Camouflaging Honeynets

Vinod Yegneswaran
Computer Science Laboratory,
SRI International,
333 Ravenswood Ave,
Menlo Park, CA 94025.
vinod@csl.sri.com

Chris Alfeld
Computer Sciences Dept.,
University of Wisconsin,
1210 W. Dayton St,
Madison, WI 53706.
alfeld@cs.wisc.edu

Paul Barford
Computer Sciences Dept.,
University of Wisconsin,
1210 W. Dayton St,
Madison, WI 53706.
pb@cs.wisc.edu

Jin-Yi Cai
Computer Sciences Dept.,
University of Wisconsin,
1210 W. Dayton St,
Madison, WI 53706.
jyc@cs.wisc.edu

***Abstract*—Over the past several years, honeynets have proven invaluable 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, so we can ensure that honeynets remain viable in the long term. Our approach is based on two ideas: (*i*) counting the number of probes received in the honeynet, and (*ii*) shuffling the location of live systems with those that comprise the honeynet in a larger address space after the probe count has exceeded a threshold. We describe four different strategies for randomizing the location of the honeynet. Each strategy is defined in terms of the degree of defense that it provides and its associated computational and state requirements. We implement a prototype middlebox that we call Kaleidoscope to gain practical insight on the feasibility of these strategies. Through a series of tests we show that the system is capable of effectively defending honeynets in large networks with limited impact on normal traffic, and that it continues to respond well in the face of large resource attacks.**

I. Introduction

One of the most basic components of malicious activity in the Internet is the process of scanning the address space for potentially vulnerable hosts. A fundamental challenge in this activity is determining how to select a target address. While many methods have been proposed, the most common approach is by brute force, *i.e.,* simply sending scans to *all* addresses in a given network segment. While this approach can be quite effective, it also offers the network security community a significant opportunity for gathering detailed information on attacks by deploying measurement systems on the routed but otherwise unused IP addresses in a network segment. These measurement systems are typically referred to as a *honeynet*.

The value of honeynets to security analysts is certain to be well known to their adversaries. In fact, a series of recent studies has shown that the locations of monitoring systems used to gather data for well-known reporting services such as Dshield.org [28] can be pinpointed using active probing methods [6], [24], [21]. One must assume that savvy adversaries have either adopted these methods already or will do so in the near future. If so, then they will be able to create blacklists, which they can incorporate into their tool, and thereby avoid being tracked by honeynets. Obviously, this would have a serious, detrimental impact on the network security community.

In this paper we address the problem of protecting honeynets from adversaries who seek to systematically identify the addresses that are monitored. We propose an approach that is based on shuffling monitored addresses with addresses used by live (and potentially vulnerable) systems so that an adversary is unable to be certain of the honeynet's location. In this case, the problem boils down to two important questions: *when* to shuffle? and *how* to shuffle? In the case of the former, we make a simplifying if conservative assumption for this paper, which is to shuffle as soon as we count one probe sent to each of the N addresses monitored by the honeynet. We discuss more realistic probing strategies in [7]. The issue of *how* to shuffle actually has two components: the method for randomizing the honeynet's location, and the network mechanism that will be used to facilitate the shuffling.

We describe a series of techniques for randomizing the honeynet's location within a large address space. While random shuffling at the granularity of individual IP addresses is optimal in terms of hiding the honeynet, it may not be practical from an implementation perspective. Therefore, 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 by the shuffling mechanism.

Our deployment strategy is based on a middlebox that sits in front of the live hosts and the honeynet. To investigate the feasibility of this approach, we implemented a prototype system that we call Kaleidoscope. The basic functionality of Kaleidoscope includes counting probes to the honeynet, shuffling addresses using one of the four proposed methods, and providing network address translation between the systems outside of the target network and both live and honeynet systems within. We use this system in a series of laboratory-based experiments with different traffic load profiles. Our prototype system demonstrates that the shuffling policies can be implemented efficiently, with minimal latency (under a millisecond) and zero packet loss, under traffic loads typical for a large campus and under different types of malicious attacks. While quantifying the level of protection afforded by this system is beyond the scope of this work, we argue that the effective level of protection provided by this system is sufficient for all but the most determined adversary.

