2006-06

Fake Honeypots: A Defensive Tactic for Cyberspace

Rowe, Neil C.; Duong, Binh T.; Custy, E. John

Monterey, California. Naval Postgraduate School

Fake Honeypots: A Defensive Tactic for Cyberspace

Neil C. Rowe, Binh T. Duong, and E. John Custy

Abstract?Cyber-attackers are becoming more aware of honeypots. They generally want to avoid honeypots since it is hard to spread attacks from them, attacks are thoroughly monitored on them, and some honeypots contain planted false information. This suggests that it could be useful for a computer system to pretend it is a honeypot, to scare away smarter attackers. We examine here from a number of perspectives how this could be accomplished as a kind of ?vaccination? of systems to reduce numbers of attacks and their severity. 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 also report experiments with real attackers against a honeypot of increasing obviousness. Results show that attacks on it decreased over time which may indicate that attackers are being scared away. We conclude with some speculation about the escalation of honeypot-antihoneypot techniques.

This paper appeared in the Proceedings of the 7th IEEE Workshop on Information Assurance, West Point, NY, June 21-23 2006.

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

I. INTRODUCTION

Honeypots are computer systems designed for the one purpose of learning about cyber-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. This permits 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, attackers are now becoming aware of honeypots, and this became noticeable between the 2002 and 2004 editions of 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 they thrive on stealth and concealment for their effectiveness. The main purpose of honeypots is to learn about current threats so countermeasures can be developed; many attackers jealously guard their attack methods to gain status with other attackers and to preserve their effectiveness, so revelation of the methods is a disaster for them.

Attackers also want to avoid honeypots because an attack on one is unlikely to achieve much and could hurt the attacker in the long run. Honeypots are designed by people that are knowledgeable about security threats, and most could be disabled anytime by their owners if an attack were to get out of hand. While honeypots give the appearance of being easy to subvert, important features like logging facilities are difficult to subvert, and careful controls 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 a small minority of the machines in the world, attackers could more fruitfully spend their time attacking others.

Of course, a few attackers who discover a honeypot may want to stay and fight it. They may like a challenge; the may want revenge; or they may react emotionally in other ways, since deception often stimulates emotions [4, 5]. But then the honeypot is still useful because disabling the monitoring is hard, and in the meantime the attacker will be recorded. Furthermore, people unconsciously may reveal more themselves when emotionally motivated. So hackers that overreact to honeypots can also benefit the defender.

So a simple defensive ploy might be to make ordinary systems appear to be honeypots. Unlike most security measures, this would work best against the smarter attackers, and could be an effective defense in asymmetric warfare [6] against surprise attacks. Attackers are usually deceptive 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. These clues would provide a kind of "vaccination" for computer systems that would reduce the length and severity of the attacks on them although not blocking them completely, much as how biological vaccines reduce the length and severity of biological infections. While fake honeypots would be ineffective if widespread, they could be effective if used occasionally on critical or important systems.

II. CLASSIFYING THE DECEPTION IN FAKE HONEYPOTS

Honeypots are part of a long tradition of decoys for deception in both 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 probably the best 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.

III. CLUES TO HONEYPOTS

To make a good fake honeypot, we must understand the clues attackers are using to detect honeypots. Just as 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 signature-based approach can look for particular 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 [11]. The Sebek keystroke-monitoring honeypot tool of the Honeynet Project modifies some kernel system activities and tries to conceal them. [10] suggests a number of specific clues this kind of modification can create, although some have since been fixed in Sebek. Since a honeynet is doing secret activities, there will be additional code to do these activities, data left in main memory, and extra time delays associated with the monitoring.

Thus to create a fake honeypot, we should put these clues into a computer system. The clues in [2] are a start. [12] 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. [12] also points out that traces of Sebek are easy to find in memory, and mentions a hacker tool "module_hunter.c" looks for such signatures. So the designer of a fake honeypot should obtain this tool and create some background processes to put clues into memory that the tool will recognize. [12] also notes 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 include unnecessary timing delays in 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.

