Shedding Light on the Configuration of Dark Addresses

Sushant Sinha, Michael Bailey, and Farnam Jahanian
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, Michigan, 48109
{*sushant, mibailey, farnam*}*@umich.edu*

Abstract

A popular approach to detecting and characterizing threats such as worms and botnets involves the use of sacrificial host collections called honeynets. These collections are explicitly deployed to be scanned, compromised, and used in attacks. Unfortunately, existing approaches to deploying honeynets largely ignore the problem of configuring operating systems and applications on individual hosts, leaving the user to configure them in a manual and often ad hoc fashion. In this paper, we demonstrate that such ad hoc configurations are inadequate–they misrepresent the security landscape of the networks they are trying to protect and are relatively easy for attackers to discover. We show that manually building configurations with good visibility and resistance to discovery is hard, as each network has its own unique threat and vulnerability spaces, and the potential number of hosts to configure in the honeynet is quite large. Therefore, we need automated systems to assist network and security administrators in building honeynet configurations. We argue that honeynets with individually consistent hosts and proportional representation of the network will achieve the two desired goals of visibility into network attacks and resistance to discovery. We develop an automated technique based on profiling the network and random sampling to generate these honeynet configurations. Through experimental evaluation and deployment of configurations generated by our technique, we demonstrate significantly more visibility and higher resistance to discovery than current methods.

1 Introduction

The rapid growth in malicious Internet activity, due to the rise of threats like automated worms, viruses, and recent bots like AgoBot [10], has driven the development of tools designed to protect host and network resources. The fundamental problem in instrumenting live (i.e., production) hosts is that such instrumentation affects the performance and availability of these systems. A security approach that alleviates these problems is to create a non-productive host (also called a honeypot) and place it at an unused or dark address. Honeypots are host and network resources that do not have production value and so any interaction with a honeypot is suspicious. For example, a honeypot deployment might consist of a mail server that is configured identical to a production mail server and placed nearby to detect possible intrusions against the real mail server. Because no legitimate clients are configured to use the honeypot mail server, any traffic to it is the result of malicious activity or misconfiguration.

Individual honeypots provide excellent visibility into threats that affect specific host and operating system configurations. However, detecting threats quickly with only a few honeypots is difficult. In order to quickly capture threat details, a number of commercial and non-profit organizations [2, 34], academic projects [6, 22, 36, 37], and tools [4, 25] have started monitoring large numbers of unused and dark addresses. We collectively call honeypot systems that monitor multiple unused and dark addresses *honeynets*.

Unfortunately, as systems have moved from a single honeypot to a network of honeypots, the problem of determining the configuration of these systems has largely been ignored. A single honeypot is easily configured with the operating system and applications matching an existing production system. However, for a large number of addresses, the network administrator must make decisions about what systems they want to protect and what attacks they wish to observe. For example, a single academic network in this study with 5,512 hosts had over 1,386 unique host and network configurations (see Table 4), and over 3000 unused addresses available for monitoring [11]. This effort is further complicated as the number of vulnerable services and potential attacks increase on the Internet. For example, Symantec documented 1,896 new software vulnerabilities from July 1, 2005 to December 31, 2005, over 40% more than in 2004 [33]. In spite of the difficulty of the task, little

work has been done on how to automate honeynet configuration, leaving administrators to perform the task manually, and often in a generic or ad hoc fashion.

In this paper, we show the potential limitations of generic and ad hoc approaches to honeynet configurations, and demonstrate that they lack visibility into network attacks. As an example, Table 1 shows that the most popular service on an example network is SSH, but attackers are focused primarily on nonexistent services such as Microsoft SQL Server. The honeynets in this network report neither of the two as the most critical security threat, indicating NetBIOS instead. As the example illustrates, these system do not represent the services running on the network (i.e., the vulnerable population) and fail to capture the existing attacks observed at the network (i.e., the threat landscape). Furthermore, we show that these honeynet configurations present very unusual operating systems and service configurations in many production networks and are trivially easy to detect, avoid, and corrupt (i.e., fingerprint). However, configuring honeynets is not trivial, as we find that there is lack of generality in vulnerable populations and threat landscapes over time and across networks. Furthermore, the potential number of addresses to configure may be very large and therefore, we require automated processes to configure a honeynet.

In order to automatically generate honeynet configurations with the goals to provide visibility into the vulnerable population and avoid detection, we argue that honeynets should be configured with individually consistent hosts and proportionally represent the surrounding network. We provide a simple, automated approach based on profiling the network and random sampling to generate honeynets with individual host consistency and proportional representation of the network. We evaluate our approach by applying it to live networks and deploying the configurations generated. We find that the honeynets achieve the desired goals of providing an accurate view of threats to the networks and resistance to discovery. For example, the honeynets representative to academic networks accurately show that SSH brute force attacks are the most widely impacting threat, and those representative to web server farms accurately show web proxy attacks as the most widely impacting threat during the period analyzed.

To summarize, the main contributions of this paper are:

- We show that ad hoc configurations do not provide visibility into the vulnerable population or into the threat landscape, and they are highly unusual in many networks, causing easy detection and fingerprinting.
- We justify the need for automatic configuration methods and then identify visibility and resistance to fingerprinting as the two goals of such a configuration system.
- We develop a simple technique that achieves the two critical goals automatically and we evaluate this technique through network monitoring and deployment of honeynets in production networks.

The remainder of this paper is structured as follows: Section 2 discusses the current state-of-the-art in honeynet configuration, and Section 3 shows the limitations of ad hoc configurations. Section 4 provides the compelling reasons for automatically generating configurations, and discusses individual host consistency and proportional representation of the network as important properties that make such a configuration. Section 5 describes our algorithm for honeynet configuration and its evaluation. Section 6 concludes our work with directions for future work.

2 Background

The way in which honeynets are configured greatly impacts their effectiveness. In an effort to understand how honeynets are currently configured, we have examined publicly available honeynet deployment data. We find that these deployments are often tied with a specific honeynet tool or technique. Each of these tools, in turn, has its own set of configuration capabilities. One of the main factors that distinguishes each of these honeypot systems is their level of interactivity [31]. Interactivity defines the degree to which a honeypot behaves like a real end-host system, as a broad spectrum of end-host behaviors can be emulated in order to capture increasing amounts of attack details. Therefore, we group these deployments by their level of interactivity and discuss the configuration capabilities of each as well as their limitations.

On the lowest end of interactivity, a passive monitor logs incoming traffic to unused addresses without any response [22]. For example, the Team Cymru [14] and Moore *et. al.* [22] have deployed passive monitors in unused addresses. Passive monitors do not allow for any response and have no service configurations. Because they do not emulate any services, they provide limited visibility into threats and require advanced techniques to fingerprint [27].

A light-weight TCP responder replies with a TCP SYN-ACK packet for every TCP SYN packet received on every TCP port in order to recover the first payload from TCP worms [6]. The Internet Motion Sensor project [13] has deployed a lightweight TCP responder in 28 monitored blocks within 18 networks, with sizes of these blocks varying from /24 to /8 [5]. These systems emulate part of the network stack, accept incoming connections on all ports, but provide no application-level response. They provide minimal visibility into all applications, but are trivial to find. We call this configuration **All TCP Responder**.