II. Background and Related Work

Over the past several years, a growing number of honeynets have been deployed in the Internet. These range from simple passive monitors to sophisticated systems that emulate responses for standard protocols (*e.g.,* [30]) to large deployments of high-fidelity virtualized systems (*e.g.,* [8]). Furthermore, research on honeynet systems is active and focused on enabling scalable and secure measurement at greater levels of detail. The utility of the honeynets that *respond* to connections is that they reveal the detailed features of attacks. This information is invaluable in both studies that characterize attack activity (*e.g.,* [20], [9], [12]), and in the creation of signatures that can be used to defend against this activity (*e.g.,* [18], [31]). In [25], a methodology is proposed for automatically generating representative honeynet configurations.

To the best of our knowledge, the problem of defending honeynets from systematic mapping has not been addressed in prior studies. The possibility of devising methods for camouflaging honeypots is outlined briefly at the end of [29]. The related problem of limiting the utility of hitlists of live systems has been treated in several studies. Antonatos *et al.* propose *address space randomization* as a method for defending live systems from worms that use hitlists [3], [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 methods for changing network configurations in order to attempt to thwart malicious activity have been proposed in [5], [14], [19], [15]. We differ from these efforts by focusing on protecting honeynets and demonstrate a simple, scalable, and extensible system for address space randomization that implements several shuffling policies.

Our method for defending honeynets is based on the use of a middlebox [23]. Typical instances of these include firewalls or network address translation systems. We are informed by Allman's work on analyzing the performance of middleboxes in [2]. Most specific to our work are the studies of network address translation methods and systems such as [16]. In that work, Kohler *et al.* describe an efficient architecture for NAT functions that forms the basis of our prototype implementation.

III. System Design and Implementation

Our threat model is an adversary who desires to identify a set of N IP addresses that comprise a honeynet in a network segment of size M that also contains a set of O IP addresses that terminate with live hosts where $N+O=M$ (although this equality is not a strict requirement). We also assume that the adversary has the ability to determine that a given IP address is part of a honeynet by sending P probes to that address (we make the conservative assumption that $P=1$ for this study), and that *all* the IP addresses that comprise the honeynet must be probed in order to complete a mapping. Our starting point for defense against this threat is that the set of addresses N must be periodically shuffled (the time between shuffles is a *shuffling epoch*) within M so that any complete map generated by the adversary will quickly become incorrect. We assume that shuffling will be done by a system that resides at the network perimeter, and is capable of network address translation and probe counting. Obviously, there are many possible specifications for this framework, but we argue that this formulation is reasonable and useful for gaining insights on the problem — in particular in terms of the feasibility from an implementation perspective.

A. Shuffling Strategies

In the four strategies described below, we assume that a honeynet monitors B blocks of IP addresses that can vary in size and that are not necessarily contiguous or disjoint. Likewise, the address blocks assigned to live systems can vary in size, and are not necessarily contiguous but will always be disjoint. Key considerations for each strategy are state maintenance and providing an acceptable level of defense against mapping. We do not attempt to make definitive claims with respect to the latter in this paper. However, we argue that a system that can quickly generate a new randomization (those below are generated on the order of microseconds) after a conservative number of probes are counted in the honeynet will provide effective protection against most adversaries.

• *Uniform Block Shuffling (UBS).* In this strategy, the address space M is divided into uniformly sized blocks. When a shuffling epoch is triggered, the algorithm generates a random permutation to remap all blocks and then randomly shifts each segment by a small offset (less than the length of a block). In this case, live network blocks could be remapped to other live blocks, which requires continuous connection state across the entire address space to be maintained. A perfectly random IP shuffling strategy could be achieved by simply setting the block size to one. However, the overhead of such a design choice would be significant for large networks, and a given block of the honeynet is not guaranteed to move from one epoch to the next.

• *Non-Uniform Block Shifting (NBS).* In this strategy, shuffling is restricted to a subset of the total network address space, $A \subset M$. One or more honeynet blocks (of possibly varying sizes) are specified during configuration and these block sizes remain fixed during operation. When a shuffling epoch is triggered, the algorithm randomizes the starting address for each honeynet block in A. Our implementation of this strategy allows two honeynet blocks to overlap. The advantage is that at any point in time only a portion of the address space may be affected. Thus, connection state for live systems affected in the current epoch and adjacent epochs would need to be maintained. However, some honeynet addresses may not change between epochs and the overall size of the honeynet is not guaranteed to remain N.