An anomaly-based approach to detecting honeypots takes metrics on the entire system and compares those to typical metrics of computer systems [13]. This works because honeypots do not have ?normal? users and do not generate normal data in their files and their characteristics. While this does require familiarity (perhaps intuitive) with typical systems, it easy to do by inspecting randomly-chosen files or directories to see if they look normal.

Fake honeypots should thus try to make their most-visible metrics different, but not too much, from those of normal computer systems. (We say ?not too much? because otherwise the attacker will know they are being manipulated and resist us.) [13] developed a set of 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 a cursory inspection of a file system by an attacker will reveal anomalous values. Of course, it 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.

IV. MATHEMATICAL MODELING OF ATTACKER REACTIONS TO HONEYPOTS

A. Analyzing manual attacks

We can analyze mathematically what it takes to make an attacker go away because they think a system is a honeypot. First let us consider a manual attack where the attacker is initiating each step of the attack separately. Such an attacker will generally be intelligent and it requires some planning to deceive them, as is observed in recent research on deception in software agents [14]. Information warfare specialists [15] exemplify this kind of attacker.

Our approach is to 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 computer from the Internet and compromise it. Most hackers and many non-hacker attackers treat all computers as equally valuable to compromise for their usual goals of impressing other hackers, building "botnets", or collecting intelligence, so there is no disadvantage besides time to trying another one. Assume:

- is the average cost perceived by the attacker (perhaps mostly in time) to do reconnaissance to "footprint" a given system;

- is the total cost perceived by the attacker to subsequently compromise a randomly chosen system;
- is the perceived remaining cost to the attacker to achieve their goals of compromise on a given system at some time;
- is the probability the system is a honeypot as perceived by the attacker;
- 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, where damage is assumed proportional at this rate to the time the attacker spends in the honeypot; and
- is the ratio of the attacker's perceived value of compromise of a honeypot to their perceived value of compromise of a non-honeypot (which should be less than 1 for most attackers, since honeypots have ways to prevent attacks from spreading and are thus less useful to an attacker, but this could be greater than 1 for "honeypot-hating" attackers). So when the attacker is not sure a computer is a honeypot, the effective value of the ratio can be calculated by a linear weighting as .

When an attacker discovers they are attacking a honeypot, they must choose whether to continue trying to compromise it or try to attack another computer chosen semi-randomly. 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. Then giving up incurs lower additional cost to the attacker if:

or if:

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 a honeypot is 0.5, 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 thanks to our deception, the right side of the inequality becomes 31*(1-0.5*0.97)(1+0.03*0.5)=15.72, so an attacker should give up with this machine if they have more than 15.72 minutes to go in the expected time (31 minutes) of the attack, else keep at it. This is encouraging for design of fake honeypots because they should be easy to design so attackers will be more than 50% certain a machine is a honeypot in the first 50% of the duration of their attack.

Note that the right side of the inequality can be greater than , which means that an attacker should give up immediately. This occurs if the cost of reconnaissance is negligible, the rate of damage to the attacker is sufficiently large, or both the probability of a honeypot is large and its value to the attacker is small. Then the goal of a fake honeypot should be to encourage belief in either of the last two conditions by clues. A high probability of a honeypot is easiest to induce by leaving clues in the operating system, but we can also encourage belief in a high rate of damage and a low value of compromise by making it appear very clear that monitoring of the attacker is ongoing.

Attackers will not 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 just one approach. Nonetheless, we can still analyze this mathematically by estimating 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.:

Good candidates for h are sigmoidal functions such as the logistic function, the hyperbolic tangent, and .

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 other unsuitable attack targets with such carelessness; feedback from the outcomes of attacks will indirectly train them to estimate a cost to attacking a honeypot. Note that it costs ourselves very little to try to scare an attacker, since the things we will propose here are mostly simple additions to an operating system.

B. Analyzing automated attacks

It might seem that this analysis would not apply to automated attacks. But automated attacks are constructed by manual testing, so if we can fool the attacker then, or even get them to anticipate being fooled, we may get the deceptive effect many times over. There is anecdotal evidence [12] that some automated attacks check for honeypots, which means if we can make the attack creator believe there are enough honeypots to be worth checking, we could repeatedly fool users of the resulting tool.