Honeyd emulates network stacks of multiple hosts on a

single machine and has a plug-in system for service emulation modules [25]. The Brazilian Honeynet Team [32] deployed numerous sensors of sizes from /28 to /24 with Honeyd, and the German Honeynet Team has also deployed a /18 honeyd sensor [20]. Finally, the French Honeynet Project [17] and the Norwegian Honeynet Team [23] have also deployed Honeyd sensors. Honeyd has a great deal of flexibility in honeypot configuration, enabling arbitrary personalities to be specified for each host. We call the default Honeyd 1.1 configuration [31] **Generic Honeyd**. To simplify configuration of large addresses, Honeyd allows declaration of default host configuration and ties it to all dark addresses that have not been assigned any host configuration. To automatically generate Honeyd configurations, RandomNet [30] allows operators to choose operating systems and emulated services, and constructs hosts with random combinations of chosen OSs and services. We call this configuration **Random Honeyd**.

Nepenthes emulates certain vulnerabilities to recover more exploits for the vulnerabilities [4]. The Chinese Honeynet Team has deployed a /24 Nepenthes sensor [8], the UK Honeynet Team [35] has deployed two Nepenthes sensors, the German Honeynet Team has deployed a /18 Nepenthes sensor [20], and the Georgia Honeynet Team [19] has started experimenting with Nepenthes. While some configuration of these systems exists, as various vulnerability modules can be turned on or off, most deployments are limited to those modules that already exist. We call the default Nepenthes 0.1.7 configuration **Nepenthes**.

Numerous end-host systems can be run on the same hardware platform by using virtual machine tools such as VMWare [24]. A variety of academic projects have deployed VMWare-based solutions including Georgia Tech [15], UCSD [36], and Purdue [21]. The configuration of end-hosts is limited only by the number of operating system and application compatibilities. However, a common approach is a vanilla operating system install, such as those promoted in [24].

In summary, we find that most published honeynet deployments use well-known techniques for honeynet construction. These techniques have a variety of configuration characteristics that range from no configuration to a fixed configuration and finally, to fully configurable. When configuration options exist, there is no automated system to choose between these options and are left to the administrator for manual configuration.

3 Limitations of Ad Hoc Honeynet Configurations

One of the original goals of deploying honeynets is to provide protection to an organization by understanding the means and motives of attackers. However, this important goal can be achieved only if the honeynets are configured to represent the network and services. In the previous section, we showed that the configuration of honeynets has largely been ignored by existing deployments and tools. In this section, we argue that ignoring this problem is dangerous because lack of proper configuration impacts both the visibility that these honeynets can provide as well as their ability to avoid detection.

3.1 Impact of ad hoc configuration on visibility

In the previous section, we observed that, while some flexibility exists in how honeynets can be configured, the absence of automated techniques has caused this process to be manual. If careful attention is not paid to this manual process, these configurations can drastically impact the effectiveness of the monitoring systems. The effectiveness depends on the honeynet ability to protect important network resources from the most likely threats. As a result, improperly configured honeynets can fail to be effective if they can not accurately represent:

- **the vulnerable population.** The vulnerable population is the set of services on the network that can be remotely accessed. It is important to note that this definition of vulnerable population includes all services on the network rather than only those with known vulnerabilities.
- **the threat landscape.** The threat landscape describes the current attacks on the network.

To understand the impact of ad hoc configurations on visibility, we compared them to the threat landscape and the vulnerable population. Table 1 shows the top 10 open TCP ports found on an academic /16 network together with the top 10 attacked ports on a /24 honeynet in the network, and several honeynet configurations. TCP ports represent broad characterization of the vulnerable population, the threat landscape, and the threats a honeynet configuration can capture. **All TCP Responder** provides minimal visibility into all ports and is not shown in the table. The results of the comparison are rather startling. *The ad hoc configurations of honeynets do a poor job of representing both the attacks to the network as well as the potentially vulnerable population.* This lack of visibility has profound implications, as the monitoring system will fail to provide any insight into attacks at large numbers of vulnerable hosts (e.g., those running TCP/106, TCP/311, TCP/427). In addition, these systems fail to capture the attacks prevalent on the network (e.g., those targeting TCP/1433, TCP/1080, TCP/1521, TCP/5900) and may make certain attacks (e.g., TCP/139) appear to be more important on the network than they in fact are. Interestingly, it also appears that attackers

TCP Port	% of Vulnerable Population	% of Attacks	Honeynet Configurations		
			% of **Nepenthes** hosts	% of **Generic Honeyd** hosts	% of **Random Honeyd** hosts
21	24	-	100	4	45
22	53	-	100	3	-
23	34	-	-	3	65
80	26	0.4	100	4	62
106	14	-	-	-	-
135	9	0.6	100	-	-
139	17	0.5	100	96	74
311	14	-	-	-	-
427	26	-	-	-	-
445	10	1.4	100	1	40
497	28	-	-	-	-
554	-	0.9	-	-	-
1080	-	1.7	-	-	-
1433	-	6.8	-	-	-
1521	-	0.9	-	-	-
4444	-	0.5	-	-	-
5900	24	0.9	-	-	-

Table 1. The lack of visibility into both the threat landscape, as well as the vulnerable populations for a variety of different honeynet configurations.

do not necessarily target the most popular services. In fact, only a small number of common ports are actually popular across the threat landscape, vulnerable population, and configurations including 80, 135, 139, and 445. We will show in the next section that the threat landscape and the vulnerable populations change over time and networks, and these honeynet configurations will have difficulty in representing them too.

3.2 Impact of ad hoc configuration on fingerprinting

As we saw in the previous section, ad hoc honeynet configurations do not represent either the exploit space or the vulnerable population. In addition to reducing important visibility, this behavior can be effectively used by attackers to detect, or fingerprint, the locations of the honeynets within a network. This type of activity is important to defenders, as fingerprinted honeynets can be actively avoided or corrupted by attackers. The key insight that makes this fingerprinting possible is the observation that some combination of services and operating systems on an ad hoc configured honeynet may seem highly anomalous when compared with the rest of the network. In the next two sections, we discuss a novel technique for fingerprinting honeynets and show how ad hoc configurations are easy to spot within productions network using this technique.

3.2.1 Fingerprinting algorithm

Honeynets configured in an ad hoc fashion provide inconsistent view of services on a network. Because we do not know of any systematic approach to detect such configurations in a network, we developed a general approach that involves comparing service configurations within a subnet to the configuration of the entire network. A naive approach, such as examining each individual host in the subnet to see if it is anomalous with respect to the rest of the network, will only determine if that host has a unique configuration in the network. In order to compare collections of host configurations, we first perform active tests to detect host configurations, aggregate these configurations by common operating systems and services, and then compare the distribution of a new service in the subnets with the network as a whole.

