Computer Sciences Department

Honeygames: A Game Theoretic Approach to Defending Network Monitors

Jin-Yi Cai
Vinod Yegneswaran
Chris Alfeld
Paul Barford

Technical Report #1577

August 2006

UNIVERSITY OF
WISCONSIN
MADISON

Honeygames: A Game Theoretic Approach to Defending Network Monitors

Jin-Yi Cai, Vinod Yegneswaran, Chris Alfeld and Paul Barford
{jyc, vinod, alfeld, pb}@cs.wisc.edu
University of Wisconsin, Madison

Abstract

A honeynet is a portion of routed but otherwise unused address space that is instrumented for network traffic monitoring. Over the past several years, honeynets have proven to be an invaluable tool for understanding the characteristics of unwanted Internet traffic from misconfigurations and malicious attacks. In this paper, we address the problem of defending honeynets against systematic mapping by malicious parties to ensure that they remain viable in the long term. Our first step is to abstract the problem into a simple two player game. The objective of the *Attacker* is to probe a range of address space in order to identify the embedded honeynet. The objectives of the *Defender* are (*a*) to prevent the honeynet from being mapped by periodically shuffling the honeynet's location within the address space and (*b*) to minimize frequency of shuffling. We establish provably optimal strategies for both the attacker and defender. We also consider progressively more realistic variants of the game. Next, we evaluate the strategies analytically to understand how they apply over a range of honeynet configurations. We find that network size, monitor size, presence of unused address space, and probe rates directly impact shuffling frequency. Finally, we discuss experimental results from prototype implementation of a network shuffling middlebox that provides insights on expected resource requirements and performance implications. We show that the system is capable of effectively defending large networks, with limited impact on normal traffic, and responds well in the face of network attacks and anomalies.

1 Introduction

Malicious activity in the Internet is a large and growing problem. The spectrum of threats include DoS attacks, self propagating worms, spam, spyware and botnets. Malicious parties (Black Hats) are constantly looking for new potential victim systems which are typically identified by scanning across portions of the Internet address space. Many tools have been developed to facilitate this task, and there are research papers that describe and evaluate the magnitude of threat posed by novel, highly efficient scanning methods (*e.g.* [23]).

The network security community has recognized that the process of scanning for victims actually offers a unique opportunity to track malicious activity in the Internet. Standard network configurations typically drop packets that are destined for addresses that are not associated with a live system. Changing this configuration to route packets to a monitoring system enables passive collection of scanning packets. Even more compelling is the notion of routing these packets to systems with the ability to engage in conversations with the malicious hosts thereby enabling deep details of attack profiles to be gathered. This configuration is commonly referred to as a *honeynet*.

Over the past several years, a large number of honeynet installations have emerged across the globe which have been used to monitor and identify many important details of malicious activity such as the characterization studies of Internet Background Radiation [16], large worm outbreaks [8], and botnet activity [11]. Furthermore, research on honeynet systems is active and focused on enabling scalable and secure measurement at greater levels of detail.

The successes and the utility of honeynets are not likely to have been lost on the Black Hats. Several recent studies have shown that well known monitoring entities such as Dshield.org [24] can be identified effectively

using different probing methods [7, 20]. If these techniques are adopted by Black Hats (one must assume that they will be adopted if they are not being used already) then blacklists of address space for well known honeynets could be created, which would seriously reduce their continued effectiveness.

In this paper we address the problem of defending honeynets from being mapped. Our first step in addressing this problem is to abstract it to a simple two player game between an Attacker and a Defender. The objective of the Attacker is to identify the contiguous segment of monitored address space (*i.e.,* the honeynet) embedded in a larger segment of address space. The Attacker sends probes to the address space, receiving responses that can be eventually distinguished as coming from a honeynet. The Attacker attempts to identify the honeynet with a minimum number of probes. The Defender has two objectives which must be balanced. The first is to prevent the honeynet from being mapped, which is done by periodically shuffling its location within the larger address space. The second objective is to extend the duration of shuffling epochs in order to minimize demands on the system responsible for shuffling. This is accomplished by delaying shuffling until a maximum probe threshold has been reached.

An important capability in some kinds of Honeynets is that they respond to probes in a protocol compliant way (*e.g.,* [27]). However, one must assume that after a certain number of probes, the Attacker can distinguish a Honeynet from a live host. We model this abstractly by assigning a global lie quota to the Defender. Given this abstract context, we are able to prove optimal strategies for both the Attacker and Defender. Our Main Theorem (Theorem 3.4.2), establishes *unique optimality* of strategies. The proof relies on a key combinatorial lemma, the proof of which was the most mathematically challenging component of our study.

We extend our basic formulation of the game in several ways to provide a foundation for evaluation and practical implementation. First, we consider the possibility of breaking the honeynet into multiple address segments within the larger address space. We show that this adds a small factor per each individual segment to the time required to identify the honeynet, which could be meaningful in a live deployment. Next, we consider several different strategies for the shuffling process. While random shuffling at the granularity of individual IP addresses is optimal, it may not be practical from an implementation perspective. We describe four different policies for shuffling blocks of IP addresses that trade-off frequency of shuffling with the amount of state that is maintained and degree of resiliency to mapping.

Our basic game theoretic formulation plus the extensions enable us to analyze the processes of attacking and defending honeynets analytically. We conduct a series of evaluations over a range of possible configurations to assess the trade-offs between address space size, honeynet size, probe rates and shuffling frequency. We find that the duration of game (*i.e.*, the time between shuffles) increases proportionally with the lie budget (ℓ) and decreases proportionally with the monitor size (m), as dictated by our theorems.

To assess the practical issues of defending honeynets using our game theoretic formulation, we built a prototype middlebox system that we call Kaleidoscope. The basic functionality of this system includes counting probes to unused address space, shuffling honeynet addresses and providing network address translation between the systems outside of the target network and both live and honeynet systems within. We used Kaleidoscope in a series of experiments in a laboratory setting that vary traffic rates, honeynet size, address space size and probe rates. Our prototype implementation demonstrates that the shuffling policies can be implemented efficiently, with minimal latency (under a millisecond) and zero packet loss under normal and attack traffic conditions.

In summary, the contributions of this paper are the following, (*i*) the definition and game theoretic formulation of the problem of defending honeynets, (*ii*) the proof of the optimal strategies for both the honeynet Attacker and Defender, (*iii*) the examination of practical issues in defending honeynets both analytically and through a prototype implementation.

2 Background and Related Work

The utility of unused address space monitoring as a means for tracking malicious activity in the Internet stems from several practical issues. First, networks do not typically terminate their entire allotment of IP addresses with

live systems, and address assignments within networks often change periodically. Second, the effectiveness of the domain name system insures that virtually all non-malicious traffic gets routed to IP addresses that terminate at live hosts (misconfigurations do, however, occur from time to time *e.g.,* [26, 19]). Third, the combination of on-going identification of new vulnerabilities and application of security patches make victim hosts somewhat ephemeral. As a result, scans across routed segments network address space are continually used to identify new victim hosts [28].

Recently, a number of systems have been developed that respond to remote hosts that send traffic to unused address space enabling details of attacks to be gathered. The most simple of these is the Internet Motion Sensor which has the ability to respond to TCP connection requests with a SYN/ACK packet [6]. While this capability is useful, a growing number of attacks require deeper interaction before an exploit is sent. The iSink system described in [27] has the ability to emulate protocol specific interactions in a scalable fashion. The iSink approach is similar in many respects to *honeyd*, which also maintains per-flow state at the cost of scalability [17]. From the perspective of mapping, these systems are perhaps the easiest to identify since their response capability is limited.

Despite the utility of emulation-based monitors, they are limited in that they may be unable to capture details of some attacks, and that they cannot be used to observe an attackers behavior once a host has been compromised (*e.g.,* in the case of a botnet). A solution is to deploy real hosts running real operating systems, but this approach is obviously impractical for anything other than very small honeynets. Thus, the idea of using virtual machines as a platform for monitoring has become popular. VM systems such as VMWare, Xen and Virtual PC can be used to run different versions of different operating systems in a manageable environment thereby enabling a broad range of attacks to be monitored in a somewhat more scalable fashion. There are a number of examples of how these *virtual honeynets* have been used to track attacks and intrusions including [17, 9, 12]. Of particular note is the recent Potemkin Virtual Honeyfarm work which addresses the scalability limitations of standard virtual honeynets enabling emulation of over 64K systems on a small number of physical hosts [25]. While virtual honeynets are somewhat more difficult to identify, it is clear that Black Hats are already aware of their capability and are developing methods to identify them [2].

To the best of our knowledge the problem of defending honeynets from systematic mapping has not been addressed in prior studies. The problem of camouflaging honeypots from detection and analysis is outlined briefly at the end of [25]. The related problem of limiting the utility of hitlists of live systems has been addresses in several studies. Antonatos *et al.* propose *address space randomization* as a method for defending live systems from worms that use hitlists [4]. Their objective of accelerating hitlist decay is somewhat similar to ours, however both their basic approach (in terms of considering hitlists of live systems) and suggested mechanisms for random address shuffling (primarily DHCP) differ from ours. Similar efforts at changing network configuration in order to attempt to thwart malicious activity include [5, 13, 15]. None of these projects propose a game-theoretic formulation for defending live nodes and the methods they develop for address space randomization differ from ours.

3 Foundations for Address Space Shuffling

3.1 Basic Formulation

We model the adversary/honeynet interaction as a 2-person game between an Attacker and a Defender. To simplify the analysis we view a given segment of address space as circular. A contiguous segment of monitored addresses is placed randomly within the address space. We call monitored addresses (*i.e.,* the honeynet) 'black' and all other addresses 'white'. During the game the Attacker queries addresses and receives a reply based on the color of the address queried. White addresses must reply 'white' but black addresses may answer black or 'lie' and answer 'white'. We impose a global limit on the number of lies allowed and for the purpose of establishing

our theoretical result we assume lies can be used flexibly. [1]

Formally, let m, k and ℓ be positive integers. Let $n = mk$. Our address space is a circular array of n cells, identified with the additive group $\mathbf{Z}_n$, *i.e.*, each cell is indexed by some $i \in \mathbf{Z}_n$. We use the following notation:

DEFINITION **3.1.1.** *A* block *is a contiguous subset of* $\mathbf{Z}_n$ *of size* m. *Denote the* j*-th block by* $B_j = \{j, j+1, \ldots, j+m-1\}$. *As an alternate notation, let* $B^j = \{j-m+1, \ldots, j-1, j\}$. *There is a single block of black addresses:* $B^c = \{c-m+1, \ldots, c-1, c\}$.

