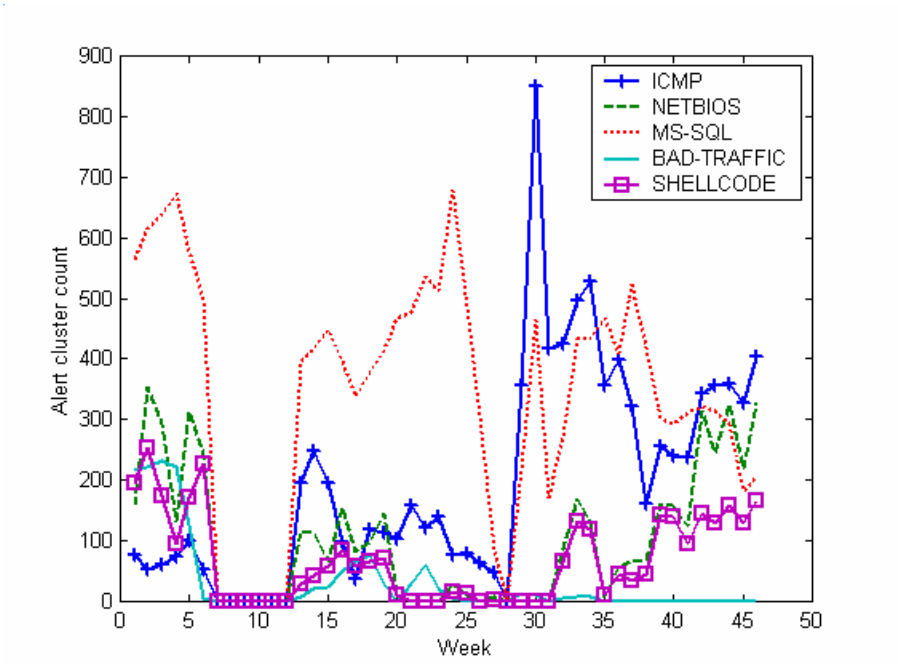

Figure 2. Alert clusters per week for our honeynet of the five most common types.

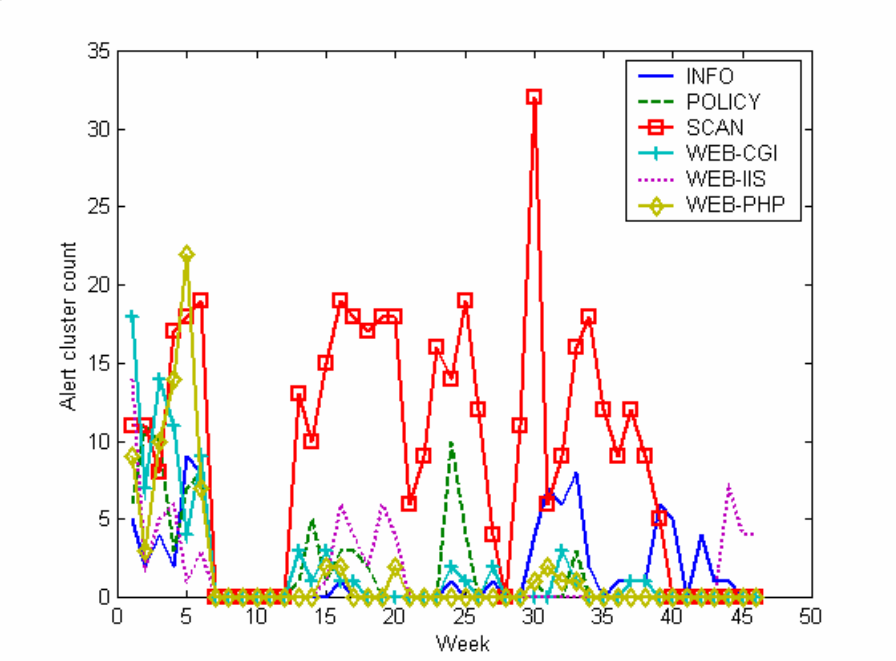

Figure 3. Alert clusters per week for our honeynet of six lesser categories.