Consider a set of tests $\{T_1, T_2, \ldots, T_n\}$ that can be performed on an Internet host, with the result of test T_i being any member of the set $R_i = \{r_i^1, r_i^2, \ldots, r_i^{m_i}\}$ of size m_i. For example, a test (or variable) that checks whether TCP port 139 is open or not on an Internet host has results (or values) in the set $\{open, closed\}$ of size 2. The combination of test values for an Internet host makes a *host profile*. For example, checking port 139 and port 445 on an Internet host

might have host profiles in the set {(closed, closed), (closed, open), (open, closed), (open, open)}. While the algorithm is test-set independent, in this paper we utilize three sets of tests. Tests constructed based on existing tools, such as nmap [18], easily differentiate TCP software by exploiting ambiguities in RFCs and implementations, and we call them **tcp**. We similarly developed a set of 26 new tests to identify a web server and its configurations and call them **http**. Finally, we test open TCP ports from 1-1024 with a simple program that attempts connection to these ports that we call **ports**.

Our main idea is to group hosts by test values and then compare the subnet with the network as a whole for the values of a new test. First, we order tests by the increasing value of their entropy, each of which is computed by the distribution of test values among the hosts in the network. The tests are ordered by increasing entropy because a test with lower entropy has most certain values for the hosts on the network and hence, a significant anomaly can be detected if a subnet differs from the dominant values found in the network. Second, we compare each subnet with the network for the values of first test, then we aggregate by values of this test and compare for values of the second test. We iterate this comparison between the subnet and the network over the determined test order until the last test.

To quantify anomalies for a new test, the values for the test is arranged on a normal distribution curve using the frequency associated with each value. Then, we measure the anomaly using a *test of significance* called ***z*-statistics** [16] because of three specific reasons. (1) The anomaly quantified by z-statistics is significant if the values of a new test for hosts on the subnet are *significantly* different from those on the network. (2) If the network exhibits significant variation in the values of the new test chosen for analysis, then the anomaly is of low magnitude. (3) If only a few hosts in the network are aggregated by test values, then it causes an anomaly of smaller magnitude for the new test values. For a formal definition and a small discussion of z-statistics, please refer to Appendix A. A complete illustrative example of the above algorithm is presented in Appendix B.

There are a wide variety of techniques for fingerprinting honeypots. Our goal is not to enumerate all possible attacks but rather to illustrate the importance of configuration artifacts in honeynet fingerprinting. One relevant approach for detecting honeypots involves probing and analyzing published records from honeypots [7] . However, this attack is possible only when attack statistics from honeypots are publicly available. Rajab *et. al.* [27] fingerprint passive darknets that do not respond to any probe whereas we discover active honeynets without a prior knowledge of how they are configured. This is achieved by comparing subnet configurations with the surrounding network and detecting anomalous configurations.

3.2.2 Detecting ad hoc configurations

In order to study the impact of ad hoc configurations in terms of their ability to be discovered, we embedded different honeynet configurations into six network (A/16 and B/16 are academic networks, and C#1/17, C#2/19, D#1/19 and D#2/19 are web-server farms) profiles. The network profiles were created by probing live networks and storing their configurations. The honeynet configurations were created by a variety of existing approaches. We then applied our fingerprinting algorithm to create a list of anomalous subnet configurations in each of these networks and sorted them into descending order of how anomalous they were (i.e., by z-values). Ideally, each of the embedded subnets (i.e., honeynets) should appear on the top of these lists. Therefore, we will evaluate a honeynet resistance to fingerprinting by the position (or rank) in the lists.

Figure 1 presents the rank of the first honeynet anomaly in six networks when the honeynet is configured using **All TCP responder**, **Nepenthes**, **Generic Honeyd** and **Random Honeyd** (average of ten random honeyd configurations). We find that the ad hoc honeynet configurations surface as the most anomalous configurations in almost all of the networks for many set of tests. This result is perhaps easy to realize for honeynets such as **All TCP Responder** with all ports turned on, but quite surprising for honeynets configured with individually consistent hosts (i.e., they look like a real host). This is because **Nepenthes**, **Generic Honeyd**, and **Random Honeyd** often emulate services or run service combinations that are anomalous in many networks. As was the case with visibility, the impact of ad hoc configurations can be profound, allowing attackers to easily fingerprint honeynets.

4 The Properties of Honeynet Configuration

In previous sections, we have shown that existing methods for honeypot configuration are largely manual in nature. We have shown that if proper care is not taken in the construction of these configurations, they can produce honeynet systems that fail to provide visibility and that are easy to fingerprint. In this section, we describe the process of generating honeynet configurations. First, we motivate the need to automate honeynet configuration by showing the lack of temporal generality of exploits and spatial generality of vulnerable populations, the diversity of systems in networks, and the large number of addresses to be configured. Second, we present visibility into vulnerable population and resistance to fingerprinting as the two desired objectives and argue that a honeynet configuration with individually consistent hosts and proportional representation of the network will achieve the desired objectives.