The game starts with a random spin of B^c, that is, a uniformly distributed $c \in \mathbf{Z}_n$. We assume that both players know the parameters m, k and ℓ. The position c is known to the Defender but unknown to the Attacker whose aim is to find it. In a round of the game (*i.e.* between shuffling epochs), the Attacker moves first and the two players alternate their moves until the Attacker knows c. A move of A consists of a *query* at a cell j. If $j \notin B^c$ then D replies with the bit $b = 0$. If $j \in B^c$ then D has a choice: D can either answer $b = 1$ or, *commit a lie* by answering $b = 0$. However, D can only lie $< \ell$ times in total. In short, n is the size of the circular address space, m is the length of a monitor block, $k = n/m$, and ℓ is the *quota* of *lies*.

Denote by A (respectively D) a strategy by the Attacker (respectively the Defender). We will primarily analyze *pure* (non-randomized) strategies; but with a slight abuse of notation we will use the same notation for randomized strategies. Let $V = V(A, D)$ denote the number of queries A makes against D until A learns the value c. Thus V is a variable randomized over the uniform choice of $c \in \mathbf{Z}_n$ (and possibly over the randomization of the randomized strategies of the two players). Let v denote the expectation

$$v = v(A, D) = \mathbf{E}[V(A, D)]. \tag{1}$$

The objective of the Attacker is to minimize this quantity v. The objective of the Defender is to maximize v thereby enabling the maximum amount of data to be gathered in the honeynet between shuffles.

3.2 The Strategies: Round-Robin (RR) and Delay-Delay (DD)

Before giving a more detailed analysis of this game, we make some general observations. Suppose, during a round of the game, A learns a cell $j \in B^c$. If $m = 1$, then $c = j$ and the game is over. If $m > 1$, then, because S occupies a contiguous segment of cells, A can be sure that B^c is located in one of the following m possible ways (recall that addition in $\mathbf{Z}_n$ is done modulo n):

$$c = j, j+1, \ldots, j+m-1. \tag{2}$$

Then A can 'zero-in' on the boundaries of B^c by performing the following binary search. First, A queries the cells $j^- = j - \lfloor m/2 \rfloor$ and $j^+ = j + \lfloor m/2 \rfloor$. At least one of j^- and j^+ must belong to B^c. This is because the number of cells located strictly in between these two cells is $2\lfloor m/2 \rfloor - 1 < m$. Thus, D must either choose to answer truthfully $b = 1$ at least once, or (if possible) commit an additional lie. If A receives both answers with $b = 0$, then he knows that a lie has been committed. Since A knows that D can commit only $< \ell$ lies, A continues to query j^+ and j^- until D answers $b = 1$. Assume D answers $b = 1$ to the query j^+; the case for j^- is symmetric. At this point, A knows that c is among the following $\lceil m/2 \rceil$ locations:

$$c = j^+, j^+ + 1, \ldots, j+m-1. \tag{3}$$

So A can effectively consider a 'shrunken' version of the problem as follows. Assume j^+ is in B^c (the j^- case is symmetric). Identify the cells between j and j^+ as a single cell. This identification reduces the length of the block by $\lfloor m/2 \rfloor$ to $\lceil m/2 \rceil$.[2] Now we can consider the placement of the shortened block on the condition that it contains a certain cell (the identified j and j^+). This process is a general step in the binary search.

[1] The reason for this is that, in practice, it may take multiple probes to distinguish honeynet addresses.

[2] We are abusing notation here by allowing blocks of different lengths. This paragraph is the only instance where we discuss non-m length blocks.

It is clear that every step of this 'binary search' reduces the length from m' to $\lceil m'/2 \rceil$. When the length has shrunk to 1, the game is over. If $2^{r-1} < m \leq 2^r$, then it takes exactly $r = \lceil \log_2 m \rceil$ steps to reduce the length to 1. If A queries j^- and j^+ in random order, then on average each step takes 1.5 queries. Let ℓ' be the number of lies D is allowed to commit ($\ell' < \ell$ and may be much less due to lies committed prior to the 'binary search'). Then we have an upper bound of $2\ell' + 1.5\lceil \log_2 m \rceil$ queries under this Attacker strategy, valid for any D.

We see that once the Attacker gains the knowledge of one $j \in B^c$, A can zero-in fairly rapidly. Therefore, a rational Defender will not reveal any $j \in B^c$ until absolutely necessary. This reasoning leads us to the following Delay-Delay (DD) strategy for the Defender.

DEFINITION **3.2.1.** *The strategy* Delay-Delay *(DD) is as follows. Lie as long as the quota of lies allows.*

DEFINITION **3.2.2.** *The* First-Time, *T, is the time (number of queries made) when the Attacker first learns some $j \in B^c$.*

Clearly, against any Attacker strategy, DD maximizes T among all Defender strategies.

DEFINITION **3.2.3.** *The strategy* Round-Robin *(RR) is as follows. Pick any cell to start, e.g., $j = 0$. Query j consecutively ℓ times. Then set $j := j + m$, and repeat. As soon as a reply of $b = 1$ is received switch to a 'binary search' strategy to zero-in.*[3]

Note that with at most $k\ell$ queries, RR *will* obtain a $b = 1$ answer, and commence its 'binary search'. Indeed, when RR has made $k\ell$ queries, these would have been for each $j = im$ for $0 \leq i < k$ and each such j is queried exactly ℓ times. For any c, some query j will be in B^c (in particular, for $i = \lfloor c/m \rfloor$). For any D, up to ℓ repeated queries at this j must yield an answer $b = 1$. On average, RR will take $\frac{1}{k}\sum_{i=1}^{k} i\ell = (k+1)\ell/2$ queries against DD to arrive at its First-Time T. Therefore,

$$V(\mathrm{RR}, \mathrm{DD}) \leq k\ell + 2\lceil \log_2 m \rceil \tag{4}$$

in the worst case, and

$$v(\mathrm{RR}, \mathrm{DD}) \leq (k+1)\ell/2 + 1.5\lceil \log_2 m \rceil \tag{5}$$

on average. This upper bound is quite sharp; if m is a power of 2, then the exact equality holds. For any m, the following lower bound holds:

$$v(\mathrm{RR}, \mathrm{DD}) \geq (k+1)\ell/2 + \lceil \log_2 m \rceil. \tag{6}$$

Against any other D, after First-Time T, each lie can buy at most two queries. Thus,

$$v(\mathrm{RR}, D) \leq k\ell/2 + 1 + 1.5\lceil \log_2 m \rceil + 2(\ell - 1). \tag{7}$$

In view of this we will only consider Delay-Delay as the strategy for the Defender in the following.[4]

3.3 Delay-Delay against any Attacker

In this section we give a simple lower bound for $v(A, \mathrm{DD})$. The bound here partially justifies our adoption of DD for the Defender exclusively, namely (1) DD performs well against any A, and (2) for one particular A, namely

[3] If the game is slightly modified so that Defender picks c in secret, then a modified Round-Robin strategy where the Attacker randomly picks a starting j will achieve the same performance as below.

[4] Curiously, if the Defender knew that the Attacker is RR, then he should not commit any lie during the first stage up to the First-Time T (the opposite of DD) making $\mathbf{E}[T] = k\ell/2 + 1$, and save his entire quota of lies for the 'binary search' stage where each lie buys him two query steps against RR. If instead the Defender uses DD against RR, then each lie committed up to First-Time T buys him only one query step.

$A = \mathrm{RR}$, DD performs almost as well as any D (Section 3.2).

DEFINITION **3.3.1.** *For an Attacker A and Defender D, the* Capitulation-Time, *$L = L(A,D)$, is the first time (the number of queries made) when A has made ℓ queries in B^c against D.*

Against DD the Capitulation-Time is the same as First-Time. For an arbitrary pair (A,D) it is possible that the game ends (when A learns c) before A ever hits ℓ times in B^c. In this case we define $L = \infty$.

Observe that $v(A,DD) \geq \mathbf{E}[L]$ (because against DD, without ℓ hits, all the answers A gets are $b = 0$, *i.e.*, A gets no information, and therefore the game can not end), and, for any reasonable A, it takes no more than $O(\log m)$ extra queries to locate B^c after L. As such, we will only be concerned with a lower bound for $\mathbf{E}[L]$.

Consider the first p queries of A. We say that a query at j is a *hit* if $j \in B^c$. Let H_p be the number of hits for the first p queries by A against DD. If we think of H_p as a sum of 0-1 random variables $X_1 + \ldots + X_p$, where $X_i = 1$ iff the i-th query is a hit, then it seems reasonable to conclude that $\Pr[X_i = 1] = m/n = 1/k$, as a hit is to pick a cell among the m cells of B^c which are located randomly within $\mathbf{Z}_n$. Then, by linearity, $\mathbf{E}[H_p] = p/k$. It follows that if $p \leq k\ell/2$, then the expected number of hits is at most $\ell/2$. Then, by Markov's inequality, $\Pr[H_p \geq \ell] \leq 1/2$. Since L is the number of queries A needed to get at least ℓ hits, by Markov's inequality again, for any $i \geq 1$,

$$\mathbf{E}[L] \geq i\Pr[L \geq i] = i\Pr[H_{i-1} < \ell]. \tag{8}$$

Pick $i = \lfloor k\ell/2 \rfloor + 1$, then

$$\Pr[H_{\lfloor k\ell/2 \rfloor} < \ell] \geq 1/2. \tag{9}$$

It follows that

THEOREM **3.3.2.**

$$\mathbf{E}[L] \geq (\lfloor k\ell/2 \rfloor + 1)/2 > k\ell/4.$$

However, the above argument has a subtle flaw. The problem is with the claim $\Pr[X_i = 1] = 1/k$. The i-th query by A may depend on answers by DD to previous queries, which in turn is influenced by where the secret c is randomly placed. In the following we rigorously establish (9). Then Markov's inequality from (8) proves Theorem 3.3.2.