• *Per Source Shuffling (PSS).* This strategy is similar to NBS 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 shuffling takes place only when a single source has reached the probing threshold. At that point, the network is shuffled only for that external source. The concern for this strategy is the state required to maintain the network maps for each external host. However, this potentially enables the system to shuffle less frequently, and thus may require fewer resources for connection entries. Perhaps more significantly, connections from benign hosts (hosts that target only one or a small number of internal hosts) would never be shuffled. This might be a feasible design choice for networks with a small number of honeynet blocks. An alternative formulation of *PSS* could include blocking all probes from sources after they reach a threshold.

• *Source Group Shuffling (SGS).* This strategy attempts to balance the benefits of *NBS* and *PSS* by maintaining network maps for *groups* of sources. In our prototype implementation, sources are grouped by their target port numbers, with the intuition that sources, which are coordinated (distributed, but controlled by a single agent), would attempt to identify honeynet addresses using a consistent probing method on a small set of services. Thus, when the scanning threshold is met by a group of sources, their view of the network will be shuffled. This approach is likely to be more resilient to distributed network mapping and provides some of the benefits of *PSS*.

B. The Kaleidoscope Prototype

We developed a prototype shuffling middlebox as a collection of Click routing elements [17], shown in Figure 1. Click provides a C++ software framework for packet processing, with impressive scaling performance and a flexible configuration language, that makes it ideal for building software routers. Our enhancements to Click involved the development of two new classes of network elements.

- **Address Modification Element.** This element (*NetRandomizer*) (*i*) handles packet input (including probe counting), (*ii*) invokes shuffling elements to obtain address mappings, and (*iii*) rewrites addresses in the IP header.
- **Shuffling Elements.** These elements (*UBS, NBS, PSS, SGS)* 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 to swap between shuffling strategies. We implemented information elements for each of the shuffling policies discussed above. The design consideration for shuffling strategies involves a trade-off between the amount of state maintained and frequency of shuffling.

Our system maintains two types of state: connection pool (*i.e.,* live connections) and address mapping tables.

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 four operations `find`, `insert`, `delete` and `rotate`. During `find` operations, connection entries

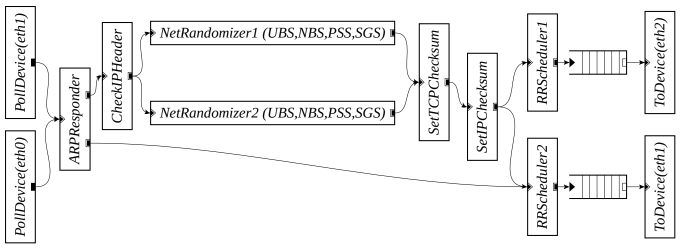

Fig. 1. Kaleidoscope Click element configuration.

are automatically moved at the head of the HashBelt. On `rotate` operations, a new hash table is inserted to the head of the HashBelt deleting 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. Every connection is associated with the following tuple of mapping information: <*external-ip*, *external-port*, *permuted-ip*, *local-ip*, *local-port*>, where the *permuted-ip* is simply the internal address as seen by the external host, and the *local-port* is identical to the port in the permuted flow. Hence, the combination of <*external-ip, external-port*, *local-port*> forms a unique identifier for each tracked connection.

Address Maps. Address maps keep track of honeynet and live network blocks and translation between internal and permuted IP address. For per-source shuffling, we reuse 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 honeynet block, except for the case of *UBS*, which incurs an entry per address block.

Strategies that maintain global state (*NBS* and *UBS*) 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 might require smaller connection pools.

Cost/Benefit Analysis. The variables affecting the performance of network shufflers include the packet rate, number of sources, number of ports being scanned, and the fraction of scanning sources observed by the honeynet. Let μ be the average probe rate per honeynet IP address and c be the arrival rate of legitimate connections per IP address. Let s and g be the average number of sources and source-groups respectively tracked by the shuffler at any given time. Further, let F and G be the fraction of scanning sources (those that scan the entire honeynet) and the fraction of packets sent to scanned ports respectively. We can then summarize the cost/benefit of the four strategies, as shown in Table I. We consider the resiliency of UBS to be low as a knowledgeable attacker could employ block sampling to quickly identify honeynet blocks.