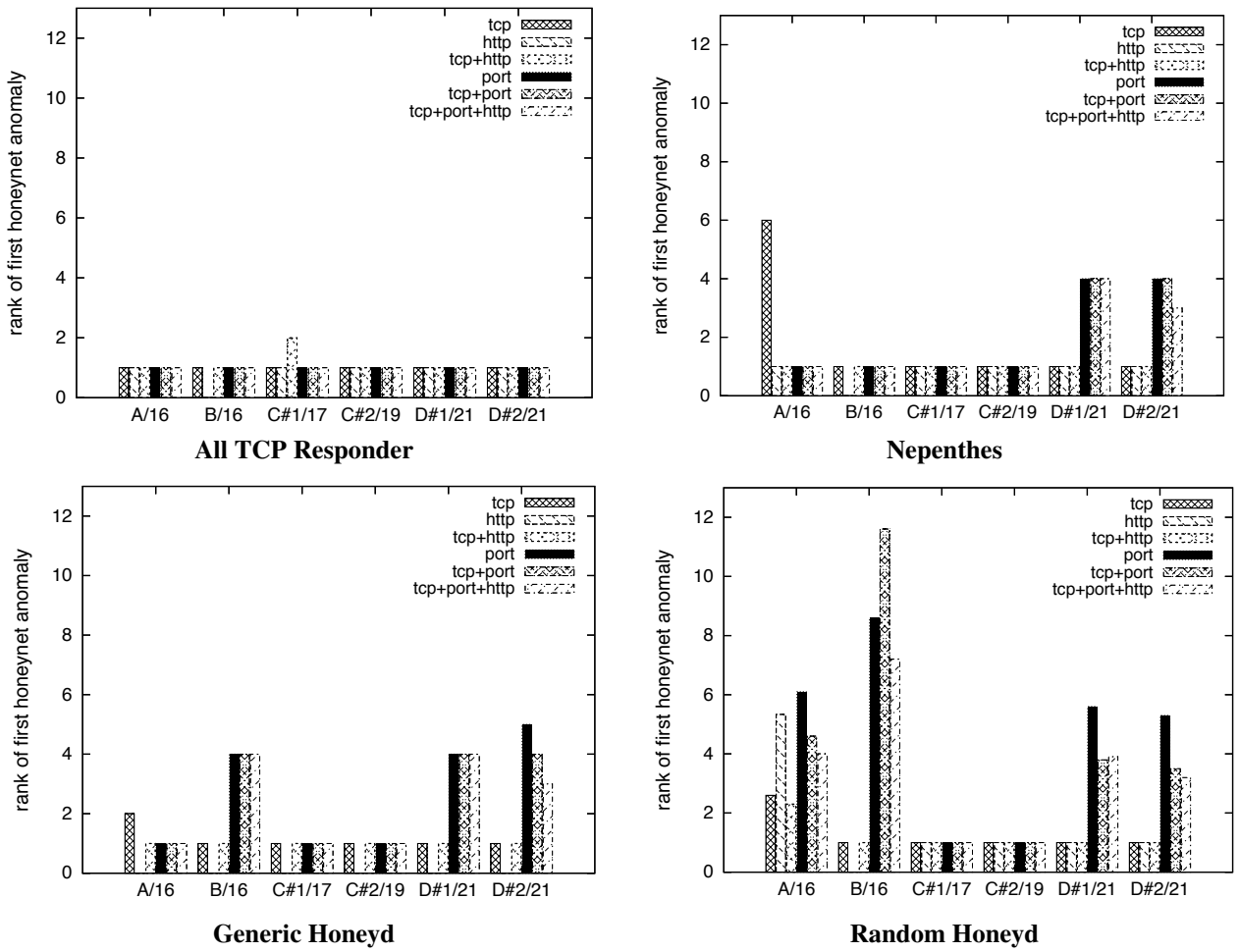


Figure 1. The rank of the first anomaly with ad hoc honeynet configurations in six networks for various combinations of tests that identify operating systems and services. Most honeynets represent the first, or most, anomalous subnet in a network

4.1 Need for automatic configuration

While we have clearly shown that ad hoc configurations are a danger to the goals of any honeypot deployment, we have not yet shown that the process of generating these requires anything more than careful manual configuration. We believe that two properties of the configuration space motivate the need for automatic configuration:

4.1.1 Lack of generality

A major impact on the effort of manual configuration of honeynets is the lack of temporal generality of attacks. Table 2 shows the top five TCP ports and the number of packets observed over a day for last five months on the /24 honeynet in the B/16 network. We find that new services were targeted heavily each month. The TCP ports 6000 and 1080 were the unusual ones targeted in April, the TCP port 5000 was targeted in May, the TCP ports 22 and 5900 were targeted in June, and TCP port 4444 in July. Moreover, the exploits observed vary significantly with locations, as demonstrated by significantly different data observed on sensors deployed in different networks [12]. Therefore, a highly variable nature of threat landscape makes chasing the exploits really difficult for the defenders, who must manually configure their honeynets.

Spatial generality also impacts the effectiveness of manually generating honeynet configurations. Differences in attack behaviors and vulnerable populations across networks make sharing of configuration a near impossibility. For example, Table 3 compares vulnerable population in various networks in two ways, by the operating system and the TCP ports. The second largest operating system in network A/16

04/19/2006	05/19/2006	06/19/2006	07/19/2006	08/19/2006
6000	445	22	135	1433
445	139	5900	80	1080
1433	5000	3128	4444	445
1080	1433	8080	445	5900
135	80	80	1433	1521

Table 2. The top 5 TCP ports observed in a /24 sensor in network B/16, over a period of 5 months. Exploits change quickly over time.

Operating System	Networks			
	A/16	B/16	C#1/17	D#1/19
Windows	44	25	76	77
Cisco IOS	14	7	-	-
Apple	9	36	-	-
Linux	9	7	15	6
HP printer	3	13	-	-
Solaris	9	7	1	2
*BSD	1	-	8	15

Operating Systems

TCP Port	Networks			
	A/16	B/16	C#1/17	D#1/19
139	42	17	-	-
22	41	53	30	25
135	39	10	42	69
23	27	34	4	5
445	27	11	-	-
80	21	26	93	96
25	12	10	70	83
21	8	24	77	79
427	4	26	-	-
497	3	28	-	-
110	1	-	39	17

TCP ports

Table 3. Comparing the vulnerable population in four networks, by operating systems and TCP ports. The vulnerable populations are different across networks.

is surprisingly Cisco IOS, which is found in the wireless access points and routers in the academic campus. On the other hand, Apple Mac OS is the dominant operating system in network B/16. As expected, the web-server farms were dominated by Windows servers. While SSH seems to be predominant service found in A/16 and B/16, HTTP, FTP and SMTP seems to be the dominant services in the web server farms. Therefore, the vulnerable population varies significantly with the nature of the networks making it difficult for a single honeynet configuration to represent the vulnerable population across networks.

4.1.2 Scale of configuration

Traditionally, a small number of individual honeypots were configured manually. For example, a dedicated host will be configured as a clone of a valuable network resource, such as a mail server. However, as the number of services and the number of hosts running them on the network have increased, defenders have been under increasing pressure to manage and provide visibility into these new services. Unfortunately, manual honeynet approaches for achieving this visibility are difficult to apply. Consider the data presented in Table 4, which shows surprisingly large numbers of TCP implementations, HTTP implementations, services, and the complex associations between them in various production networks. For example, in network A/16, we find that the 5,512 hosts result in 352 unique TCP stack implementations. Of those 5,512 hosts, 1,237 were running Web servers, and when probed, yielded 241 differing servers and configurations. This included TCP stacks and Web-servers from a bewildering variety of devices including NATs, wireless access points, printers, power switches, and webcameras.

In addition to the diversity in the numbers and types of individual host configurations, network administrators are also required to configure a potentially large number of unused addresses in order to maximize visibility. Traditional honeynet deployments of statically allocated network blocks are giving way to dynamic discovery of all unused addresses [11]. The number of addresses available can be surprisingly large, even in small organizations.

Network	Type of organization	Hosts	Web-servers	TCP	HTTP	TCP+ Ports	TCP+ HTTP	TCP+ Ports+ HTTP
A/16	university network	5512	1237	352	241	1210	699	1386
B/16	university network	1289	169	156	73	392	249	463
C#1/17	web-server farm	11342	10080	256	862	1625	1764	3394
C#2/19	web-server farm	2438	2208	93	293	394	559	811
D#1/19	web-server farm	1859	1513	118	221	330	451	590
D#2/19	web-server farm	1652	1171	137	208	266	417	487

Table 4. The number of unique host configurations observed at six production networks for various tests and their combinations. Each network has a surprisingly large number of unique configurations.

4.2 Individual host consistency and proportional representation

A number of objectives can be used to define honeynet configurations. For example, one may choose the easiest set of configurations to deploy, or one may want to represent global host or threat distribution. In this paper, we set our objective as providing visibility into the vulnerable population on the defender's network and resistance to fingerprinting. We chose these objectives so that the honeynet provides intelligence into zero-day threats to the network quickly. Even though we chose specific objectives, we believe that the general approach described in this paper (sample and configure) can be applied to configure honeynets representing other distributions (e.g., threat).