A more fundamental reason not to be concerned with automated attacks when we are being deceptive is that they can usually be stopped by any kind of unexpected response, such as asking for additional confirmation before executing a command. Automated attacks are generally so "brittle" (lacking conditionals in their execution scripts) that mild unexpected deceptions can stop them.

V. EXPERIMENTS WITH TEXT DECEPTION DETECTION

A key issue in designing a fake honeypot is whether an attacker will notice it. Deceptions that are too subtle to be noticed are useless, and people in general are not especially good at detecting deception [16].

A. Detecting deception

There is considerable research on detecting deception because of its relevance to police work, law, business, psychology, and the performing arts. [17] provides a good introduction to the psychological aspects, and [18] provides useful ideas about cooperative autonomous methods. People are especially poor at detecting online deception as witnessed by the many Internet scams [19]. 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 [20].

We consider deception as a surreptitious information-narrowing in a communications channel. A spy or an attacker needs to obtain information about your computer systems; defense could be to "choke" or 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 messages and mission plans. However, effective deception requires the deceivee to believe they are receiving the same amount of information as they expect. Thus a good strategy for fake honeypots is to provide roughly the same amount of information as real systems, but qualitatively different to be detectable by the attacker. This requires careful planning because if we do not do enough modification, the attacker will not think it is a honeypot, but if we do too much, the attacker will see we are trying to deceive them.

B. Experiments with distorted text files

One question is to what extent users notice distortions in text and file listings. [21] pioneered in automated construction of fake online documents, but did not study their effectiveness. [22] developed sophisticated methods for detecting fake documents, but these are beyond the abilities of humans to duplicate. So we set up a simple 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 [13]. The fake file directories were created from real Web-page directories at our school by choosing randomly selected filenames and appending them to a single directory name. The fake 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, two, or 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 either 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 intuition that something was "not right" in order 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 small a sample of directory 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 shorter half of the questions versus 62.5% for longer questions (text varied from 108 to 411 characters in length, and directories from 8 to 18 lines).

From these experiments we conclude that honeypots can be detected from analysis of their text files by an attacker even if the clues are subtle, despite the difficulty in general that people have in detecting deception, and fake 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. Thus fake honeypots need not be obviously anomalous to be noticed by an attacker. It is interesting to measure deception "intuition" because intuition has seen a recent revival of interest in psychology [23].

VI. EXPERIMENTS WITH REACTIONS TO HONEYPOTS

A. Setup

We also conducted experiments with real honeypots to assess whether malicious traffic could be influenced by clues suggesting a honeypot [24]. We used a three-computer configuration: a router, a data-capture computer, and a machine implementing two virtual honeypots in a virtual honeynet (see Figure 1). 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 in December 2005. Snort alerts were sent through the BASE (BAsic Analysis and Security Engine) to a Web front end running on an Apache Web server (which could be attacked too). The honeynet machine used VMware 5.5 to simplify restart after compromise; one virtual machine 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 version 8.1.1.

The machines were attached to an Internet connection that did not go through any firewall, unlike most of our site'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, and added some directories with names of honeypot tools from the Honeynet Project. We also copied some innocent files to the honeypots, like course notes, to help populate their file systems. To encourage unauthorized visitors, 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 42 days (Table 1). During this period we gradually increased the obviousness of the honeypots and built the honeynet, so a metric to measure honeypot effects is to compare the data for the first 21 days with the data for the last 21 days. The table gives pairs of statistics for these two periods for raw alert counts and clustered counts under four kinds of clustering. Time clustering was done by identifying all identical alerts within a certain number of minutes as a single event. It can be seen that the choice of a window for time clustering of 3 minutes, 10 minutes, or 30 minutes did not matter too much. Clustering by remote IP address was not as helpful since we had a significant number of similar attacks from multiple machines (reflecting either spoofing or controlled "zombie" machines).

There was variation in the clustered data with day but it was not excessive (Figure 2); much of this traffic reflects the "background radiation" of the Internet [25] which has been studied and characterized. Classes averaging less than 5 alerts per day appeared to be quite random, so we have not bothered plotting them separately. There was a mild effect of day of the week on the traffic, with Tuesdays and Wednesdays seeing 11% more attack clusters than average and Fridays seeing 13% less; hence it is important to average over a week.