Let $t = \lfloor k\ell/2 \rfloor$. Let $q_1, q_2, \ldots, q_t$ be the first t cells A queries in a hypothetical round of the game when the first $t-1$ answers the Defender gave are all $b = 0$. These locations are defined by A alone, and not dependent on where c is. We say $a \in \mathbf{Z}_n$ is a *heavy* point iff a belongs to at least ℓ many blocks B_{q_j}, for $1 \leq j \leq t$.

LEMMA **3.3.3.** *The (secret) random c is a heavy point iff in a play of A against DD, $H_t \geq \ell$.*

Proof. Suppose c is a heavy point. Let j be the minimum in $1 \leq j \leq t$, such that c is covered ℓ times by $B_{q_1}, \ldots, B_{q_j}$. Then in a round against DD, the first $j-1$ answers A gets from DD are all $b = 0$. Therefore $q_1, q_2, \ldots, q_j$ are in fact the first j queries made by A against DD. It follows that there are ℓ hits among them, as B^c contains q iff c is covered by B_q.

Conversely, suppose $H_t \geq \ell$ when A plays against DD. Let j be the minimum in $1 \leq j \leq t$ such that $H_j = \ell$. Then the first $j-1$ answers A gets from DD are all $b = 0$. Therefore the first j queries made by A against DD are in fact $q_1, q_2, \ldots, q_j$. By $H_j = \ell$, it follows that c is covered ℓ times among $B_{q_1}, \ldots, B_{q_j}$. □

We now prove (9). No matter what $q_1, q_2, \ldots, q_t$ are chosen by A, counting with multiplicity the blocks $B_{q_1}, \ldots, B_{q_t}$ cover $tm \leq k\ell m/2$ points. Thus, at most $km/2 = n/2$ points can be covered with multiplicity at least ℓ. It follows that the random c is among them with probability $\leq 1/2$. Then the Lemma implies $\Pr[H_t \geq \ell] \leq 1/2$. This is (9).

Note that our bound on $\mathbf{E}[L]$ is within a factor of 2 of the case when the Attacker uses RR against any Defender strategy D. Once the Attacker is at the Capitulation-Time, all that remains is to pinpoint c. Doing so takes $\Omega(\log m)$ time. Thus we arrive at the following corollary.

COROLLARY **3.3.4.** $v(A,DD) > kl/4 + \Omega(\log m)$.

3.4 Round-Robin is optimal against Delay-Delay

In this section we establish our main result: Theorem 3.4.2. We want to prove that, for any Attacker A, $\mathbf{E}[L(\mathrm{RR},\mathrm{DD})] \leq \mathbf{E}[L(A,\mathrm{DD})]$. In fact, we will prove a stronger theorem of *unique optimality*.

DEFINITION **3.4.1.** *An Attacker strategy A is* essentially Round-Robin *(eRR) if it is of the following form. The strategy A first makes ℓ queries at some cell j. It then updates j to j' where $j' \equiv j \pmod m$ and j' has not been queried and makes ℓ queries at j'. The strategy repeats this behavior until it finds some $j \in B^c$,* i.e., *receives $b = 1$, at which point it switches to a 'binary-search'.*

Clearly,

$$\mathbf{E}[L(\mathrm{eRR},\mathrm{DD})] = \mathbf{E}[L(\mathrm{RR},\mathrm{DD})]. \tag{10}$$

THEOREM **3.4.2** (Main Theorem). *For any Attacker A that is not eRR,*

$$E[L(RR,DD)] < E[L(A,DD)]. \tag{11}$$

To prove the theorem we will need the following combinatorial lemma.

LEMMA **3.4.3.** *Let m, k, p and ℓ be positive integers. Let $n = mk$. Let $\mathbf{Z}_n$ be a circular array of n cells. Let $\mathcal{S}$ be a multi-set $\{S^1, S^2, \ldots, S^p\}$, where each $S^i \subset \mathbf{Z}_n$ is a contiguous segment of cells, we call it a* block, *and each $|S^i| = m$. Then, the following upper bound on the number of heavy points holds:*

$$|\{c \in \mathbf{Z}_n \mid c \text{ is heavy}\}| \leq m \cdot \lfloor \frac{p}{\ell} \rfloor.$$

Mathematically the proof of this Lemma is the most challenging and reveals the most subtle aspect of this game. It is easy to prove a weaker version of the upper bound, $\leq \lfloor m \cdot \frac{p}{\ell} \rfloor$. This follows from a basic volume argument. But we need the stronger version to prove the Main Theorem.

Let $p < k\ell$ and write $p = q\ell + r$, where $q = \lfloor \frac{p}{\ell} \rfloor < k$ and $0 \leq r < \ell$. For any strategy, a query at j defines a block $B_j = \{j, j+1, \ldots, j+m-1\}$. It is obvious that j hits B^c iff c belongs to the block B_j, *i.e.,* $j \in B^c$ iff $c \in B_j$. Now it is clear that the upper bound in the Lemma is achieved by RR (and eRR), since each batch of ℓ repeated queries defines a block, each of which is repeated ℓ times, and these blocks are spaced exactly m cells apart from one batch to the next. Thus, the first $q \cdot \ell$ queries define q distinct and disjoint blocks each repeated ℓ times, and together they cover $m \cdot q$ *heavy* points. The point of the lemma is that one cannot possibly use the left-over 'capacity' of rm to cover more *heavy* points. The proof is presented in the Appendix.

Proof of Theorem 3.4.2. We are concerned with

$$\mathbf{E}[L] = \sum_{i=1}^{\infty} \Pr[L \geq i] = 1 + \sum_{i=1}^{\infty} \Pr[L > i]. \tag{12}$$

Our goal is to show that *RR* minimizes $\mathbf{E}[L]$. Let $\mathbf{E}_A[L] = \mathbf{E}[L(A,\mathrm{DD})]$, the expectation of the Capitulation-Time for Attacker A against Defender DD. Similarly, $\Pr_A$ indicates the probability for Attacker A against Defender DD. Our first goal is to prove the (non-strict) dominance of RR:

$$\mathbf{E}_A[L] \geq \mathbf{E}_{\mathrm{RR}}[L]. \tag{13}$$

We establish $\mathbf{E}_A[L] \geq \mathbf{E}_{\mathrm{RR}}[L]$ by proving term by term $\Pr_A[L > i] \geq \Pr_{\mathrm{RR}}[L > i]$. Observe that if $i \geq k\ell$, then $\Pr_{\mathrm{RR}}[L > i] = 0$. Thus, we only need to consider $i < k\ell$. The above inequality is equivalent to $\Pr_A[L \leq i] \leq \Pr_{\mathrm{RR}}[L \leq i]$. Recall H_i is the number of hits B^c received among the first i queries. Then the event $[L \leq i]$ is equivalent to $[H_i \geq \ell]$. Thus, we seek to show for all i,

$$\Pr_A[H_i \geq \ell] \leq \Pr_{RR}[H_i \geq \ell]. \tag{14}$$

Imagine that we are given the first i (not necessarily distinct) query points. Each query point j defines a block B_j. As we remarked after Lemma 3.4.3, j hits B^d iff d belongs to the block B_j. Let S_i be the configuration consisting of i blocks corresponding to the queries the Attacker would make *assuming the Defender answers b=0* to the first $i-1$ queries. We claim

$$\Pr_A[H_i \geq \ell] = \frac{C(S_i)}{n}. \tag{15}$$

The equality is clear so long as only $b = 0$ answers are received. We prove the result for any interaction by induction. For $i = 1$ all previous (0) queries are vacuously $b = 0$ so the result holds. Assume the result holds up to $< i$. The probability $\Pr_A[H_i \geq \ell]$ is $1/n$ times the number of d such that B^d was hit at least ℓ times during the first i queries. Either B^c was hit at least ℓ times during the first $i-1$ queries, or it was *only* hit ℓ times counting the i-th query. Thus,

$$\Pr_A[H_i \geq \ell] = \Pr_A[H_{i-1} \geq \ell] + \Pr_A[H_i = \ell \wedge H_{i-1} < \ell]. \tag{16}$$

By induction $\Pr_A[H_{i-1} \geq \ell] = \frac{C(S_{i-1})}{n}$. Now, for the second term, the condition $H_{i-1} < \ell$ implies that DD answers the first $i-1$ queries with $b = 0$. Thus, the probability $\Pr_A[H_i = \ell \wedge H_{i-1} < \ell]$ counts the number of heavy points $d \in \mathbf{Z}_n$ which are heavy in S_i but not heavy in S_{i-1}. It follows that the sum of these two probabilities is exactly $\frac{C(S_i)}{n}$, completing the induction.

By Lemma 3.4.3, the probability $\Pr_A[H_i \geq \ell]$ is maximized by RR. Therefore, by the dominance of RR, term by term, we obtain $\mathbf{E}_{\mathrm{RR}}[L] \leq \mathbf{E}_{\mathrm{A}}[L]$.

To prove the strict dominance of RR (and eRR) over any other A against DD, we reason as follows. If the first ℓ queries are not at the same cell, then at the ℓ-th query, RR produces exactly m heavy points while A has strictly less. Thus, at the ℓ-th term in the above infinite sum for $\mathbf{E}_A[L]$, the inequality $\Pr_A[H_\ell \geq \ell] < \Pr_{\mathrm{RR}}[H_\ell \geq \ell]$ is strict. As we have the (non-strict) dominance of RR for every term we arrive at $\mathbf{E}_{\mathrm{RR}}[L] < \mathbf{E}_A[L]$.