To provide visibility into network threats and resistance to fingerprinting, we argue that a honeynet configuration must have individually consistent hosts and should proportionally represent the network. An individually consistent host is one whose configuration appears in a live network. Individual consistency is necessary for two reasons. First, in order to provide visibility into threats that impact the defenders network, the configuration must provide the same opportunities for attackers as they occur in the real network. It is obvious that in order to capture threats against a service, you must run that service. Perhaps less obvious is that sophisticated worms, like Slapper, attempt to identify specific services and versions of operating systems to deliver the correct exploit [28]. In addition to visibility, individual host consistency provides resistance to fingerprinting against attackers that look for unusual services on the host.

In addition to individual consistency, we argue that visibility and resistance require proportional representation of the network. Honeynets are primarily deployed for detecting new threats. However, we do not know the operating systems and the service configurations that will be targeted by a new attack. Since we cannot anticipate any particular configuration, the ratio of honeynet hosts for any combination of operating system and services should be equal to the ratio of network hosts with those combinations. Proportionality provides visibility in that the most prominent software on the live network appear more frequently and hence can capture new threats to the software quickly. This also helps in prioritizing attacks by their widespread possible impact on the network, and hence arranging attacks by the maximum number of honeynet hosts that observed each one of them. Furthermore, when configuring large addresses with realistic operating systems and services, proportionality ensures resistance to fingerprinting in that the system will not add a number of configurations that skew the distributions of these configurations in the network as a whole.

5 A Simple Technique for Honeynet Configuration

Our algorithm for generating representative and consistent configurations has two components: profiling the network and generating configurations. Profiling the network is the process of determining the configuration of the existing production network for which we want visibility. After profiling the live network, we need to determine host profiles to be deployed on the honeynet. While we limit our discussion to the widest vulnerable population, our approach is also valid for specific vulnerabilities scanned by a tool such as Nessus [3].

The generated host profiles should be individually consistent and proportionally represent the network for all combination of tests values (as they identify operating systems and services). For example, consider a network with three hosts whose profiles for port (139, 445) tests are [(open, open), (open, closed), (closed, closed)]. A honeynet proportional to this network should meet multiple constraints. When each port is considered separately, $2/3$ of their hosts should have port 139 open, and $1/3$ of their hosts should have port 445 open. When the ports are considered in combination, $1/3$ of their hosts should have both ports open, $1/3$ of their hosts with port 139 open and port 445 closed, and

finally $1/3$ of their hosts with both ports closed [1]. Due to resource (address space and machines) constraints, a honeynet may only approximate the network. Therefore, we need to develop an algorithm that best approximates proportional representation for all combinations of test values, and at the same time, produces individually consistent profiles. However, it is difficult to construct an algorithm that minimizes error in proportional representation for *all* possible combinations of test values. Surprisingly a simple technique to select a *random sample* [9] of hosts in a network ensures our two critical properties:

- **Proportional representation.** The probability that a combination of test values $(T_{i_1} = r_{i_1}^{j_1}, T_{i_2} = r_{i_2}^{j_2}, \ldots, T_{i_k} = r_{i_k}^{j_k})$ is selected by the sampling algorithm is equal to the fraction of live hosts that satisfy them. Therefore, simple random sampling chooses host profiles with a probability distribution present in the actual network and approximates it in the selection of hosts. If n hosts are picked from a total population of N, then the standard error in estimating the ratio on the honeynet for a combination of test values with standard deviation σ is equal to $\sigma\sqrt{(\frac{1}{n} - \frac{1}{N})}$ [9]. Therefore, as more and more hosts are picked up to be represented on the honeynet, the better the honeynet represents the actual network.

- **Individual host consistency.** To be effective in capturing threats and in warding off possible fingerprinting efforts, we would like the configuration for each host on the honeynet to be individually consistent. This means that none of the hosts should be assigned an unusual configuration that is not possible in the real world. Since a configuration is assigned to a host in the honeynet only if there is a host with that particular configuration in the network, every host in the honeynet has a realistic configuration. Hence, for a subset of configuration values, a honeypot host will match a number of live hosts, yet represent a very specific live host when all configuration values are considered together.

The simple random sampling is effective because we do not know the host configuration that will be targeted by a new attack. However, when we know certain tests are more important than other ones, then we can achieve better proportionality to the important tests. For example, if we know that services are frequently being targeted, then we can achieve better proportionality for the service distribution than say the operating system distribution. This is usually achieved using *stratified sampling* [9]. Stratified sampling involves hierarchically separating the population and allocating the size to be sampled proportionally at each point in the hierarchy.

Previously we have assumed that all hosts in a network are equally important. Situations can easily arise in which this will not hold true. An operator might need better visibility to specific machines. For example, a system administrator may wish to weigh his DNS server, a single point of failure, twice when compared to other hosts on the network. To provide additional visibility to a machine when compared to others, additional matching configurations are added to the configuration pool before sampling. In addition, the operator might indicate a particular value for a test variable as more important than other values. For example, configuring a host with port80 *open* is more important than when it is *closed*. To account for these cases, we allow users to specify such preferences as input to tune honeynet configuration. Then a host is replaced by a preferred host before sampling. For example, if port 80 open is preferred over closed, then a Windows XP host might be replaced by Windows XP with IIS web server found in the network. This ensures that honeynet hosts are individually consistent even with user input.

Now we evaluate our approach to configuring honeynets. Recall that our objective was to create configurations that provide visibility into network attacks and resistance against fingerprinting. To demonstrate this, we validated our configurations by comparing with the vulnerable population and by applying the fingerprinting algorithm. To evaluate visibility of honeynets into real network threats, we created and deployed various honeynet configurations in production networks. Then, we examined the attacks observed by each of the honeynets, and show that non-representative configurations provide skewed views of the network threats. Finally, we deploy representative configurations for various other networks in our academic network and show that they provide different insights, even when deployed in a single network.

5.1 Evaluating correctness of representative configurations

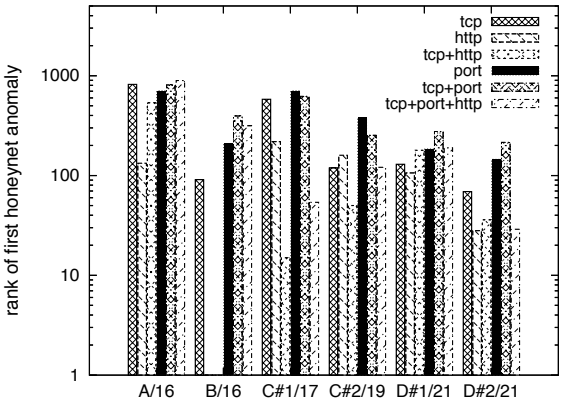
In order to achieve visibility, we have argued that a configuration needs to provide proportional representation of the target network. Table 5 (a) shows the visibility that a representative honeynet provides into the vulnerable population. We find that the distribution of network services and operating systems on the representative honeynet closely match the vulnerable population. The combination of operating systems and services between a representative honeynet and the network also matched closely and we omit those results here.

To evaluate honeynet configurations by their resistance

[1] an open port by may be preferred over a closed port, and we will discuss this later.