Table 1 shows that both the raw number of alerts and the number of clusters decreased in the second 21 days. Statistically

significant decreases occurred in clusters of a number of important attack categories such as BAD-TRAFFIC, EXPLOIT, MS_SQL, NETBIOS, POLICY, SHELLCODE, WEB-ATTACKS, WEB-CGI, and WEB-IIS, if we define significance as more than two standard deviations away in a binomial model where alerts are drawn from a large population of traffic (then the standard deviation is approximately the square root of the average of the two counts). Significant increases were observed for 10-minute clusters only in the ICMP, INFO, SCAN, and WEB-PHP classes, and only for the SCAN and WEB-PHP raw data. This supports our hypothesis that attackers are recognizing our honeypots and getting scared away. However, we need to discriminate this from other possible hypotheses such as that attacks always decrease over the first few months of a new site (in which case new sites should pretend they are old). Compromise of our systems was infrequent during testing, so that is not a likely explanation of the attack decrease. We will explore this in the future by modifying the features of our honeypots.

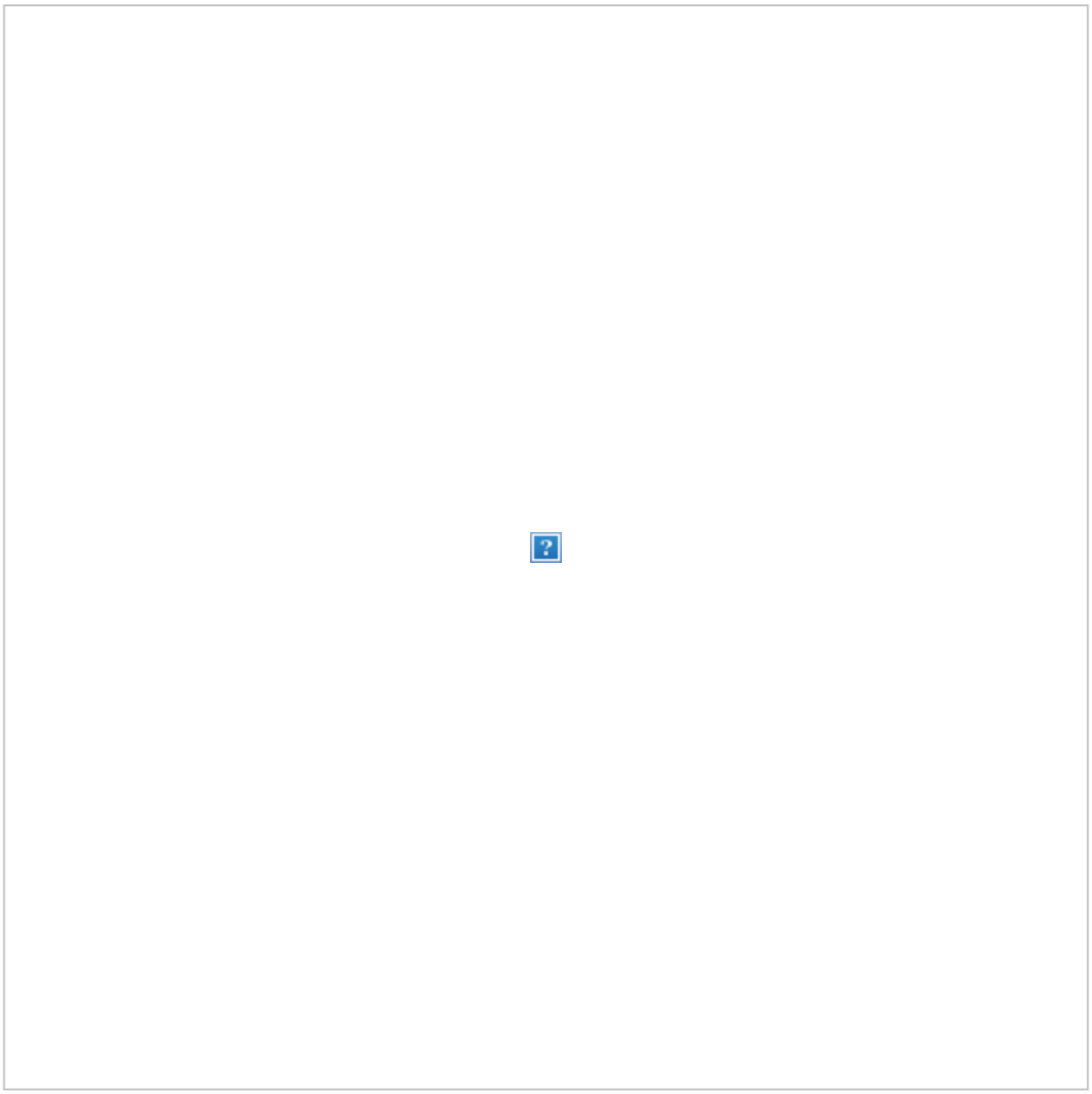

Fig. 1. Honeypot network configuration.

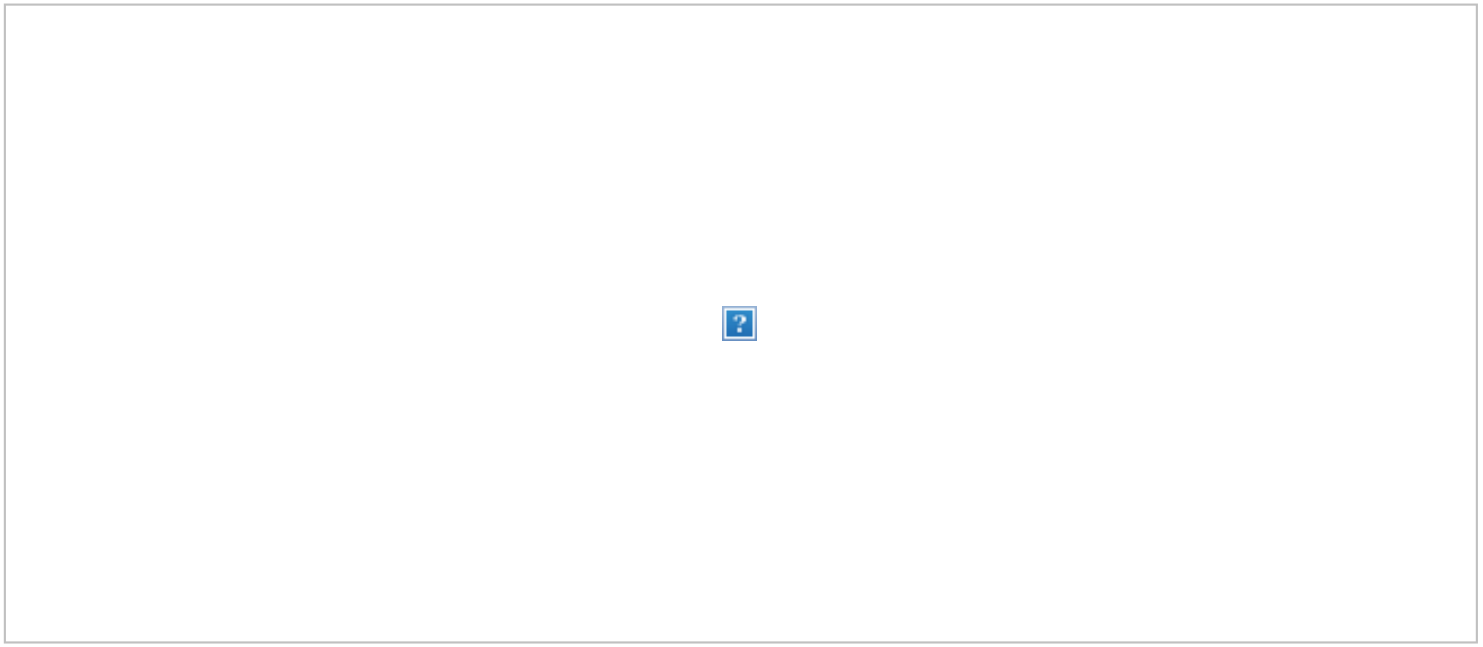

Fig. 2. Variation by day and class in the number of observed time-clusters of major Snort alerts, 1/23-3/3.