Figure 4 plots the time-clustered data from Figure 1 in the last 17 weeks against alert counts from our school's firewall in the same time period (with zeros in weeks 5-8 when the firewall data collection was not working). Only three categories were used since the school does not collect data in several major categories, and the clustering method used by the school was different: It was the count of successive identical alerts from the same IP address. The correlation between the two sets of alerts is only mild. Thus most attacks we saw were targeted rather specifically at operating system and software versions.

Effects of deceptions are often best seen in subtler measures on a network. Figures 5 and 6 show histograms of the time gaps between successive Snort alerts on the honeypot involving the same external IP address. Figure 5 shows gaps from July 27 through October 19, and Figure 6 shows gaps October 20 to November 21 after we started advertising our site again. Given the number of clusters included, the sharp peaks likely represent statistically significant distinctive timings of automated attacks. Since Figure 6 covers a time when we made a number of configuration changes to the honeypot, the new peaks and fewer alerts within the same millisecond likely represent reactions of attackers and new attack methods. This gives us good clues to study further to understand attacker reactions to features of a system.

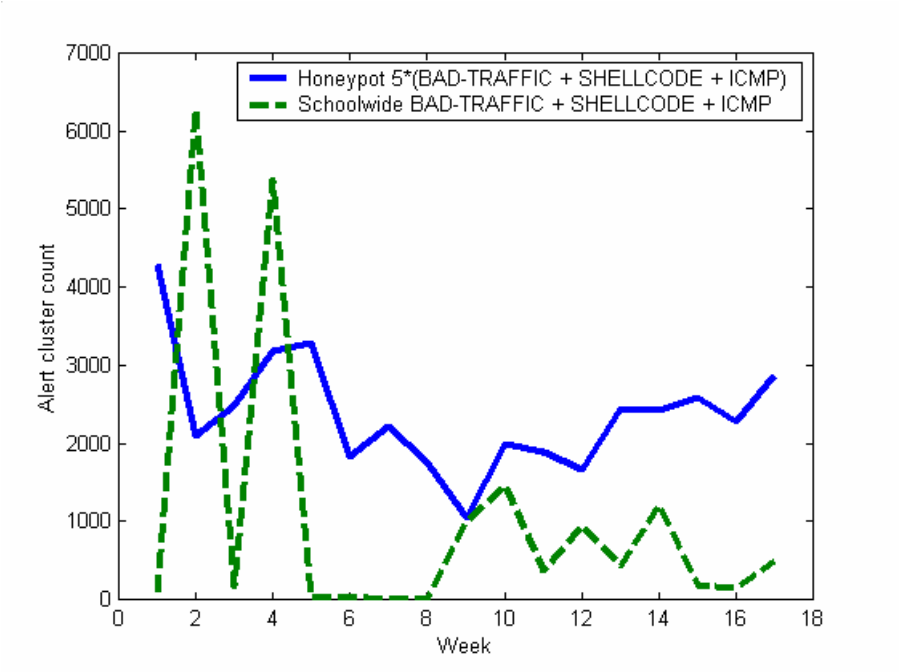

Figure 4. Alert clusters per week on our honeynet versus alerts on our school's firewall.