		% of Network Hosts	% of **Representative** Hosts
Operating Systems	Apple	36	35
	Windows	25	23
	HP printer	13	15
	Linux	7	5
	Solaris	7	6
TCP Ports	22	53	54
	23	34	34
	497	28	31
	427	26	27
	80	26	24

(a) Visibility into the vulnerable population



(b) Resistance to fingerprinting

Table 5. Evaluating representative honeynet configuration by (a) visibility into vulnerable population and (b) resistance to fingerprinting. The percentage of vulnerable hosts for the top five services and top five operating systems in network B/16 match closely to the representative honeynet. The representative honeynet is very resistant to fingerprinting as well.

Rank	Exploit	% Network Hosts	% **Representative** Hosts
1	RPC portmap rusers request UDP	90	100
2	NETBIOS SMB-DS C$ share unicode access	7	10
3	NETBIOS SMB-DS IPC$ share unicode access	7	10
4	BARE BYTE UNICODE ENCODING	4	24
5	WEB-MISC robots.txt access	4	24

Table 6. The visibility of configurations into the exploits on the network. The percentage of network hosts that observed each of the top five exploits in network B/16 is compared with the distribution in the representative configuration.

to fingerprinting, the representative configurations for six networks were embedded in the networks and the anomaly detection algorithm from Section 3 was used on all of the networks. Table 5 (b) shows the rank of the first honeynet anomaly for six networks when the honeynet is configured to be **Representative** to these networks. It shows that a representative honeynet is very resistant to techniques that attempt to detect network-wide anomalies.

5.2 Evaluating visibility of representative configuration by network monitoring

We demonstrated that the broad characterization of vulnerable population on the network matches the representative honeynet. In this section, we explore the visibility of representative honeynet into specific exploits on the network. For this, we capture exploits on the network B/16 using an intrusion detection system called Snort [29] and then compare honeynet visibility to those threats.

We monitored the gateway router to B/16 via a span port connected to a FreeBSD machine running Snort 2.1.3 configured with default signature distribution and recent bleeding-snort [1] signatures. The experiment was conducted for five hours on 2^{nd} May, 2006. The detected exploits were ordered by spread of the attack i.e., by the percentage of network hosts that observed the attack. We then grouped hosts that received a particular exploit and identified host characteristics necessary to receive the exploit. Finally, we analyzed the honeynet profile to discover susceptible hosts on the **Representative** honeynet. Table 6 shows the top five attacks (server-based) discovered on the network and compares it with the percentage of susceptible hosts on the honeynet. Ninety percent of the network hosts observed attempts to list currently logged users. This was

Configuration	Exploits	% of Honeynet hosts	% of Susceptible Network Hosts
Representative	1. SSH Brute-Force attack	41	53
	2. NETBIOS DCERPC ISystemActivator path overflow attempt	8	9
	3. SHELLCODE x86 NOOP	6	9
	4. NETBIOS SMB-DS C$ share unicode access	1	10
	5. NETBIOS SMB-DS IPC$ share unicode access	1	10
All TCP Responder	1. NETBIOS DCERPC ISystem Activator path overflow attempt	60	9
	2. WEB-IIS view source via translate header	17	26
	3. P2P GNUTella GET	5	0
	4. LSA exploit	4	10
	5. SHELLCODE x86 NOOP	4	9
Generic Honeyd	1. SSH Brute-Force attack	8	53
	2. SCAN Proxy Port 8080 attempt	2	100
	3. BACKDOOR typot trojan traffic	1	100
	4. Behavioral Unusual Port 135 traffic	1	9
	5. Squid Proxy attempt	1	100
Random Honeyd	1. WEB-IIS view source via translate header	21	26
	2. LSA exploit	6	10
	3. FTP anonymous login attempt	6	24
	4. MS04011 Lsasrv.dll RPC exploit (WinXP)	3	10
	5. MS04011 Lsasrv.dll RPC exploit (Win2k)	3	10

Table 7. The impact of different configuration approaches into threat visibility. Non-representative configurations are misleading about the real threats to the network.

targeting UDP port 111 and would have been visible on all honeynet hosts. Seven percent of network hosts observed administrative access attempts to SMB service on TCP port 445 and would have been visible on 10% of the honeynet hosts. The rest of the exploits were slow moving attacks and were observed on a few hosts on the network. Therefore, existing exploits on the network would be detected on the honeynet depending on their impact on the network.

5.3 Evaluating visibility of honeynet configurations by real deployments

In this section, we deploy honeynet configurations in B/16 and evaluate them by the accuracy of threat view they provide. First, we compare threat observed on different honeynet configurations with the vulnerable population on the network. Second, we deploy representative configurations for various networks in B/16 and observe how representative honeynets provide unique view of threats to a network.

We deployed a real /24 honeynet (monitoring 256 addresses) with different honeynet configurations in the B/16 network and exposed it to the present day attacks. Each honeynet configuration was deployed with Honeyd 1.0 [26], for a period of one day using the service scripts for FTP, SMTP, FINGER, HTTP and IIS-emulator, and the entire experiment lasted from 22^{nd} April, 2006 to 1^{st} May, 2006. We evaluated the results across two dimensions: the affect of various configuration approaches on the threats observed, and the impact of context in representative configurations. It is important to note that comparisons across configurations in this experiment are not valid due to temporal changes in threat. Instead, our goal is to evaluate a honeynet configuration by comparing it with the vulnerable population.

Table 7 compares a honeynet representative of network B/16 with three other honeynet configurations. It shows the top five exploits, ordered by the number of honeynet hosts that observed the exploit. For the **Representative** configuration, we found that the SSH Brute-Force attack was observed on a large number of honeynet hosts and it matched with the university network B/16, which has a significant number of hosts configured with SSH service. An attack on NETBIOS DCERPC service on port 135 was observed on a small percentage of hosts, as only 9% of the network

A/16	C#1/17	C#2/19	D#1/19	D#2/19
1. SSH Brute-Force attack	1. Web Proxy GET Request	1. Web Proxy GET Request	1. NETBIOS DCERPC ISystem-Activator path overflow attempt	1.NON-RFC HTTP DELIMITER
2. NETBIOS DCERPC ISystem Activator path overflow attempt	2. WEB-IIS view source via translate header	2. NETBIOS DCERPC ISystem Activator path overflow attempt	2. SSH Brute-Force attack	2. NETBIOS DCERPC ISystem-Activator path overflow attempt
3. MS04-007 Kill-Bill ASN1 exploit attempt	3. NETBIOS DCERPC ISystem-Activator path overflow attempt	3. HTTP Challenge/ Response Authentication	3. WebDAV search access	3. FTP anonymous login attempt

Table 8. The importance of context. The top three attacks captured on honeynets representatively configured for five different production networks, but placed within the same network.

hosts were susceptible to this attack. On the other hand, **All TCP Responder** observed a NETBIOS DCERPC attack on a 60% honeynet hosts. The **Generic Honeyd** configuration observed the SSH brute-force attack on only 8% of the hosts, but 53% of the hosts were susceptible to this attack. The **Random Honeyd** configuration observed an IIS attack on 21% of the hosts, which was close to 26% of Web servers in the network. However, around 50% of **Random Honeyd** hosts were capable of receiving the exploit (much higher than what actually received) but did not receive it because of slow moving attacks. When examining these results, we find that the non-representative configurations bias the threat view of a network away from the exploits that are actually the most widely affecting, and hence most important.