Shuffler	E(epoch duration)	State maintenance overhead		Resiliency	Consistency
		Address Map	*Connection Pool*		
UBS	PM/μ	N/B	$\frac{cO+\mu M}{PM/\mu}$	low	high
NBS	PM/μ	$2M/B$	$\frac{cM+\mu M}{PM/\mu}$	high	high
PSS	$\frac{PM}{F\mu}$	$2Ms/B$	$\frac{(F\mu)*(cM+\mu M)}{PM}$	medium	medium
SGS	$\frac{PM}{G\mu}$	$2Mg/B$	$\frac{(G\mu)*(cM+\mu M)}{PM}$	high	medium

TABLE I
COST/BENEFIT SUMMARY OF THE FOUR SHUFFLING STRATEGIES.

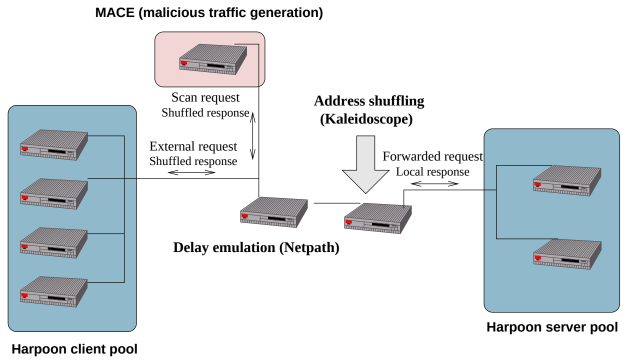

Fig. 2. Experimental setup for lab tests with Kaleidoscope.

NBS has high resiliency and high intra-epoch consistency, but may be subject to frequent shuffling. The resiliency of PSS is medium as it is vulnerable to attacks from coordinated sources; moreover, the network appears inconsistent across sources. Finally, SGS appears inconsistent across source-groups, but might be subject to least frequent shuffling.

IV. EXPERIMENTAL EVALUATION

To assess the feasibility of deployment, we evaluated Kaleidoscope under a number of different traffic scenarios in a laboratory setting. The key evaluation metrics are packet loss and delay introduced by the system. Our design goal is to sustain traffic rates as high as 200 Mbps, which is typical for our large campus network. Figure 2 shows the network topology used to evaluate our system. Key components of this setup include

1) **Harpoon** — a synthetic packet traffic generator [26]. 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 servers being 1024 and 512 respectively. We used the default heavy-tailed file size distribution, and set the interconnection times to be exponentially distributed with a mean value of 0.5 s.
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) **MACE** – python-based malicious traffic generator [27].
4) **Kaleidoscope** — our address shuffling prototype, that is implemented as a collection of elements in the Click modular router, and described in Section III.

Synchronized, high-performance DAG cards [11] were used to capture packets as they enter and exit the Kaleidoscope system. These cards provide high-precision timestamps on all packets over the course of an experiment, which enabled us to establish a detailed performance profile for Kaleidoscope. The honeynet was configured to be two /18 address blocks (32K addresses) within a larger address space of two /16 address blocks (128K addresses). Any Harpoon connection to the honeynet blocks was considered a probe. Each experiment was run over a period of 15 minutes. Shuffling epochs varied in each test but averaged several seconds in length depending on the shuffling strategy, traffic load, and randomness of Harpoon.

• **Impact of Shuffling on Performance.** We evaluated system performance in terms of packet loss, packet delay and connection disruptions that were introduced by Kaleidoscope. We configured Harpoon to generate average traffic loads from 50 to 200 Mbps. Kaleidoscope caused *zero* packets to be lost and *zero* connection disruptions. In Figure 3, the distribution of delays introduced by the system are reported for different shuffling strategies. It can be seen that delays are quite minimal, typically under 300 μs, which is likely to be acceptable in most environments and far less than the response time for some middleboxes [2]. As might be expected, delay introduced by *PSS* is slightly higher than *UBS*, *NBS*, or *SGS* since the former policies maintain more state. To examine performance of the system under heavy load, we conducted additional measurements at 400 Mbps. Table II summarizes the packet loss introduced by the shuffling middlebox at 200 Mbps and 400 Mbps. 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 knowledgeable 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 the system. We expect the latter to be a more significant issue for *PSS* and *SGS* strategies.