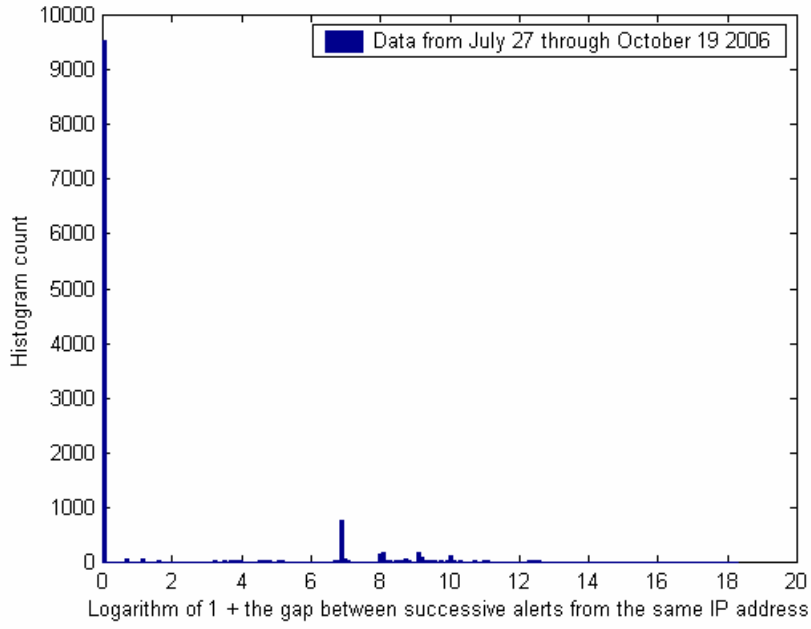

Figure 5: Histogram of interarrival times in milliseconds between alerts over a period of no changes to the honeynet.

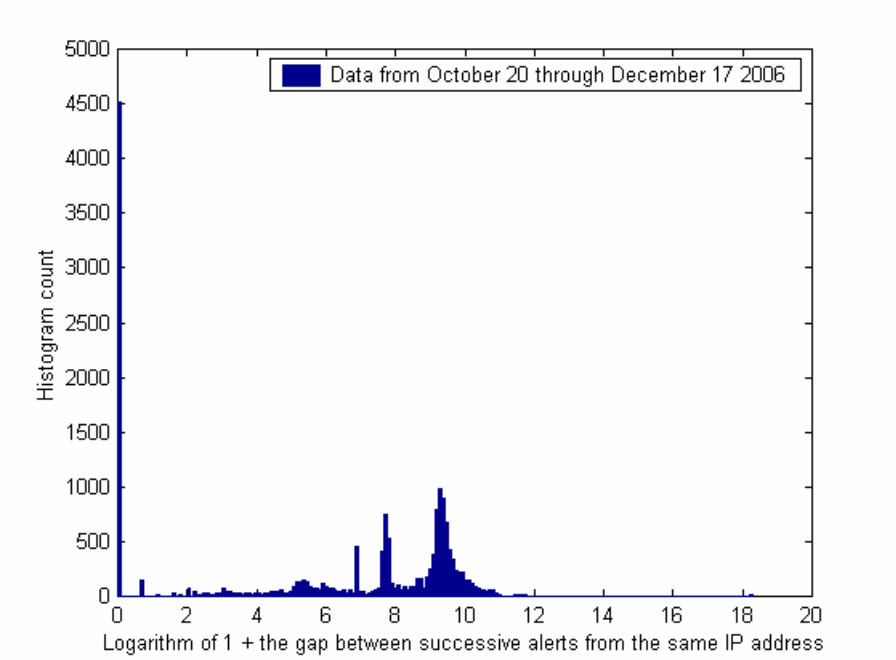

Figure 6: Histogram of interarrival times in milliseconds for alerts during a period of system changes.

Results on our honeypots are not conclusive, but there seem to be effects of our deceptions on attackers. We will use this testbed to continue experimenting with more sophisticated manipulations of attackers.

VI. The Future of Fake Honeypots

Fake honeypots are a third step in information warfare after first honeypots and then methods for detecting honeypots [6]. Our experiments of the last section used a fourth step of "fake fake honeypots" that are actually real honeypots but pretend to be noticeable fake honeypots. But we believe this escalation of tactics will not continue indefinitely, much in the way that antivirus software and subsequent countermeasures by virus writers have not led to infinite layers of measures and countermeasures. Instead, deception and counterdeception are more like a game where one player makes a move and the other counters by a move of their own [7], exemplified by our three-by-three game in Table I. Counter-counterdeception is often difficult to distinguish from just plain deception, and can usually be interpreted as just an independent strategy for deception, since diversity in deception methods is essential to their effectiveness.