To analyze how well a representative configuration reflected attacks on a network, we deployed honeynet configuration representatives for five other networks. Each of them were deployed for one day in the /24 unused address space in the B/16 network. Table 8 compares representative honeynet configurations with the susceptible network population for five networks. We find that the SSH Brute force attack was commonly observed on honeynet hosts, and it coincided with the large number of hosts in the university network A/16 that were configured with SSH service. The dominant exploits found on the web server farms were those impacting web service and the TCP port 135, which are the dominant services in those networks. What is interesting to note is that, although these configurations were placed within the same network, the configurations observed vastly different views depending on the network vulnerability spaces.

In this section, we have examined a variety of different aspects of visibility. We have shown that the representative honeynet provides accurate view of threats on the network than ad hoc configuration methods whose results are often misleading. We also demonstrated that representative configurations observes different threat views depending on the network vulnerability space.

6 Conclusion

In this paper, we addressed the problem of honeynet configuration. We showed that the existing approaches to configure a honeynet are manual, causing honeynets to be configured either in a generic or ad hoc fashion. We demonstrated the limitations of generic and ad hoc configuration in that they provide poor visibility into network attacks and they are easy to discover. We show that providing visibility into network attacks is not trivial because the threat landscape and the vulnerable populations change with time and across networks. Furthermore, large number of dark addresses available in an organization requires automated approaches to configure honeynets. We identify individual host consistency and proportional representation as two properties required to achieve the desired goals of providing visibility into vulnerable population and resistance to discovery. We then described an automatic approach based on profiling the network with active tests and random sampling that achieves the desired goals. We evaluated the efficacy of representative honeynets through deployment of these configurations in a production network.

In this paper, we have explored and evaluated honeynet configurations only for public facing networks. However, the deployment of firewalls and NATs have led to the need for internal honeynets [11] and we plan to evaluate the effectiveness of our techniques in these types of deployment. There are numerous means of discovering honeynets, and

we by no means attempt to address all of them. In this paper, we only examined configuration artifacts that can be used to aid attackers. In future, we plan to deploy representative honeynets with high interaction honeypots and detect exploits using host based intrusion detection systems.

Acknowledgments

This work was supported in part by the Department of Homeland Security (DHS) under contract number NBCHC040146, by the National Science Foundation (NSF) under contract number CNS 0627445 and by corporate gifts from Intel Corporation and Cisco Corporation. We would like to thank Evan Cooke for providing valuable feedback on early drafts and anonymous reviewers for critical and useful comments. We would like to also thank Dan Peek, Jose Nazario, Ed Balas, Dave Dittrich and Andrew Myrick for numerous insightful conversations.

References

[1] Bleeding edge Snort. http://www.bleedingsnort.com/.

[2] Honeynet research alliance. http://honeynet.org/alliance/index.html.

[3] Nessus: Vulnerability scanner. http://www.nessus.org/.

[4] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. Freiling. The nepenthes platform: An efficient approach to collect malware. In *9th International Symposium On Recent Advances In Intrusion Detection*. Springer-Verlag, 2006.

[5] M. Bailey, E. Cooke, T. Battles, and D. McPherson. Tracking global threats with the Internet Motion Sensor. 32nd Meeting of the North American Network Operators Group, October 2004.

[6] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson. The Internet Motion Sensor: A distributed blackhole monitoring system. In *Proceedings of Network and Distributed System Security Symposium (NDSS '05)*, San Diego, CA, February 2005.

[7] J. Bethencourt, J. Franklin, and M. Vernon. Mapping Internet sensors with probe response attacks. In *Proceedings of the 14th USENIX Security Symposium*, Baltimore, MD, August 2005.

[8] Chinese Honeynet Team. Project status report: Period September 2005 to March 2006. http://www.icst.pku.edu.cn/honeynetweb/honeyneten/statusreport_200604.h%tm, March 2006.

[9] W. G. Cochran. *Sampling Techniques, 3rd Edition.* John Wiley, 1977.

[10] Computer Associates. Win32.Agobot. http://www3.ca.com/securityadvisor/virusinfo/virus.aspx?id=37776, July 2004.

[11] E. Cooke, M. Bailey, F. Jahanian, and R. Mortier. The dark oracle: Perspective-aware unused and unreachable address discovery. In *Proceedings of the 3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI '06)*, May 2006.

[12] E. Cooke, M. Bailey, Z. M. Mao, D. Watson, and F. Jahanian. Toward understanding distributed blackhole placement. In *Proceedings of the 2004 ACM Workshop on Rapid Malcode (WORM-04)*, New York, Oct 2004. ACM Press.

[13] E. Cooke, M. Bailey, D. Watson, F. Jahanian D. McPherson, and Z. M. Mao. The Internet Motion Sensor Project. http://ims.eecs.umich.edu, June 2004.

[14] T. Cymru. The darknet project. http://www.cymru.com/Darknet/index.html, June 2004.

[15] D. Dagon, X. Qin, G. Gu, J. Grizzard, J. Levine, W. Lee, and H. Owen. Honeystat: Local worm detection using honeypots. In *Recent Advances in Intrusion Detection, 7th International Symposium, (RAID 2004)*, Lecture Notes in Computer Science, Sophia-Antipolis, French Riviera, France, Oct. 2004. Springer.

[16] D. Freedman, R. Pisani, and R. Purves. *Statistics*. Norton, 1998.

[17] French Honeynet Team. Status report: October 2005 – March 2006. , March 2006.

[18] Fyodor. Nmap. http://www.insecure.org.

[19] Georgia Tech Honeynet Team. Georgia tech honeynet report (September 15, 2005 – March 15, 2006). http://www.ece.gatech.edu/research/labs/nsa/honeynet/status/2006-03/ind%ex.shtml, March 2006.

[20] German Honeynet Team. Project status report: Period April 2005 to October 2005. http://lufgi4.informatik.rwth-aachen.de/projects/honeynet/status2005-2, October 2005.

[21] X. Jiang and D. Xu. Collapsar: A VM-based architecture for network attack detention center. In *Proceedings of the 13th USENIX Security Symposium*, San Diego, CA, USA, August 2004.

[22] D. Moore, C. Shannon, G. M. Voelker, and S. Savage. Network telescopes. Technical Report CS2004-0795, UC San Diego, July 2004.

[23] Norwegin Honeynet Team. Bi-annual status report 2005 q4. http://www.honeynor.no/docs/Honeynor_bi-annual_statusreport_2005q4.p%df, 2005.

[24] H. Project. Know Your Enemy: Learning with VMware. 2003.

[25] N. Provos. Honeyd — A virtual honeypot daemon. In *10th DFN-CERT Workshop*, Hamburg, Germany, Feb. 2003.

