Diala , Jour , Volume , 31 , 2008

Design of Dynamic Honeypot for Intrusion Detection

By
Adel A. Ghedan and Bashar Talib Hameed
Diyala University / Collage of Science / Computer Science Departement

Abstract

A modern technology in the area of intrusion detection is honeypot technology that unlike common IDSs tends to provide the attacker with all the necessary resources needed for a successful attack. Honeypots provide a platform for studying the methods and tools used by the intruders (blackhat community), thus deriving their value from the unauthorized use of their resources. This paper discusses the design of a dynamic honeypot, which is an autonomous honeypot capable of adapting in a dynamic and constantly changing network environment. The dynamic honeypot approach integrates passive or active probing and virtual honeypots. This approach addresses the challenge of deploying and configuring virtual honeypots

الخلاصة:

ان التقنية الحديثة و الخاصة بالعمل على اكتشاف و تعقب المتطفلين باستخدام تقنية جرار العسل و التي تقود الى عملية اليى عملية توفير كل المصادر المطلوبة حول هوية المهاجمين اي عملية اكتشاف المهاجمين الناجحة، جرار العسل توفر رصيف او ارضية جيدة لدراسة الطرق و الادوات الخاصة بعملية التعقب تلك و من خلال عملية الاشتقاق لتلك المصادر يمكن توفير معلومات جيدة و ناجحة عن عملية التقب. في هذا البحث يناقش تقنية جرار العسل المتحركة و التي لها القابلية على التكيف مع عملية تحويل في بيئة و هيئة الشبكات المستخدمة، كذلك فان جرار العسل المتحركة تقدم طريقة صحيحة و جيدة و فعالة في عملية التقصي باستخدام جرار العسل الوهمية كما ان البحث يقدم و يوفر عناوين و التي من خلالها يتم التحدي من خلال الانتشار و تشكيل جرار العسل الوهمية حول الشبكات المتحركة و الذي يتم من خلال توفير جرار العسل الافتراضية التي من خلالها يتم حجب و اكتشاف الاشخاص المتطفلين داخل الشبكة التحركة.

Keywords: IDS (Intrusion Detection System).

1. Introduction

Security in the enterprise has lately become the primary concern of both IT managers and other executives. The challenges of securing enterprise networks in the face of intruders armed with the tools of compromise have become overwhelming and are still growing. To improve their network security, organizations have sought solutions such as firewalls, Virtual Private Networks (VPNs), and intruder detection variants. All of these solutions, however, continue to leave proprietary data accessible to determined intruders. The Computer Security Institute/FBI 2003 Computer Crime and Security Survey indicate that the total annual losses reported in the 2003 [1].
Moreover, many of the more advanced current network security solutions require a good deal of administration, with consistent needs for reconfiguration, management, and report analysis. With security administrators supporting an ever growing number of users, such consistent interaction with security mechanisms has become impractical. Therefore, today's enterprise requires a security solution that will not only prevent the most advanced intruder, but will as well accomplish this with minimal configuration and supervision.

2. Intrusion Detection Systems

Intrusion detection is the process of monitoring computers or networks for unauthorized entrance or activity. IDS can also be used to monitor network traffic, thereby detecting if a system is being targeted by a network attack. There are two basic types of intrusion detection: host-based (HIDS) and network based (NIDS). Each has a distinct approach to monitoring and securing data, and each has distinct advantages and disadvantages.
Host-based IDSs examine data held on individual computers that serve as hosts; they are highly effective for detecting insider abuses. Examples of host-based IDS implementations include Windows NT/2000 Security Event Logs, and UNIX Syslog. On the other hand, Network-based intrusion detection systems analyze data packets that travel over the actual network. These packets are examined and sometimes compared with empirical data to verify whether they are of malicious or benign nature [2]. An example of NIDS is Snort [3], which is an open source network intrusion detection system that performs real-time traffic analysis. It can be used to detect a variety of attacks and probes, such as buffer overflows, stealth port scans, and OS fingerprinting attempts.
The rest of the paper is organized as follows. In Section 3, we provide coverage of previous work in the domain of honeypot technology and intrusion detection systems. In Section 4, the design of the new system of dynamic honeypots is presented with a description of the associated algorithms. Finally in Section 5, we finish the paper with concluding remarks.