Figure 4 (left) shows Kaleidoscope's resiliency to a connection pool overload attack in terms of average delay. We used MACE [27] to initiate SYN-flood attacks of increasing severity (from 300 to 2400 connection attempts per second). Harpoon

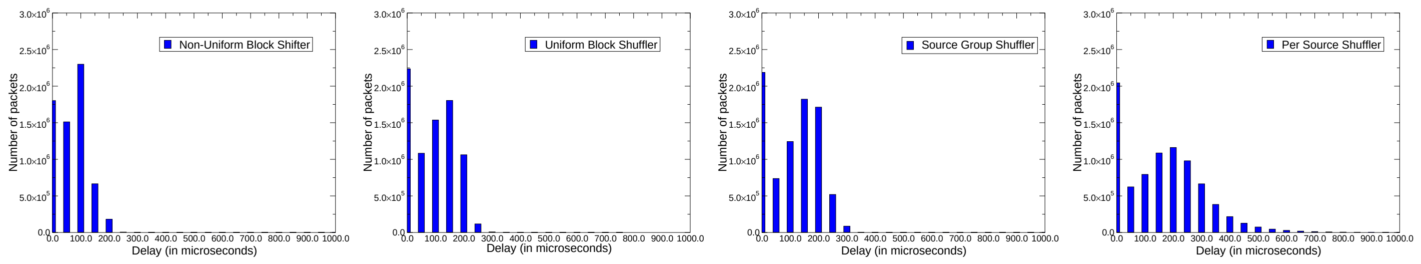

Fig. 3. Packet delay distribution for Kaleidoscope under different shuffling strategies.

was configured to generate traffic at our target average rate of 200 Mbps. The results indicate 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 300 μs). As expected *UBS*, *NBS*, and *SGS* outperform *PSS*.

Figure 4 (right) shows Kaleidoscope's resiliency to an address pool overload attack in terms of average delay. We used MACE to initiate scanning attacks of increasing severity (from 300 to 2400 connection attempts per second with source addresses and destination addresses picked randomly). Harpoon was configured to generate traffic at our target average rate of 200 Mbps. The results indicate that the system is able to handle these attacks without service disruption — zero percent packet loss and no stale connections. The average delay across the shuffling strategies also remains minimal (under 1 ms). As expected, *UBS*, *NBS*, 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. Those and *in situ* tests are future work.

V. Discussion

Initial experiments with Kaleidoscope demonstrate the feasibility of a shuffling middlebox for protecting honeynets. However, there are other important considerations involving the placement and protocol for the shuffling logic.

A. Deployment Issues

Network Address Translation (NAT). The address translation logic built into the 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 routed based on the shuffling epoch. We built our network address shuffler on the premise that most addresses are client hosts that do not require static routing.

Dynamic Host Configuration Protocol (DHCP). The network protocol commonly used to automate the allocation of IP addresses to dynamic participants in a network is

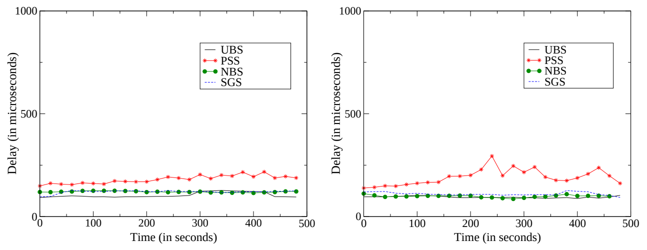

Fig. 4. Packet delays caused by Kaleidoscope under SYN-flood (left) and scanning attacks (right).

Shuffler	200 Mbps	400 Mbps
UBS	0%	0%
NBS	0%	0%
SGS	0%	0%
PSS	0%	0.11%

TABLE II
Summary of packet loss caused by Kaleidoscope under different shuffling strategies.

DHCP [22]. 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 plan to investigate opportunities for more active integration of DHCP management with network address shuffling, *e.g.*, support for adaptive management of address pools.

Integration with Routers. One alternative to our middlebox deployment is using 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 bypass the shuffler. Alternately, one could build the route shuffling request as a new OSPF message type.

B. Collaborative Shuffling

A possible extension of Kaleidoscope is a collaborative shuffling infrastructure (CSI) that provides Internet-wide honeynet defense. Our architecture makes such an extension quite straightforward, *i.e.*, without any changes to the Internet routing infrastructure. We envision CSI as a globally distributed overlay network [13] with Kaleidoscope nodes providing *on-demand address shuffling service*. The service would maintain a list of honeynet and non-honeynet (live host) address blocks. For each address block, the system would also maintain a current list of *producer*(owner) and *consumer* networks. The consumers of network blocks would change periodically when a given network segment has seen sufficient scans. The producers remain constant and all traffic gets routed via overlays between the producers and consumers. The benefit for participants is gaining real-time perspective on attack activity at other networks and enhanced heterogeneity of the local network. Further, one could imagine anonymizing producer networks by onion routing the communication over Tor [10]. A CSI could also enable networks to efficiently share information that could be used in building real-time blacklists of probe-response attack sources. There are obvious challenges that we need to tackle such as privacy and economic implications. We intend to investigate them in future work.