[26] N. Provos. A Virtual Honeypot Framework. In *Proceedings of the 13th USENIX Security Symposium*, pages 1–14, San Diego, CA, USA, August 2004.

[27] M. A. Rajab, F. Monrose, and A. Terzis. Fast and evasive attacks: Highlighting the challenges ahead. In *Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID)*, Sept. 2006.

[28] E. Rescorla. Security holes... Who cares? In *Proceedings of USENIX Security Symposium*, August 2003.

[29] M. Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of Usenix Lisa Conference*, November, 2001.

[30] A. B. Smith and J. M. Fox. RandomNet. http://www.citi.umich.edu/u/provos/honeyd/ch01-results/3/, March 2003.

[31] L. Spitzner. *Honeypots: Tracking Hackers*. Addison-Wesley, 2002.

[32] K. Steding-Jessen. The honeynet project: Distributed honeynet deployment in Brazil. http://www.honeynet.org.br/presentations/hnbr-dod-honeynet-project2006.%pdf, 2006.

[33] Symantec. Symantec Internet threat report: Trends for July '05 - December '05. http://www.symantec.com/enterprise/threatreport/index.jsp, March, 2006.

[34] Symantec Corporation. DeepSight Analyzer. http://analyzer.securityfocus.com/, 2005.

[35] UK Honeynet Team. Status report for the period October 2005 – March 2006. http://www.ukhoneynet.org/reports.html, March 2006.

[36] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker, and S. Savage. Scalability, fidelity and containment in the Potemkin virtual honeyfarm. In *Proceedings of the 20th ACM Symposium on Operating System Principles (SOSP)*, Brighton, UK, Oct. 2005.

[37] V. Yegneswaran, P. Barford, and D. Plonka. On the design and use of Internet sinks for network abuse monitoring. In *Recent Advances in Intrusion Detection—Proceedings of the 7th International Symposium (RAID 2004)*, Sophia Antipolis, French Riviera, France, Oct. 2004.

Appendix

A: Z-Statistics

Z-Statistics is formally defined as:

$$z = \frac{average\ of\ the\ sample - average\ of\ the\ population}{standard\ error\ per\ object}$$

where *standard error* is dependent on the *standard deviation* and the size of the sample as:

$$standard\ error\ per\ object = \frac{standard\ deviation\ of\ population}{\sqrt{size\ of\ the\ sample}}$$

Therefore, a small sample will result in high *standard error* that will reduce z values. Furthermore, high standard deviation implies significant variation in value for the new test and would cause high standard error resulting in low z value.

Z-Statistics is primarily used to measure the likelihood of a predicted average when only the average of a sample of population is known. However, this requires computing the standard deviation of the population. This is particularly difficult to estimate for many statistics problems (e.g., predicting poll outcome, estimating number of people suffering from AIDS) as it requires gathering the values of each object in the population. Statisticians approximate the standard deviation of the population by assuming that the standard deviation of the population is the same as that of the sample. We can compute the average and the standard deviation of the population easily, as we can probe the entire network with lightweight tests. Hence, the z value *accurately* measures the degree of inconsistency between the subnet hosts when compared with the network hosts.

B: Illustrative example

Figure 2(a) shows the configuration of machines in 10.0.0.0/29 with different subnets. Figure 2(b) detects, quantifies, and produces reasons for various anomalies on the network using our algorithm. First, it orders the tests by their increasing entropy values. The order of the tests is 445, 497, 139, 22, and 135, which can be observed on the successive depths of the tree. The values assigned to test results are shown on tree edges. For port 445, the subnet 0 has an average of $1/2$ while the network has an average of $1/8$. The standard deviation for test 445 is $\sqrt{\frac{1\times(1-1/8)^2+7\times(0-1/8)^2}{8}} = 0.33$, and the standard error per object is $std.dev./\sqrt{2} = 0.23$. Therefore, the magnitude of anomaly when subnet 0 is analyzed for the test 445 is $z = \frac{1/2-1/8}{0.23} = 1.6$. Similarly, subnets 1, 2, and 3 produce a very low z-value of 0.53. However, for hosts aggregated by port 445=open, analyzing port 22 for subnet 3 produces a strong z-value of 2.23. Hosts in subnet 0 with port 445 open, when analyzed for port 22, produce a subnet average of 0, a network average of $2/7$, a standard deviation of $\sqrt{\frac{5\times(0-2/7)^2+2\times(1-2/7)^2}{7}} = 0.45$, a high standard error of 0.45 because of small sample size, and a low z-value of -0.63 (only positive values shown in the figure). For subnet 2, hosts with port 445 open and port 22 closed, when analyzed for port 135, raise a small anomaly with a z-value of 1.1. Figure 2(c) presents the reasons and z-values for the top three anomalies in the subnet. A node with single edge does not classify reason for an anomaly and can be removed from the reason. For example, the top anomaly, because of 22/445=open, 497=closed and 139=open, has been reduced to just the reason 22/445=open.

		22	135	139	445	497
		ssh	msrps	netbios	smb	dantz
Mac OS	10.0.0.0	open				open
Windows	10.0.0.1		open	open	open	
Windows	10.0.0.2		open	open	open	
Windows	10.0.0.3		open	open	open	
Linux	10.0.0.4			open	open	
Windows	10.0.0.5		open	open	open	
Linux	10.0.0.6	open		open	open	
Linux	10.0.0.7	open		open	open	

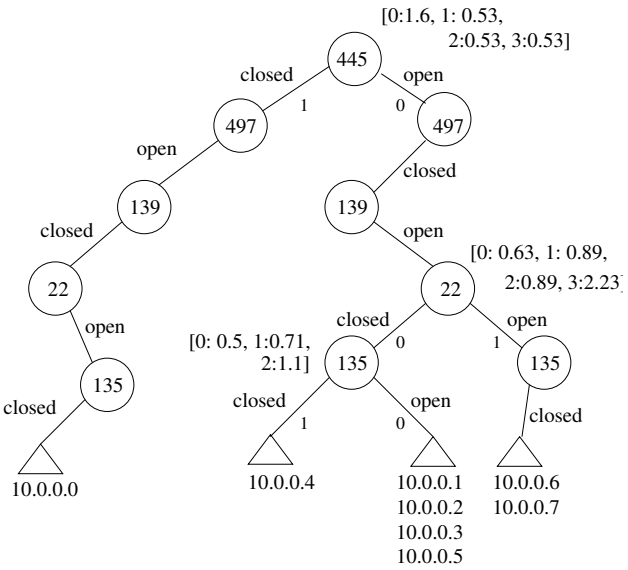
subnet 0
subnet 1
subnet 2
subnet 3

(a)

Top 3 Anomalies

subnet	reason	z−value
3	22/445=open	2.23
0	445	1.6
2	135/445=open, 22=closed	1.1

(c)



(b)

Figure 2. (a) Eight systems over the network 10.0.0.0/29 with configuration for different ports (unmarked ports are closed). (b) The anomaly computed for different subnets at various aggregation points in the graph (shown as subnet: z-value). (c) The top three anomalies in the network together with the reason that triggered the anomaly.