TABLE 1: SNORT ALERT COUNTS BY TYPE COMPARED FOR TWO SUCCESSIVE 21-DAY PERIODS ON THE HONEYPOT, 1/23-2/12 VERSUS 2/13-3/5.

Snort Alert Class	Raw Count	Time-Clustered Count, 3 Minute Window	Time-Clustered Count, 10 Minute Window	Time-Clustered Count, 30 Minute Window	Remote-IP-Clustered Count
NETBIOS	46078 / 29484	927 / 899	804 / 690	661 / 480	2302 / 4895
BAD-TRAFFIC	3230 / 1946	708 / 370	671 / 354	550 / 287	729 / 383
MS-SQL	2310 / 2214	2115 / 2013	1818 / 1749	1032 / 1014	2058 / 2034
INFO	2720 / 37	11 / 25	11 / 19	11 / 17	13 / 27
SHELLCODE	1875 / 1448	708 / 583	625 / 493	452 / 313	321 / 258
ICMP	1382 / 656	195 / 234	190 / 226	188 / 214	359 / 352
WEB-PHP	202 / 351	23 / 44	22 / 43	21 / 40	23 / 44
WEB-CGI	152 / 99	43 / 24	39 / 24	37 / 21	51 / 29
WEB-IIS	148 / 18	22 / 11	21 / 10	21 / 10	22 / 11
WEB-ATTACKS	30 / 0	9 / 0	9 / 0	9 / 0	9 / 0
WEB-FRONTPAGE	4 / 1	4 / 1	2 / 1	2 / 1	2 / 1
WEB-MISC	15 / 41	9 / 16	7 / 15	7 / 13	7 / 23
SCAN	32 / 89	30 / 68	30 / 54	26 / 47	25 / 53
POLICY	30 / 24	27 / 18	27 / 18	27 / 17	27 / 17
EXPLOIT	22 / 1	7 / 1	6 / 1	5 / 1	5 / 1
SNMP	12 / 30	6 / 6	6 / 6	6 / 6	6 / 6
ATTACK-RESPONSES	1 / 0	1 / 0	1 / 0	1 / 0	1 / 0

We compared this data with that for our organization's main local network for 14 days in the hope of adjusting the above rates by the observed local trends; our Snort rules came from this installation. But NETBIOS and BAD-TRAFFIC rules were disabled in this installation, and other frequencies differed significantly. MS-SQL, INFO, SHELLCODE, ICMP, SCAN, and POLICY clustered alert counts had Pearson r-correlations respectively of 0.21, -0.74, -0.49, 0.02, -0.33, and -0.28 over the 14 days between the two sites. These were insufficient since we need positive correlations greater than +0.43 for 95% significance at N=14. Clearly attackers are sensitive to the software and configurations they are attacking, and our honeypot looks too different from our organization's network to see many of the same attacks.

VII. THE FUTURE OF FAKE HONEYPOTS

Fake honeypots can be seen as the third step in deception for information warfare after first honeypots and then methods for detecting honeypots [6]. The obvious next question is whether this escalation of responses will occur indefinitely, as with "fake fake honeypots" that are actually real honeypots but pretend to be overly obvious fake honeypots. We believe it will not, much in the way that antivirus software and the 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]. Counter-counterdeception is often difficult to distinguish from just plain deception, and can usually be interpreted as just a good independent strategy for deception, since diversity in deception methods is essential to their effectiveness.

However, we are likely to see an increased rate of measure-countermeasure evolution and escalation in the near future [3]. Virus-antivirus measures went through such a period during the 1990's and it has now slowed down; spam-antispam methods underwent the same sort of flurry of evolution in the last few years which is only now starting to slow down as blacklisting of sites and content analysis of spam has matured [26]. 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 chances of a mistake by the attacker increase in the same way that the number of bugs in software increases with its size. Growth in the category of 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.

REFERENCES

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

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

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

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

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

[6] Tirenin, W., and Faatz, D., "A Concept for Strategic Cyber Defense," *Proc. Conf. on Military Communications*, Atlantic City, NJ, Vol. 1, pp. 458-463, October 1999.