We now assume that the first ℓ queries are at a single cell and apply the same argument. Let j_1 be the location of the first ℓ queries. If the next ℓ queries are not at some cell j_2 then consider the time step 2ℓ. At 2ℓ we have a strict inequality and a (non-strict) dominance of RR elsewhere, which again gives $\mathbf{E}_{\mathrm{RR}}[L] < \mathbf{E}_{\mathrm{A}}[L]$. This argument proves that to be optimal, the locations of the queries $j_1, j_2, \ldots$ must be repeated ℓ times each in succession.

Finally, consider the possibility of two query locations j and j' with $j \not\equiv j' \pmod{m}$. Then at time step $i = k\ell$, RR produces a *perfect* cover of all $\mathbf{Z}_n$ with multiplicity ℓ. Meanwhile, A has an *imperfect* cover of all $\mathbf{Z}_n$ which produces a strict inequality in favor of RR. So again, we find that $\mathbf{E}_{\mathrm{RR}}[L] < \mathbf{E}_A[L]$.

This proves the strict optimality of RR (and eRR). □

4 Extension to Multiple Monitoring Blocks

We now briefly discuss the situation when there are multiple monitoring blocks present in the address space. We assume for simplicity that the monitor blocks are pairwise disjoint and each has length m. Assume there are b monitor blocks, $b < k$, and $n = km$ is the total size of the circular address space identified with $\mathbf{Z}_n$ as before.

The Attacker can still start with a Round Robin. Randomly pick a cell $j_0 \in \mathbf{Z}_n$ to start. If the current query cell is j, then query it until receiving an answer $b = 1$, or have queried it ℓ times. Then replace j by $j + m$. We repeat this until $j = j_0$ again. At this point, we will have queried $k = n/m$ locations. At each such location j we record the final bit b_j. Notice that these bits are all correct answers.

We will show that, in the presence of multiple monitoring blocks, this Round Robin strategy followed by a certain 'one-sided binary search' (we denote it by OSBS) can achieve essentially as good an upper bound, as

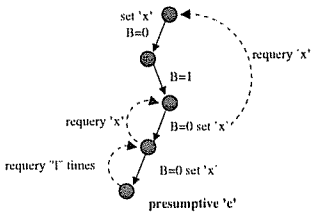


Figure 1: Example OSBS traversal

compared to the case when there is only one block of length m, regardless of Defender's strategy. This justifies our consideration of *only* this Round Robin strategy for A. At the same time, a suitable Defender's strategy can achieve essentially the same quantity as a lower bound for the Defender, and therefore we have characterized the game.

Consider those bits $b_j = 1$. These indicate the presence of the monitoring blocks. We observe that no two bits can refer to the same block and each block must make its presence known through one of these bits. The bits $b_j = 1$ partition the k queried locations into continguous runs of 1's separated by 0's in a circular array of k bits. We will concentrate on one such run of 1's. There is no inteference between different runs of blocks indicated by the corresponding runs of 1's.

Now consider one particular run of 1's, and without loss of generality let 0 be the right-most location where we have a bit $b_0 = 1$. This bit indicates the presence of a block, B^c, where $c \in \{0, 1, \ldots, m-1\}$. OSBS will try to determine the location of c. After that, it can remove this (right-most) block B^c (in the run). If the run has more blocks left, we repeat OSBS on the new right-most block $B^{c'}$, with the slight modification that the right-most location for c' is $c-m$, *i.e.*, $c' \in \{-m, \ldots, c-m\}$. As $c \leq m-1$, the range is $c+1 \leq m$. Therefore the bound we prove for OSBS on B^c applies to every block in the run.

Now consider OSBS on B^c. First we assume D does not commit any lies during this search. This is just ordinary binary search. If $m \geq 2$, our first query is at $i = \lfloor m/2 \rfloor$. The answer maybe 0 or 1, indicating $c < i$ or $c \geq i$, and we continue until finding c. It is well known that this takes $\lceil \log_2 m \rceil$ steps.

Now we have to consider the complication that D can lie a total of $< \ell$ times. However, it can only lie in one direction, namely if the query i is such that $c < i$, the answer must be $b = 0$. Only when $c \geq i$, does the Defender D have the option of lying with $b = 0$, while the true answer is $b = 1$.

OSBS algorithm: In view of this, we carry out our OSBS as follows. We select our next new query point just as in ordinary binary search, as if there were no lies. Record the answer bits as $\beta_1\beta_2\ldots$, where for each $\beta_i = 0$ or 1 we branch left or right respectively. If any $\beta_i = 0$ at a cell x, however, we modify ordinary binary search as follows. We will come back to do a 'repeated querying' at this location x, one query per each step, as we descend in the binary search to its left, until either:

- (*a*) we reach bottom (finding a presumptive c). If so, confirm $\beta_i = 0$ by querying ℓ times, else goto (b).
- (*b*) we discover $\beta_i = 0$ at x was a lie. If so, abandon work to the left of x, resume at x branching to the right.
- (*c*) have made ℓ queries at x, confirming $\beta_i = 0$ is true. If so, resume OSBS with no more queries at x.
- (*d*) got a new bit $\beta_j = 0$ at a cell y, further down in the binary search (*i.e.*, $j > i$). If so, replace y with x and proceed recursively.

Note that in case (*b*), it is possible that there is a 'higher' x' in the binary search tree where the corresponding bit $\beta_{i'} = 0$, and the 'repeated querying' at x' was suspended due to x by case (*d*) for x'. At this point we resume the 'repeated querying' at x'. The search ends by finding c and the confirmation that the smallest cell to its right for which the bit $\beta_i = 0$ is not a lie (by having made a total of ℓ queries there). If all $\beta_i = 1$, then we must end in $c = m-1$. This must be correct and there were no lies committed. In Figure 1, we illustrate OSBS traversal with an example.

We claim that we make at most $O(\ell + \log m)$ queries in total. It is clear that this bound holds if there is no lies committed. The $O(\log m)$ pays for the 'binary search' and $O(\ell)$ pays for the final confirmation.

Suppose at some step during OSBS a lie was committed at x. We make the following observation: in the descent in the binary search to the left of x, *all* subsequent answers with a $\beta = 0$ are lies. In this case, every step of the 'repeated querying' at x, as well as recursively all 'repeated querying' at locations to the left of x with a $\beta = 0$ in OSBS is charged to a total quota of ℓ lies. Each of ℓ lies pays for $O(1)$ queries. This pays for the ordinary binary search queries at new locations descending from x as well. We also note that in case (c) above, a confirmation at x for a $\beta_i = 0$ also confirms all 'higher' x' in the binary search tree where the bit $\beta_{i'} = 0$.

In short, OSBS makes $O(\ell + \log m)$ queries in total. Corresponding to an ordinary binary search, it does the 'repeated querying' which costs at most two queries per each ordinary binary search step. In addition, when OSBS discovers a lie at x, it abandons the portion of the work done after x. But that amount of work is proportional to the number of queries made at a location with a lie, and therefore costs $O(\ell)$. Every lie $\beta = 0$ is eventually discovered. The total amount of work consists of (1) at most double the work in ordinary binary search, and (2) the abandoned work (due to the discovery of lies) which is at most $O(\ell)$. Finally it costs $O(\ell)$ to confirm the answer. This proves the upper bound $O(\ell + \log m)$.

It is also clear that $O(\ell + \log m)$ is optimal up to a constant factor. The Defender can use $O(\ell)$ to delay, and information theoretically it takes $O(\log m)$ to find the $c \in \{0, \ldots, m-1\}$.

Overall, for b blocks, as $b < k$, we get a simultaneous upper and lower bound $\Theta(k\ell + b(\ell + \log m)) = \Theta(k\ell + b \log m)$. The lower bound means that the Defender can achieve this on the average. If $k\ell \geq b \log m$, then clearly it requires this much with DD (even with only one block). If $k\ell < b \log m$, we can imagine all b blocks are separated, then information theoretically we have the bound $\Omega(b \log m) = \Omega(k\ell + b \log m)$.

5 System Design and Implementation

The theorems in the prior sections provide a foundation for developing a system for protecting honeynets from being mapped. In this section, we describe various shuffling strategies and the prototype implementation of a honeynet defense system.

5.1 Shuffling Strategies

We describe four possible strategies for shuffling honeynets within a larger address space after a specified number of probes to that address space have been counted (*i.e.,* a shuffling epoch). We assume shuffling will be done by a system that is interposed between the Attackers and the address space with honeynets and live systems.

• *Restricted Swap Shuffling (RSS).* In this strategy, we shuffle only over a portion of the entire address space. The parameters to the shuffler include the internal address space and the black segments in the local network. The shuffling algorithm randomly modifies the starting address of each black segment so it maps to another equally sized segment (which could be purely black, purely white or span boundary of black and white segments). This strategy allows two black segments to overlap. The advantage of this strategy is that at any point in time only a portion of the address space is affected. Thus, we need to maintain connection state only for segments affected in the current epoch and adjacent epochs. *Restricted No-Overlap Shuffling (RNOS)* is a straightforward extenstion to RSS where we do not allow black segments to overlap. If the random locations of two black segments happen to overlap, we simply re-shuffle until there is no overlap.

• *Discrete Random Map Shuffling (DRMS).* In this strategy, we divde the internal address space into discrete equally sized segments. The parameters for instantiation of the DRMS element include the internal address space, the discrete segment size and the starting points of black segments in the local network. At the beginning of each epoch, the shuffling algorithm generates a random permutation to remap all segments and randomly shifts all segments by a small offset (less than length of a segment). Under this strategy, white segments could get remapped to other white segments, thus requiring us to maintain continuous connection state across all segments. A perfectly randomly shuffling strategy could be acheived by simply setting the segment size to 1. However, the overhead of such a design choice would be significant for large network blocks.