However, we are likely to see an increased rate of honeypot measure-countermeasure evolution and escalation in the near future [3]. Virus-antivirus measures went through such a period during the 1990's and evolution has now slowed; spam-antispam methods developed rapidly in the last few years and innovation is only now slowing as blacklisting of spamming sites and content analysis of spam has matured [31]. The fundamental reason for the slowdown in both cases was the increasing difficulty to malicious users of keeping pace with evolution of defenses. An attacker need only make one bad mistake to get detected, and as attacking becomes more complicated, the chance of a mistake by the attacker increases in the same way that the number of bugs in software increases with its size. Growth in malicious attacks on computer systems has been slower than the growth of viruses and spam due to the greater difficulty of innovation and smaller number of experts, but specific categories like honeypot-antihoneypot methods can show evolutionary spurts. We suspect that is happening now. However, in the long run the benefit will fall to the defender because now attackers have one more thing to worry about, not just whether a site is a honeypot but whether it is a false honeypot.

Acknowledgment

This work was supported by NSF under the Cyber Trust Program. Opinions expressed are those of the authors and not necessarily those of the U.S. Government.

References

[1] The Honeynet Project, *Know Your Enemy: Learning about Security Threats, Second Edition.* Boston, MA: Addison-Wesley, 2004.

[2] M. Vrable, J. Ma, J. Chen, D. Moore, E Vadekieft, A. Snoeren, G. Voelker, and S. Savage, "Scalability, fidelity, and containment in the Potemkin Virtual Honeyfarm", *Proc. ACM Symposium on Operating Systems Principles*, Brighton UK, 148-162, 2005.

[3] B. McCarty, "The honeynet arms race," *IEEE Security and Privacy, 1*(6), pp. 79-82, November/December 2003.

[4] C. V. Ford, *Lies! Lies!! Lies!!! The Psychology of Deceit.* Washington, DC: American Psychiatric Press, 1996.

[5] P. Sztompka, *Trust.* London, UK: Cambridge University Press, 1999.

[6] W. Tirenin and D. Faatz, "A concept for strategic cyber defense," *Proc. Conf. on Military Communications*, Atlantic City, NJ, Vol. 1, pp. 458-463, October 1999.