VI. Summary and Conclusion

We present an address camoflauging methodology for safeguarding honeynet monitors from being mapped by adversaries. We designed and built a prototype middlebox system that 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 laboratory 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 middlebox 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.

VII. Acknowledgements

The authors thank Phil Porras and the GI 2007 symposium reviewers for encouraging and valuable feedback that improved our presentation of the paper. This material is based upon work supported through the U.S. Army Research Office under the Cyber-TA Research Grant No. W911NF-06-1-0316 and NSF grants No. CNS-0347252 and CCR-0325653. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or ARO.

References

[1] S. Agarwal, J. Sommers, and P. Barford. Scalable network path emulation. In *Proceedings of IEEE MASCOTS*, 2005.

[2] M. Allman. On the performance of middleboxes. In *Proceedings of Sigcomm Internet Measurement Conference*, 2003.

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

[4] S. Antonatos and K. Anagnostakis. Protecting against Hitlist Worms using Transparent Address Obfuscation. In *Proceedings of CMS*, 2006.

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

[6] J. Bethencourt, J. Franklin, and M. Vernon. Mapping Internet sensors with probe response packets. In *Proceedings of USENIX Security Symposium*, 2005.

[7] J.-Y. Cai, V. Yegneswaran, C. Alfeld, and P. Barford. HoneyGames: A Game-Theoretic Approach to Defending Network Monitors. University of Wisconsin, Technical Report, 2006.

[8] M. Casado, T. Garfinkel, W. Cui, V. Paxson, and S. Savage. Opportunistic measurement. In *Proceedings of ACM Hotnets*, 2005.

[9] E. Cooke, M. Bailey, M. Mao, D. Watson, F. Jahanian, and D. McPherson. Toward understanding distributed blackhole placement. In *Proceedings of CCS WORM*, 2004.

[10] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, 2004.

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

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

[13] A. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure overlay services. In *Proceedings of ACM Sigcomm*, 2002.

[14] D. Kewley, J. Lowry, R. Fink, and M. Dean. Dynamic approaches to thwart adversary intelligence gathering. In *DISCEX*, 2001.

[15] S. Khattab, R. Melhem, D. Mosse, and T. Znati. Honeypot Back-propagation for Mitigating Spoofing Distributed Denial-of-Service Attacks. In *Proceedings of SSN*, 2006.

[16] E. Kohler, R. Morris, and B. Chen. Modular components for network address translation. *IEEE OPENARCH*, June 2002.

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

[18] C. Kreibich and J. Crowcroft. Honeycomb–creating intrusion detection signatures using honeypots. In *Proceedings of ACM Hotnets*, 2003.

[19] 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, 2002.

[20] 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.

[21] M. A. Rajab, F. Monrose, and A. Terzis. Fast and evasive attacks: Highlighting the challenges ahead. In *Proceedings of RAID*, 2006.

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

[23] RFC 3234 - Middleboxes: Taxonomy and issues. http://www.faqs.org/rfcs/rfc3234.html.

[24] Y. Shinoda, K. Ikai, and M. Itoh. Vulnerabilities of passive internet threat monitors. In *Proceedings of USENIX Security Symposium*, 2005.

[25] S. Sinha, M. Bailey, and F. Jahanian. Shedding Light on the Configuration of Dark Addresses. In *Proceedings of NDSS*, 2007.

[26] J. Sommers and P. Barford. Self-configuring network traffic generation. In *Proceedings of ACM SIGCOMM Internet Measurement Conference*, 2004.

[27] J. Sommers, V. Yegneswaran, and P. Barford. A framework for malicious workload generation. In *Proceedings of ACM SIGCOMM Internet Measurement Conference*, 2004.

[28] J. Ullrich. Dshield. http://www.dshield.org, 2007.

[29] 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*, 2005.

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

[31] V. Yegneswaran, J. T. Giffin, P. Barford, and S. Jha. An architecture for generating semantics-aware signatures. In *Usenix Security Symposium*, 2005.