• *Per Source Shuffling (PSS).* This strategy is similar to RSS except that it maintains a different view of the network for each external host that sends packets. It assumes that sources act independently, and thus we need to shuffle only when a single source has reached the probing threshold. At that point, we shuffle the network simply for that external source. The obvious overhead for this strategy is the amount of resources allocated to maintain the network maps for each external host. However, this potentially enables the system to shuffle less frequently, and thus may devote fewer resources to connection entries. Perhaps more significantly, connections from benign hosts (hosts that target only one or a small number of internal hosts) would never get shuffled. This might be a feasible design choice for networks with a small number of black segments.

• *Source Group Shuffling (SGS).* This strategy balances the benefits of RSS and PSS strategies, by maintaining network maps for "groups" of sources. In our prototype implementation, we group sources by their target port numbers. The intuition here is that sources that are coordinated (distributed, but controlled by a single agent) typically probe the network looking to exploit a specific set of vulnerabilities. Hence they target the same set of services. For each group of sources, we shuffle their view of the network if the group has exceeded the specified scanning threshold. This approach is more resilient against distributed network mapping and provides some of the benefits of PSS.

5.2 The Kaleidoscope Prototype

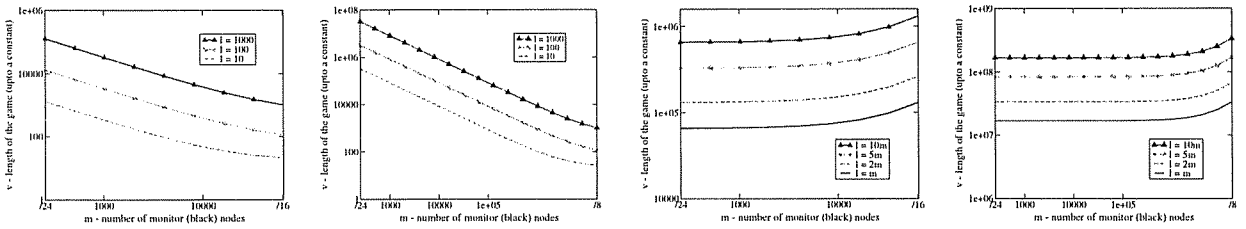
We developed a prototype shuffling middlebox as a collection of network elements on top of the Click modular router platform [14]. We call this system Kaleidoscope. Our decision to use Click is based on its proven scalability and the flexibility provided by its network configuration language. Our enhancements to Click involve the development of two classes of network elements.

- **Address modification element.** This element handles (*i*) packet input (including probe counting), (*ii*) invokes the address shuffling elements to obtain address mappings, (*iii*) and rewrites addresses in the IP header.
- **Shuffling elements.** These elements maintain state and implement shuffling policies. They are programmed as Click 'information elements' and do not have inputs or outputs. The elements maintain the mapping tables and do not directly modify network packets. This design makes it easy to specify new shuffling components and also swap between shuffling strategies. We implemented information elements for each of the shuffling policies discussed above (RSS, RNOS, DRMS, PSS and SGS).

The design choice between shuffling strategies involves a tradeoff between the amount of state maintained and frequency of shuffling. Our system maintains two types of state: current address-mapping tables and connection pool (of live connections). Strategies that maintain global state (RSS and DRMS) devote fewer resources to maintaining address mappings. However, they end up shuffling more frequently, and thus are likely to devote more resources in maintaining ongoing connection state across shuffling epochs *i.e.* larger connection pools. In contrast, strategies that maintain per-source state (PSS and SGS) suffer from the overhead of maintaining multiple address-mapping tables, but they shuffle less frequently compared to the former strategies, and thus require smaller connection pools.

Connection pools. We implement connection pools using the HashBelt data structure that we designed, which approximates the LRU eviction policy and periodically expires stale connections. The HashBelt maintains a list of N hash tables and supports three operations `find`, `insert`, `delete` and `rotate`. On every `find` operation, connection entries are automatically moved to the head (first hash table) in the HashBelt. On every `rotate`, we insert a new hash table to the head of the HashBelt and delete its tail (expiring all the entries in the bottom). In our experiments, we maintain HashBelts with five hash tables and rotate when average chain length exceeds three lookups.

For every connection we maintain the following tuple of mapping information: <external_ip, external_port, advertised_ip, local_ip, local_port>. The advertised_ip is simply the internal address as seen by the exter-



(a) ℓ is an integer multiple; left (n=/16), right (n)=/8

(b) ℓ varies with respect to m; left (n = /16), right (n=/8)

Figure 2: Expected shuffle frequency as we vary ℓ and m.

nal host, and the local_port is identical to the port that is advertised. We use the combination of external_ip, external_port and local_port as a unique identifier for each connection.

Address maps. Address maps keep track of black and white segments and translation between local and advertised IP address. For per-source shuffling, we re-use the HashBelt data structure to periodically expire address-maps for stale sources. For the other strategies, we simply use a hash table or collection of hash tables to maintain the address maps. Each hash table maintains an entry per black segment, except for the case of DRMS which incurs an entry per segment.

6 Experimental Evaluation

6.1 Analytic Results

We use the equations developed in Sections 3 and 4 to analytically evaluate the implications of the monitor size, the lie threhold parameter and presence of segmented monitor blocks on required shuffling frequency. These analyses provide insight into feasibility and practicality of our game theoretic method for protecting honeynets.

• **Impact of lie budget and monitor size on shuffling frequency.** In Figure 2 we consider the impact of lie budget (ℓ) and monitor size (m) on expected shuffling frequency. We plot the expected length of the game for network sizes (n) = /8 (16M IP addresses) and /16 (64K IP addresses). In the left set of graphs, if the value of ℓ is independent of size of m, then we see that the expected duration of the game decreases roughly proportionally with monitor size. This reflects the fact that once the lie budget is exhausted, RR results in efficient identification of the honeynet. While interesting from the perspective of analysis, we argue that in practice it is more likely that ℓ is proportional to monitor size *i.e.,* it will take on average $O(\ell)$ probes per node to determine if it is 'black' or 'white'. The second pair of graphs reflects this case showing that the expected duration of the game increases with monitor size. For *e.g.,* a /16 honeynet in a /8 address space would require over 80M probes to identify (for ℓ = 10M) which would require significant resources for an attacker or a long time.

It is important to consider the tradeoff between larger monitor size and benefit in terms of perspective they provide for monitoring. We consider a simple measure of the *utility* of a monitor segment – the number of source IP addresses observed. For this analysis, we assume that the distribution of sources with respect the volume of scans they generate follows a Zipf distribution [28] and that sources choose destinations randomly. We fix the network size to be a /16, the number of scans to be 5M and the number of sources to be 20K. [5] In Figure 3, we plot the utility as we vary the monitor size m from /24 to /19 for different values of s (the exponent value for the Zipf distribution). For smaller values of s, monitors of size /18 or /19 seem to capture most sources and for more heavy tailed distributions (s = 1.3), the utility increases roughly linearly with monitor size.

• **Impact of segmentation.** Next we consider, the impact of segmenting monitors into multiple blocks on the expected length of the game. In Figure 4 (left) we plot the change in shuffle frequency as we increase the number of segments for three different monitor sizes (/18, /20 and /22). The value of n was set to be /16. We fix ℓ to be 100. We find that the length of the game increases linearly with the number of blocks. But the slope of the curve increases as we decrease monitor size (m). Next, in Figure 4 (right), we plot the change in shuffle

[5] These numbers are based on typical daily volumes seen at actual honeynet deployments.

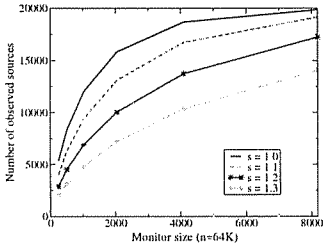


Figure 3: Utility as we expand monitor size (s = scan volume per source exponent).

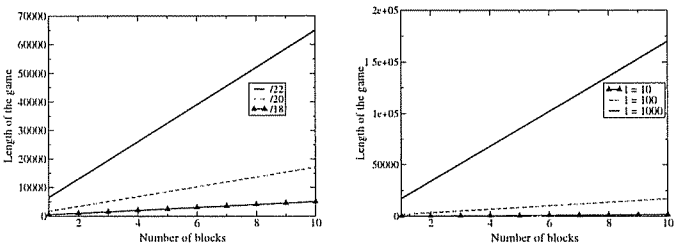


Figure 4: Change in expected shuffle frequency (left) as we vary the monitor size and the number of segments, ℓ=100, n =/16; (right) as we vary ℓ and the number of segments, m =/20, n =/16.

frequency as we increase the number of segments for three different lie budgets. The value of n was set to be /16. We fix m to be /20. Again, we find that the length of the game increases linearly with the number of blocks and the slope of the curve increases as we decrease the lie budget (ℓ).

6.2 Experimental results with Kaleidoscope

We evaluate the performance of our Kaleidoscope system under a number of different traffic configurations to assess feasibility of deployment. The key evaluation metrics are packet loss and delay introduced by the system. Our design goal is for the system to sustain traffic rates upto 200Mbps which is typical for our large campus network. In these experiments we pessimistically set the lie budget ℓ to be 0. [6] Figure 5 provides a sample configuration of the network topology used to evaluate our system. Three key components of this setup include:

1. **Harpoon** – a synthetic traffic generator that generates a wide range of realistic network packet traffic [21]. Our setup includes four Harpoon clients and two Harpoon servers. We configured the client and server IP address pools to be 256K and 128K addresses, with the number of active client and server addresses being 1024 and 512 respectively. We used the default setting for file size distribution *i.e.,* Pareto with (location parameter) x_m = 10 KB and (shape parameter) k = 1.2. The interconnection times were set to be exponentially distributed with a mean value of 0.5 seconds.
2. **NetPath** – a Click based network delay emulator. We configured NetPath to produce a constant delay of 15 ms in each direction between the client and server [1].
3. **Kaleidoscope** – our address shuffling prototype that is implemented as collection of elements in the Click modular router and described in Section 5.

To evaluate the performance of the shuffling system, we use high performance Endace DAG cards [10] to capture packets as they enter and exit the Kaleidoscope system. These cards provide high precision timestamps on all packets. Next, we describe the specific experimental results that illustrate the performance limits of our prototype.