[7] Bell, J., and Whaley, B., *Cheating and Deception.* New Brunswick, NJ: Transaction Publishers, 1991.

[8] Dunnigan, J., and Nofi, A., *Victory and Deceit, second edition: Deception and Trickery in War*. San Jose, CA: Writers Club Press, 2001.

[9] Rowe, N., "A Taxonomy of Deception in Cyberspace," *International Conference on Information Warfare and Security*, Princess Anne, MD, pp. 173-181, March 2006.

[10] Dornseif, M., Holz T., and Klein, C., "NoSEBrEaK ? Attacking Honeynets," *Proc. IEEE Workshop on Information Assurance and Security*, West Point, NY, 1555, June 2004.

[11] Krawetz, N., "Anti-Honeypot Technology," *IEEE Security and Privacy*, *2*(1), pp. 76-79, Jan.-Feb. 2004.

[12] Holz, T., and Raynal, F., "Detecting Honeypots and Other Suspicious Environments," *Proc. 6th SMC Information Assurance Workshop*, West Point, NY, pp. 29-36, June 2005.

[13] Rowe, N., "Measuring the Effectiveness of Honeypot Counter-counterdeception," *Proc. 39th Hawaii International Conference on Systems Sciences*, Poipu, HI, January 2006.

[14] Christian, D., and Young, R. C., "Strategic Deception in Agents," Proc. 3rd Intl. Joint Conference on Autonomous Agents and Multiagent Systems, New York, NY, pp. 218-226, 2004.

[15] Hutchinson, W., and Warren, M., *Information Warfare: Corporate Attack and Defense in a Digital World*, London: Butterworth-Heinemann, 2001.

[16] Vrij, A., *Detecting Lies and Deceit: The Psychology of Lying and the Implications for Professional Practice*, Chichester, UK: Wiley, 2000.

[17] Heuer, R. J., "Cognitive Factors in Deception and Counterdeception," in *Strategic Military Deception*, ed. Daniel, D., & Herbig, K., New York: Pergamon, 1982, pp. 31-69.

[18] Santos, E., and Johnson, G., "Toward Detecting Deception in Intelligent Systems," *Proc. SPIE Defense & Security Symposium*, Vol. 5423, Orlando, Florida, 2004.

[19] Grazioli, S., and Wang, A., "Looking without Seeing: Understanding Unsophisticated Consumers' Success and Failure to Detect Internet Deception," *Proc. 22nd Intl. Conf. on Information Systems*, pp. 193-204, 2001.

[20] Burgoon, J., Stoner, G., Bonito, J., and Dunbar, N., "Trust and Deception in Mediated Communication," *Proc. 36th Hawaii Intl. Conf. on System Sciences*, Honolulu, HI, 44.1, 2003.

[21] Gerwehr, S., Rothenberg, J., and Anderson, R. H., "An Arsenal of Deceptions for INFOSEC," Project Memorandum, National Defense Research Institute, Rand Corp., PM-1167-NSA, October 1999.

[22] Kaza, S., Murthy, S., and Hu, G., "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.

[23] Klein, G., *Sources of Power: How People Make Decisions*. Cambridge, MA: MIT Press, 1999.

[24] Duong, B., ?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.

[25] Pang, R., Yegneswaran, V., Barford, P., Paxson, V., and Peterson, L., "Characteristics of Internet Background Radiation," *Proc. 4th ACM SIGCOMM Conference on Internet Measurement*, Taormina, IT, pp. 27-40, 2004.

[26] Zdziarski, J., *Ending Spam: Bayesian Content Filtering and the Art of Statistical Language Classification,* San Francisco: No Starch Press, 2005.

Manuscript received on May 3, 2006. This work was supported by NSF under the Cyber Trust Program and by Grant DUE-0114018. Contact: Code CS/Rp, 833 Dyer Road, U.S. Naval Postgraduate School, Monterey, CA 93943. Email: ncrowe at nps.edu, btduong at nps.edu, custy at spawar.navy.mil. Opinions expressed are those of the authors and not necessarily those of the U.S. Government.