[7] J. Bell, and B. Whaley, *Cheating and Deception.* New Brunswick, NJ: Transaction Publishers, 1991.
[8] J. Dunnigan and A. Nofi, *Victory and Deceit, second edition: Deception and Trickery in War*. San Jose, CA: Writers Club Press, 2001.
[9] N. Rowe, "A taxonomy of deception in cyberspace," *International Conference on Information Warfare and Security*, Princess Anne, MD, pp. 173-181, March 2006.
[10] N. Krawetz, "Anti-honeypot technology," *IEEE Security and Privacy*, *2*(1), pp. 76-79, Jan.-Feb. 2004.
[11] X. Fu, W. Yu, D. Cheng, X. Tan, K. Streff, & S. Graham, "On recognizing virtual honeypots and countermeasures," *Proc. 2nd IEEE International Symposium on Dependable, Autonomic, and Secure Computing*, Indianapolis, Indiana, pp. 211-218, September 2006.
[12] M. Dornseif, T. Holz, and C. Klein, "NoSEBrEaK – attacking honeynets," *Proc. IEEE Workshop on Information Assurance and Security*, West Point, NY, 1555, June 2004.
[13] 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.
[14] N. Rowe, "Measuring the effectiveness of honeypot counter-counterdeception," *Proc. 39th Hawaii International Conference on Systems Sciences*, Poipu, HI, January 2006.
[15] C. Zou, and R. Cunningham, "Honeypot-aware advanced botnet construction and maintenance," *Proc. International Conf. on Dependable Systems and Networks*, pp. 199-208, June 2006.
[16] M. Belmonte, R. Conejo, J. Perez-De-La-Cruz, and F. Triguero, "Experiments on robustness and deception in a coalition formation model," *Concurrency and Computation: Practice & Experience*, Vol. 18, no. 4, pp. 371-386. 2006.
[17] D. Christian and R. C. Young, "Strategic deception in agents," *Proc. 3rd Intl. Joint Conference on Autonomous Agents and Multiagent Systems*, New York, NY, pp. 218-226, 2004.
[18] W. Hutchinson and M. Warren, *Information Warfare: Corporate Attack and Defense in a Digital World*, London: Butterworth-Heinemann, 2001.
[19] A. Vrij, *Detecting Lies and Deceit: The Psychology of Lying and the Implications for Professional Practice*, Chichester, UK: Wiley, 2000.
[20] R. J. Heuer, "Cognitive factors in deception and counterdeception," in *Strategic Military Deception*, ed. Daniel, D., & Herbig, K., New York: Pergamon, 1982, pp. 31-69.
[21] E. Santos and G. Johnson, "Toward Detecting Deception in Intelligent Systems," *Proc. SPIE Defense & Security Symposium*, Vol. 5423, Orlando, Florida, 2004.
[22] S. Grazioli and A. Wang, "Looking without seeing: understanding unsophisticated consumers' success and failure to detect Internet deception," *Proc. 22nd Intl. Conf. on Information Systems*, pp. 193-204, 2001.
[23] J. Burgoon, G. Stoner, J. Bonito, and N. Dunbar, "Trust and deception in mediated communication," *Proc. 36th Hawaii Intl. Conf. on System Sciences*, Honolulu, HI, 44.1, 2003.
[24] T. Cover and J. Thomas, *Elements of Information Theory, 2nd Ed.* New York: Wiley, 2006.
[25] S. Gerwehr, J. Rothenberg, and R. H. Anderson, "An arsenal of deceptions for INFOSEC," Project Memorandum, National Defense Research Institute, Rand Corp., PM-1167-NSA, October 1999.
[26] S. Kaza, S. Murthy, and G. Hu, "Identification of deliberately doctored text documents using frequent keyword chain (FKC) model," *Proc. IEEE Intl. Conf. on Information Reuse and Integration*, October 2003, pp. 398-405.
[27] G. Klein, *Sources of Power: How People Make Decisions*. Cambridge, MA: MIT Press, 1999.
[28] B. T. Duong, "Comparisons of attacks on honeypots with those on real networks," M.S. thesis, Naval Postgraduate School, www.cs.nps.navy.mil/people/faculty/rowe/oldstudents/duong_thesis_final. htm, March 2006.
[29] N. C. Rowe, H.-C. Goh, S. L. Lim, and B. T. Duong, "Experiments with a testbed for automated defensive deception planning for cyber-attacks," *2nd International Conference in I-Warfare and Security*, Monterey CA, USA, March 2007.
[30] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson, "Characteristics of Internet background radiation," *Proc. 4th ACM SIGCOMM Conference on Internet Measurement*, Taormina, IT, pp. 27-40, 2004.
[31] J. Zdziarski, *Ending Spam: Bayesian Content Filtering and the Art of Statistical Language Classification*. San Francisco: No Starch Press, 2005.

Neil C. Rowe is Professor of Computer Science at the U.S. Naval Postgraduate School where he has been since 1983. He has a Ph.D. in Computer Science from Stanford University and three other degrees from the Massachusetts Institute of Technology. He is author of 130 technical papers and a book. His main interest is information security and he has also worked on data mining, robotics, and intelligent tutoring systems.

E. John Custy is a computer scientist at the SPAWAR Systems Center, San Diego, California. He has an M.S. in Software Engineering, Naval Postgraduate School (2003), an M.Eng. in Engineering Science, Pennsylvania State University (1994), an M.A. in Cognitive and Neural Systems, Boston University (1991), and a B.S. in Electrical Engineering, New Jersey Institute of Technology (1986).

Binh T. Duong is a computer scientist at the SPAWAR Systems Center, San Diego, California. She has a M.S. in Computer Science from the Naval Postgraduate School (2006). Her main interests are information security and computer forensics. She currently supports the Navy fleet working with the Computer Network Defense Afloat team.