• **Impact of shuffling on performance.** First, we consider the impact of the Kaleidoscope middlebox on network performance. Specifically, we evaluate packet loss, packet delay and connection disruptions that might be introduced by our prototype under load. We configured Harpoon to generate average traffic load from 50-200Mbps. Our Kaleidoscope system generated *zero* packet loss and *zero* connection disruptions. In Figure 6, we report the delays introduced by the system for different shuffling policies. We find that delays are quite minimal, typically under 300μs which is likely to be acceptable in most environments and far less than the response time for some middleboxes [3]. As might be expected, delays introduced by per source shuffling (PSS) are slightly higher than discrete random map shuffling (DRMS), restricted swap shuffling (RSS) or source group shuffling (SGS) since the former policies maintain more state. To measure the maximum performance of the system, we also conducted additional measurements at 400Mbps. In Table 1, we summarize the packet loss introduced by

[6] If our "shuffling" system scales when the lie budget is 0, clearly it will scale for larger values where we shuffle less frequently.

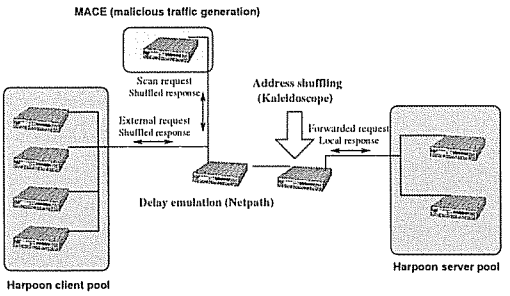


Figure 5: Experimental setup for lab tests with Kaleidoscope.

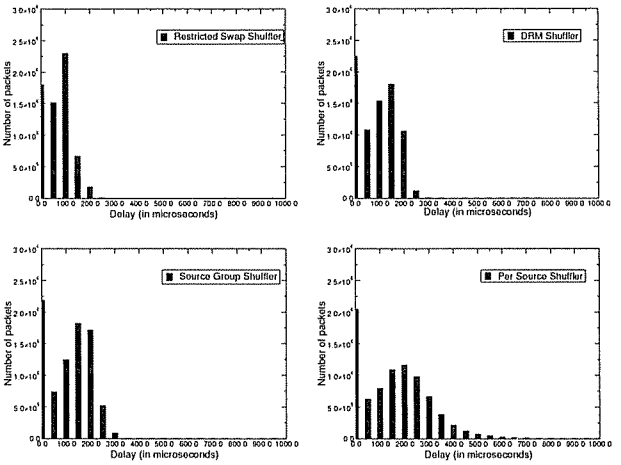


Figure 6: Delay induced by shuffling strategies.

the shuffling middlebox at 200Mbps and 400Mbps. While this rate of traffic introduces a noticeable 0.1% loss rate on the PSS shuffler, the other shufflers continue to be resilient.

• **Robustness to resource attacks.** An important consideration in the design of Kaleidoscope is its resiliency to resource consumption attacks. In particular, there are two classes of attacks that a knowledgable adversary might employ to degrade the quality of service provided by the system. First, there are attacks that overwhelm the system by filling the connection pool (the list of live connections that are tracked by the system). Second, there are attacks that overwhelm the source and destination address pools that are monitored by system. We expect the latter set of attacks to be more of an issue for per-source (PSS) and source-group (SGS) shuffling strategies.

In Figure 7 (left) we provide results of system resiliency (average delay introduced) as we subject Kaleidoscope to connection-pool overload attacks. We use MACE [22] to initiate SYN-flood attacks of increasing severity (from 300-2400 connection attempts per second). Harpoon was configured to generate background traffic at an average rate of 200Mbps. We find that the system is able to handle these attacks without service disruption – no lost packets or stale connections. The HashBelt implementation of the connection pool enables timely eviction of the malicious connections without impact on normal traffic. The average delays across the shuffling strategies remain minimal (under 300μs). As expected DRMS, RSS and SGS outperform PSS.

In Figure 7 (right) we provide results of system resiliency (average delay introduced) as we subject it to address-pool overload attacks. We use MACE to initiate a scanning attacks of increasing severity (from 300 - 2400 connection attempts per second with source addresses and destination addresses picked randomly). Harpoon was configured to generate background traffic at an average rate of 200Mbps. We find that our system is able to handle these attacks without service disruption – zero percent packet loss and no stale connections. The average delays across the shuffling strategies remain minimal (under 1ms). As expected DRMS, RSS and SGS outperform PSS. We expect the impact of the system under other attack scenarios such as worm outbreaks and flash crowds to be similar. However, we leave those tests to future work.

7 Deployment Issues

Our Kaleidoscope experiments demonstrate the feasibility of the use of an address shuffling middlebox for protecting honeynets. However, there are other important design considerations involving the placement and protocol for the shuffling logic.

Network Address Translation (NAT). The address translation logic built into shuffler is analogous to functionality implemented in commodity NAT devices. A key difference is the requirement to support inbound connection requests in an environment where the local (internal) network changes dynamically. Much like a NAT, legitimate server addresses are statically routed, however connection requests to dynamic hosts are arbitrarily routed based on the shuffling epoch. We build our network address shuffler on the premise that most

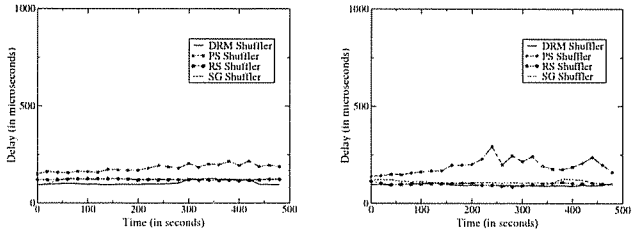


Figure 7: Delays under SYN-flood and scanning attacks.

Shuffler	200Mbps	400mbps
DRMS	0%	0%
RSS	0%	0%
SGS	0%	0%
PSS	0%	0.11%

Table 1: Summary of shuffler-induced packet loss.

addresses are client-based and do not require to be statically routed. We expect our system can be overlayed seamlessly over existing NAT deployments.

Dynamic Host Configuration Protocol (DHCP). The network procotol commonly used to automate the allocation of IP addresses to dynamic participants in a network is DHCP [18]. The transient nature of clients makes such environments particularly hospitable for our shuffling architecture. We expect that our system will readily coexist with existing DHCP networks and we plan to explore opportunities for more active integration of DHCP management with network address shuffling, for *e.g.*, support for adaptive management of address pools.

Integration with routers. One alternative to our middlebox deployment is to use existing intra-AS routing protocols such as OSPF or RIP as the basis for honeynet protection. The advantage of such an approach is the ability to utilize high-performance routing architectures. In such an environment the network address shuffler would send periodic link state updates to route *affected* address blocks through this system, while the remaining traffic would simply bypass the shuffler. Alternatively, one could build the route shuffling request as a new OSPF message type.

8 Summary and Conclusion

In this paper we develop a methodology for safeguarding honeynet monitors from being identified and mapped by malicious entities. Our approach is based on abstracting the honeynet protection problem into a two person game between an Attacker and a Defender. The Attacker's objective is to identify a honeynet embedded within a larger address space with a minimal number of probes, while the Defender's objective is to measure the maximum number of probes before shuffling the honeynet's location within the address space.

We formulate strategies for both the Attacker and Defender that we prove to be optimal. The Attacker's strategy is based on round robin probing to identify a point within the honeynet followed by a binary search to establish the boundaries. The Defender's strategy is to delay revealing the location of points within the honeynet as long as possible. We extend this foundational result to accommodate the possibility of multiple embedded honeynets, which we show will only add a constant factor to the Attacker's probe requirement. These theoretical results form the basis for an analytical investigation of the practical issues in developing a system that can be used to protect honeynets. Our analysis shows that for honeynet configurations commonly found in the Internet, substantial resources may be required of the Attacker in order to quickly identify a honeynet.

We designed and built a prototype middle-box system that we call Kaleidoscope which counts probes to a network and shuffles embedded honeynets when probe thresholds are reached. This system was developed to understand the demands and requirements for four different methods of address space shuffling. We tested Kaleidoscope in a lab environment configured to approximate the traffic conditions in our own campus network where we operate a large honeynet. First, we tested the per packet delays introduced by Kaleidoscope, and found that the worst case delays were on the order of hundreds of microseconds which would be quite acceptable for most networks. Second, we tested the state required within the middle-box for the different shuffling strategies and found that all are easily accommodated by our design. Finally, we tested resource requirements when Kaleidoscope itself is under attack, and found that our implementation approach is sufficient to continue operation without packet loss or significant increase in delay.

9 Appendix

The Main Theorem 3.4.2 states that for any Attacker strategy A other than eRR,

$$\mathbf{E}[L(\mathrm{RR},\mathrm{DD})] < \mathbf{E}[L(A,\mathrm{DD})].$$

The following combinatorial lemma, which we prove below, is the key to the proof of the Main Theorem.

LEMMA **3.4.3**. Let m, k, p and ℓ be positive integers. Let $n = mk$. Let $\mathbf{Z}_n$ be a circular array of n cells. Let $\mathcal{S}$ be a multi-set $\{S^1, S^2, \ldots, S^p\}$, where each $S^i \subset \mathbf{Z}_n$ is a contiguous segment of cells, we call it a block, and each $|S^i| = m$. We say $c \in \mathbf{Z}_n$ is **heavy** with respect to $\mathcal{S}$ if c is covered by at least ℓ blocks $S^i \in \mathcal{S}$. Then, the following upper bound on the number of heavy points holds:

$$|\{c \in \mathbf{Z}_n \mid c \text{ is heavy}\}| \leq m \cdot \lfloor \frac{p}{\ell} \rfloor.$$

Remark: We call $\mathcal{S}$ a configuration. Denote by $C = C(\mathcal{S})$ the number of heavy points covered by $\mathcal{S}$, namely $C(\mathcal{S}) = |\{c \in \mathbf{Z}_n \mid c \text{ is heavy}\}|$. The upper bound $C(\mathcal{S}) \leq \lfloor m \cdot \frac{p}{\ell} \rfloor$ is trivial by a volume argument: the total number of elements in the union $\cup\mathcal{S}$ is at most mp, counting multiplicity. Each heavy point c costs at least ℓ elements to cover. So there could be at most $\lfloor m \cdot \frac{p}{\ell} \rfloor$ heavy points. The trickier part is to prove that the stronger bound in the lemma. This stronger bound is crucial in establishing our Main Theorem 3.4.2; the weaker bound will not be useful.

The bound in the Lemma is trivial if $p \geq k \cdot \ell$. So we assume $p < k \cdot \ell$. Write $p = q \cdot \ell + r$, where $q = \lfloor \frac{p}{\ell} \rfloor < k$ and $0 \leq r < \ell$.

Consider RR (or eRR). Each query at j defines a block $B_j = \{j, j+1, \ldots, j+m-1\}$ which is a contiguous segment of m cells, starting at j. It is obvious that j hits B^c iff c belongs to the block B_j, *i.e.* $j \in B^c$ iff $c \in B_j$. Now it is clear that the upper bound in the Lemma is achieved by RR (and eRR), since each batch of ℓ repeated queries defines a block, each of which is repeated ℓ times, and these blocks are spaced exactly m cells apart from one batch to the next. Thus, the first $q \cdot \ell$ queries define q distinct and disjoint blocks each repeated ℓ times, and together they cover $m \cdot q$ *heavy* points. The point of the lemma is that one can not possibly use the left-over 'capacity' of rm to cover more *heavy* points. Note that if one is to use the first $q \cdot \ell$ queries to cover $m \cdot q$ heavy points (as we did above with RR), *then* it is certainly impossible to use the left-over 'capacity' of rm to cover more heavy points. The Lemma says much more. It asserts that, *no matter how* one does it, it can never cover more heavy points.

We note that if $p \equiv 0 \pmod{m}$, then the lemma is trivial. However, this is not sufficient for the proof of the Main Theorem. Now we prove the lemma.

Proof. Fix p. Let $\mathcal{S}$ be a configuration of p blocks which maximizes the number of heavy points, denoted by $C(\mathcal{S})$. The proof of the lemma consists of two major steps. In the first step we will show that we may always modify the configuration $\mathcal{S}$ to another $\mathcal{S}'$ with the same number of blocks, such that $C(\mathcal{S}) \leq C(\mathcal{S}')$, and in $\mathcal{S}'$ no point c is covered by more than ℓ blocks. The second step of the proof shows that for such configurations $\mathcal{S}'$, $C(\mathcal{S}') \leq m \cdot \lfloor \frac{p}{\ell} \rfloor$.

Step 1: Given $\mathcal{S}$. Define a $k \times m$ matrix N as follows: The entries of N will be indexed by (i, j), where $0 \leq i < k$ and $0 \leq j < m$, and $N_{i,j}$ is the number of blocks in $\mathcal{S}$ covering the cell $c = im + j$. For later purposes we will also denote $N_c = N_{im+j} = N_{i,j}$. Note that each cell $c \in \mathbf{Z}_n$ has a unique expression $c = im + j$, where $0 \leq i < k$ and $0 \leq j < m$. On each column j of N, the blocks counted in $N_{i,j}$, for $0 \leq i < k$, all come from distinct blocks in $\mathcal{S}$. This is because each block has length m, and therefore can not appear in more than one count in a column.

Also each block in $\mathcal{S}$ contributes a total of m to the sum $\sum_{i,j} N_{i,j}$ so that

$$\sum_{i,j} N_{i,j} = mp.$$

In fact on each column, each block contributes exactly 1, therefore for every $0 \le j < m$,

$$\sum_{i=0}^{k-1} N_{i,j} = p.$$

In particular, since $p < k\ell$, there must be at least one $N_{i,j} < \ell$ for every column $0 \le j < m$.

If all entries $N_{i,j} \le \ell$, then **Step 1** is done. Assume otherwise. We now perform what we call a sequence of '**wipe-the-cream**' operations: Start at any $c_0 = i_0 m + j_0$ with $N_{i_0,j_0} \le \ell$. Cyclically increase c_0 by 1, until we find the first $c = im + j$ such that $N_{i,j} > \ell$. If $c' = c - 1 = i'm + j'$ is the previous cell, then it is clear that $N_{i,j} > \ell \ge N_{i',j'}$, and that the increase must be due to some blocks starting at the location c. Let f be the number of blocks starting at the location c, then $f \ge N_{i,j} - \ell$. Let $f' = N_{i,j} - \ell$, then $f \ge f' > 0$. Now *move* f' of the f starting blocks one cell to the next, to start at $c+1$ instead. Note that for the new configuration the new value at c is $\tilde{N}_{i,j} = \ell$.

We then start at c and repeat this. This step of 'wipe-the-cream' operation is continued until there is no more $N_{i,j} > \ell$.

Consider the effect of one step of this 'wipe-the-cream' operation on the matrix N. There are exactly two entries of this matrix N that are changed by this one step: the entry $N_{i,j}$, which is changed from $> \ell$ to $= \ell$, and the entry $N_{i+1,j}$, which is increased by f'. Hence the number of heavy points C is *not* decreased. Moreover, if we perform the above sequence of 'wipe-the-cream' operations in row major order, but we focus on its effect on each column of the matrix exclusively, the entire cyclic sequence of 'wipe-the-cream' operations can be viewed *as if it were to be performed* on each column individually, in a parallel fashion. (At this point, the reader probably can see why we call it a 'wipe-the-cream' operation.) One can think of the effect on any column j as a virtual 'wipe-the-cream' operation which merely shifts the numbers $N_{i,j}$ in excess of ℓ to the next entry on the same column, as described above: If $N_{i,j} > \ell$, then the new $\tilde{N}_{i,j} = \ell$ and the new $\tilde{N}_{i+1,j} = N_{i+1,j} + f' = N_{i+1,j} + (N_{i,j} - \ell)$.

We note that when we perform one step of this 'wipe-the-cream' operation at c, the sum of two consecutive terms on a column at c and $c+m$ is unchanged. If we actually shifted some blocks for r consecutive terms in *one* column, this creates a plateau of r consecutive terms with the respective $N_{i,j} = \ell$. These terms effectively no longer participate in any future 'wipe-the-cream' operations; in any future rounds they simply pass any left-over 'cream' (if there is any) from the previous cell to the next cell on the same column. If we focus on an individual column, and consider the virtual 'wipe-the-cream' operation, this effectively reduces the length of the column on which we perform this virtual 'wipe-the-cream' operation. If we disregard the cells already having $N_{i,j} = \ell$ (and thus removed for the virtual 'wipe-the-cream' operation), every actual step which shifts a positive amount from one cell to the next strictly reduces the length of the column. However each column sum is $p < k\ell$ and is preserved throughout, some entry $< \ell$ on each column must remain at all time. Therefore, this virtual length of the column can not be reduced to 0. This proves that the sequence of 'wipe-the-cream' operation must terminate on each column, and therefore must terminate overall. **Step 1** is proved.

Step 2: Assume $\mathcal{S}$ is a configuration with all $N_c \le \ell$, and achieves maximum count $C(\mathcal{S})$ of heavy points.

We have already remarked that we may assume $p < k\ell$ and $p \not\equiv 0 \pmod{\ell}$, for otherwise the Lemma is easy to prove. Note that these two conditions imply $q < k$ and $1 \le r < \ell$ in $p = q\ell + r$. In particular, we may assume $\ell > 1$. We may consider the Lemma already proved inductively for any smaller values of ℓ. The base case $\ell = 1$ is trivially true. We may also consider the Lemma already proved inductively for any smaller values of p; again the base case is trivial.

Now we define two notions: a **gap** and a **train**.

A gap G is a contiguous set of points $\{a, a+1, \ldots, b-1\} \subseteq \mathbf{Z}_n$, where $a \ne b$ (thought of as $a < b < a+n$), such that $N_{a-1} = N_b = \ell$ and $N_a, N_{a+1}, \ldots, N_{b-1}$ are all $< \ell$. We say the length of the gap is $b - a$. (Strictly speaking we take the positive modulus of $b - a \pmod{n}$ in $\{1, \ldots, n-1\}$.)

A (w,t)-train T is a collection of $w \cdot t$ many blocks, such that there exists a $c \in \mathbf{Z}_n$, for all $0 \le i \le t-1$, there are exactly w blocks in T which start at the cell $c + im$. Thus, a (w,t)-train T covers tm contiguous cells starting at c, each with multiplicity w. We say T has width w and length tm.

For any $w > 0$, a (w,k)-train T is called a complete train. A complete train wraps around the whole circular array $\mathbf{Z}_n$. If we have a complete train, we can remove all the blocks in a $(1,k)$-complete train, and obtain a configuration S^* which covers the same number of heavy points for the parameters $\ell' = \ell - 1$ and $p' = p - k$. *i.e.*, $C(S) = C(S^*)$. By inductive hypothesis, this $C(S^*) \leq \lfloor p'/\ell' \rfloor \cdot m$. But by $k > q$,

$$p' = q\ell + r - k \leq q\ell + r - (q+1) = q(\ell - 1) + (r-1),$$

where $0 \leq r - 1 < \ell - 1 = \ell'$. Thus $\lfloor p'/\ell' \rfloor \leq q$. It follows that

$$C(S) \leq q \cdot m.$$

Thus, any complete train finishes the proof.

The strategy in **Step 2** will be the following. We will consider a *gap with minimum length*, and define an appropriate *train* and a *movement* of the train. Assume there are at least two gaps in S. The movement of the train will not change $C(S)$, and will either (a) reduce the number of gaps in S, or (b) reduce the length of the minimum length gap without increasing the total number of gaps in S, or (c) move the minimum length gap one cell closer to another gap while not affecting any gap other than the minimum length gap. In case (c), if repeated, a sequence of such train movements will eventually merge the minimum length gap with a nearby gap and thus reducing the total number of gaps. It is clear by $p < k\ell$ that there must be at least one gap. The above sequence of train movements will eventually reduce the situation to a configuration with only one gap, which will be directly handled.

Now we carry out this plan. Let $G = \{a, a+1, \ldots, b-1\}$ be a gap with minimum length. If s is the number of blocks starting at b and e is the number of blocks ending at $b-1$, then $s - e = N_b - N_{b-1} > 0$. Let $w = s - e$. We will define a (w,t)-train for some t. As $w \leq s$ there are at least w blocks starting at b. Choose any such w starting blocks. Consider cell $b+m-1$. If $N_{b+m-1} < \ell$, then we may move these w blocks one cell to the left, whereby producing a new configuration with $\tilde{N}_{b-1} = \ell$, $\tilde{N}_{b+m-1} = N_{b+m-1} - w$, and with no other changes to N. As $b+m-1$ was already not a heavy point, this movement would have increased $C(S)$ by 1. Since S was assumed to be optimal, this is impossible. Therefore we may assume $N_{b+m-1} = \ell$.

Next, consider cell $b+m$. Either $N_{b+m} < \ell$, or $N_{b+m} = \ell$. If $N_{b+m} = \ell$, then since there are at least w blocks ending at $b+m-1$, there must be at least that many blocks starting at $b+m$. We pick any w of these starting blocks and together with the w blocks starting at b form a $(w,2)$-train. On the other hand, if $N_{b+m} < \ell$, then $b+m$ must be the starting cell of a gap G'.

In the case with $N_{b+m} = \ell$, we will, by the same argument, continue at $b+2m$, $b+3m, \ldots$, and keep 'hooking up' the train until we have defined a (w,t)-train which borders on a gap G' which starts at the next cell $b+tm$. Note that if the (w,t)-train is extended all the way by wrapping around to $b-1$, we would have a complete (w,k)-train. As noted earlier, this completes the proof of the Lemma. Thus we assume $t < k$.

Now there are two cases: Either $G' = G$ or $G' \neq G$. If $G' \neq G$, then by moving the (w,t)-train one cell to its left, we change N_{b-1} from $< \ell$ to $\tilde{N}_{b-1} = \ell$ and N_{b+tm-1} from $= \ell$ to $\tilde{N}_{b+tm-1} < \ell$. No other changes occur to N. This keeps $C(S)$ unchanged, thus still optimum, while removing one cell out of the gap G and adding one cell to the gap G', which was to start at $b+tm$. No changes occur to any other gap. Of course it is possible that the gap G disappears (if its length was 1), or at least its length is reduced by 1. The gap G' may stay as a single gap, with size increased by 1, or it may be merged with another gap to its left. In any case, either the number of gaps is reduced or the number of gaps stays the same with the length of the minimum gap reduced (or both).

Let's consider the case $G' = G$. This means the (w,t)-train wraps around all the way to end in $a-1$; in particular $a \equiv b \pmod{m}$. If we also move the (w,t)-train one cell to its left, we would have effectively moved the cell locations of the minimum length gap G one cell to its left. No changes occur to any other gap. This may of course merge G with a gap to its left, or it simply moves the locations of G one cell to its left without changing the status of any other gap. In the latter case, both the total number of gaps and the minimum length of a gap are unchanged. We can then proceed to define another (w',t')-train starting at $b-1$ and repeat the above process. (Note that it is possible that $w' \neq w$ and $t' \neq t$.)

We have proved that the goals (a), (b) or (c) are accomplishable as stated earlier. It follows that eventually we can arrive at a configuration which has the same p, still has maximum $C(\mathcal{S})$, and has *only one* gap.

If we start at this unique gap G and repeat the above definition of a (w,t)-train, it must end in the same gap $G' = G$. Thus $a \equiv b \pmod{m}$. Thus, the gap G has a length a positive multiple of m.

If all $N_a = N_{a+1} = \ldots = N_{b-1} = 0$ within the gap G, then $p \equiv 0 \pmod{\ell}$, and we are done. Now suppose some $N_i \neq 0$, where $a \leq i < b$, and let B be a block which contributes to $N_i > 0$. If B is completely contained in the gap G, then we may drop this B and obtain a configuration $\mathcal{S}'$, which has $p' = p - 1$ and covers the same number of heavy points as $\mathcal{S}$. Then, by induction,

$$C(\mathcal{S}) = C(\mathcal{S}') \leq \lfloor p'/\ell \rfloor \cdot m \leq \lfloor p/\ell \rfloor \cdot m.$$

Suppose now every block contributing to some $N_i > 0$ in the gap is not completely contained in G. Thus B straddles the end points. Without loss of generality suppose $b \in B$. Let $B = B_c$, *i.e.*, it starts at c, and thus $0 < b - c < m$. By the same argument earlier we can define a $(1,t)$-train starting with the block B. The ending cell of this $(1,t)$-train is $c + tm - 1$. We claim it must be to the right of $a - 1$. This is because, if it were to end at a cell to the left of $a - 1$, then we could have extended the train further; it could not end at $a - 1$ either, because both the train length and the gap length are $\equiv 0 \pmod{m}$, and $0 < b - c < m$. In fact this $(1,t)$-train must end exactly $m - (b - c)$ cells to the right of $a - 1$. (It could not extend any further, or else there would have been a block completely contained in G.) Now if we move this $(1,t)$-train $b - c$ cells to the right, we will have produced a single block, namely the last block of this $(1,t)$-train, completely contained in G, and maintain the same $C(\mathcal{S})$.

This completes our proof of the Lemma.

□

References

[1] S. Agarwal, J. Sommers, and P. Barford. Scalable Network Path Emulation. In *Proceedings of IEEE MASCOTS*, 2005.

[2] W32 Agobot IB. http://www.sophos.com/virusinfo/analyses/trojagobotib.html.

[3] M. Allman. On the Performance of Middleboxes. In *Proceedings of Sigcomm Internet Measurement Conference*, 2003.

[4] S. Antonatos, P. Akritidis, E. Markatos, and K. Anagnostakis. Defending against hitlist worms using network address space randomization. In *Proceedings of ACM CCS WORM '05*, Fairfax, VA, November 2005.

[5] M. Atighetchi, P. Pal, F. Webber, R. Schantz, and C. Jones. Adaptive use of network-centric mechanisms in cyber defense. In *Proceedings of the 6th International Symposium on Object-oriented Real-time Distributed Computing*, May 2003.

[6] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson. The internet motion sensor: A distributed blackhole monitoring system. In *Proceedings of the 12th Network and Distributed Systems Security Symposium (NDSS '05)*, San Diego, CA, February 2005.

[7] J. Bethencourt, J. Franklin, and M. Vernon. Mapping Internet Sensors with Probe Response Packets. In *Proceedings of USENIX Security Symposium*, 2005.

[8] E. Cooke, M. Bailey, M. Mao, D. Watson, F. Jahanian, and D. McPherson. Toward understanding distributed blackhole placement. In *Proceedings of CCS Workshop on Rapid Malcode (WORM '04)*, October 2004.

[9] D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levine, and H. Owen. Honeystat: Local worm detection using honeypots. In *Proceedings of Symposium on Recent Advances in Intrusion Detection (RAID)*, Sophia Antipolis, France, September 2004.

[10] Endace: Accelerated Network Security. http://www.endace.com.

[11] German Honeynet Project. Tracking Botnets. http://www.honeynet.org/papers/bots, 2005.

[12] X. Jiang and D. Xu. A vm-based architecture for network attack detention center. In *Proceedings of the USENIX Security Symposium*, San Diego, CA, August 2004.

[13] D. Kewley, J. Lowry, R. Fink, and M. Dean. Dynamic approaches to thwart adversary intelligence gathering. In *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX)*, 2001.

[14] E. Kohler, R. Morris, B. Chen, J. Jannotti, and F. Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, August 2000.

[15] J. Michalski, C. Price, E. Stanton, E. Chua, K. Seah, W. Heng, and T. Pheng. Final report for the network security mechanisms utilizing network address translation ldrd project. Technical Report SAND2002-3613, Sandia National Laboratories, November 2002.

[16] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of Internet Background Radiation. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference*, 2004.

[17] N. Provos. A virtual honeypot framework. In *Proceedings of USENIX Security Symposium*, 2004.

[18] RFC 2131 - Dynamic Host Control Protocol. http://www.faqs.org/rfcs/rfc2131.html.

[19] S. Savage. Opportunistic measurement. In *Proceedings of the Fourth Workshop on Hot Topics in Networks (HotNets IV)*, College Park, MD, November 2005.

[20] Y. Shinoda, K. Ikai, and M. Itoh. Vulnerabilities of Passive Internet Threat Monitors. In *Proceedings of USENIX Security Symposium*, 2005.

[21] J. Sommers and P. Barford. Self-Configuring Network Traffic Generation. In *Proceedings of ACM SIGCOMM Internet Measurement Conference*, 2004.

[22] J. Sommers, V. Yegneswaran, and P. Barford. A Framework for Malicious Workload Generation. In *Proceedings of ACM SIGCOMM Internet Measurement Conference*, 2004.

[23] S. Staniford, V. Paxson, and N. Weaver. How to 0wn the Internet in Your Spare Time. In *Proceedings of the 11th USENIX Security Symposium*, 2002.

[24] J. Ullrich. Dshield. http://www.dshield.org, 2005.

[25] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. Snoeren, G. Voelker, and S. Savage. Scalability, fidelity and containment in the potemkin virtual honeyfarm. In *Proceedings of ACM SOSP '05*, Brighton, UK, October 2005.

[26] V. Yegneswaran, P. Barford, and V. Paxson. Using Honeynets for Internet Situational Awareness. In *Proc. ACM Hotnets*, 2005.

[27] V. Yegneswaran, P. Barford, and D. Plonka. On the design and use of Internet sinks for network abuse monitoring. In *Proc. RAID*, 2004.

[28] V. Yegneswaran, P. Barford, and J. Ullrich. Internet intrusions: Global characteristics and prevalence. In *Proc. ACM SIGMETRICS*, 2003.