3. Related Work

The honeypot technology is an attempt to overcome the shortcomings of intrusion detection systems.

3.1. Definition

"A honeypot is an information system resource who's unauthorized or illicit use of that resource." [4].

Conceptually almost all honeypots work in the same manner. They are resources that have no authorized activity and no production value. Thus, in theory, a honeypot should not see any traffic because it has no legitimate activity. Hence any interaction with a honeypot is most likely an unauthorized and any connection attempts to a honeypot are most likely probes, attacks, or compromises.

3.2. Types of Honeypots

There are two general categories for honeypots, lowinteraction and high-interaction honeypots. Interaction defines the level and extent of activity a honeypot allows an attacker.

1. Low-interaction honeypots: These types are limited in their extent of interaction; they are actually emulators of services and operating systems, whereby attacker activity is limited to the level of emulation by the honeypot. An example of an emulated service is FTP service; listening on port 21 may just emulate an FTP login, or it may support a variety of additional FTP commands. The deployment and maintenance of such systems is fairly simple and does not involve much risk. However, there should be some maintenance so as to keep up with the constantly changing nature of the systems connected to the network. In addition the emulated services mitigate risk by containing the attacker's activity.

Unfortunately low interaction systems log only limited information and are designed to capture known activity. Moreover, an attacker can detect a lowinteraction honeypot by executing a command that the emulation does not support. Examples of lowinteraction honeypots include Specter [5], Honeyd [6], and KFSensor [7]. To overcome these short comings traffic targeted to a low interaction Honeypot can be redirected to a high interaction Honeypot. In the remainder of the paper we will use virtual honeypots and Honeyd interchangeably.
KFSensor simulates system services at the application layer, thus enabling it to use Windows security mechanisms and libraries. KFSensor can be used to set up new firewall rules and develop signatures for IDSs. Honeyd is a framework for virtual honeypots, which simulates a computer on the network and simulates the IP stack for various OSs and services. Specter is a smart honeypot system, which simulates a machine of any type, with a set of services for the attackers to use.
Specter generates decoy programs that will mark an attacking computer. Specter simply provides a complete simulated machine to be installed on the network.

2. High-interaction honeypots: These are different, whereby they are a complex solution and involve the deployment of real operating systems and applications. The advantages include the capture of extensive amounts of information and allowing attackers to interact with real systems where the full extent of their behavior can be studied and recorded. Examples of high-interaction honeypots include Honeynets Sebek [8].

4. Dynamic Honeypots

The following design of dynamic honeypots is a contribution in response to a proposed idea by the Honeynet Organization [9]. One of the biggest challenges when deploying any type of a security system is in maintaining the functionality of the total system as the network topology or technology changes. This issue is especially critical for honeypots. It must be made sure that honeypots blend into the system and appear like any other entity on the network. The services should be chosen so as to mimic the real services that a certain OS is able to provide and keep up with new technologies while removing old and obsolete services. What is sought is a dynamic honeypot that we can just plug in and leave it to operate without the need to constantly update it. It should be able to automatically identify unused IPs and deploy virtual honeypots on this IPs. Also any time a system is removed or added to the network; dynamic honeypots should be able to reconfigure themselves automatically.

4.1. Proposed Design Overview

To implement the dynamic Honeypot approach, we need the components listed below. We note that the last three components represent our main contribution to the design of a dynamic honeypot.

- An active probing tool, such as Nmap [10]
- A passive fingerprinting tool, such as P0f [11] and Snort.
- A low interaction honeypot used to simulate networks, such as Honeyd.
- A collection of physical honeypots to receive redirected traffic. These high interaction honeypots are considered as a small but representative sample of the most used operating systems on the network.
- A Database, containing hosts' description, and log information.
- A Dynamic honeypot Engine, which interacts with the previous components to dynamically configure Honeyd and generate output (see Figure 1).
- An administrator's interface to configure dynamic honeypot server in real time and view reports.

Dynamic honeypot server starts by collecting information about the hosts available on the networks using active or passive approach.

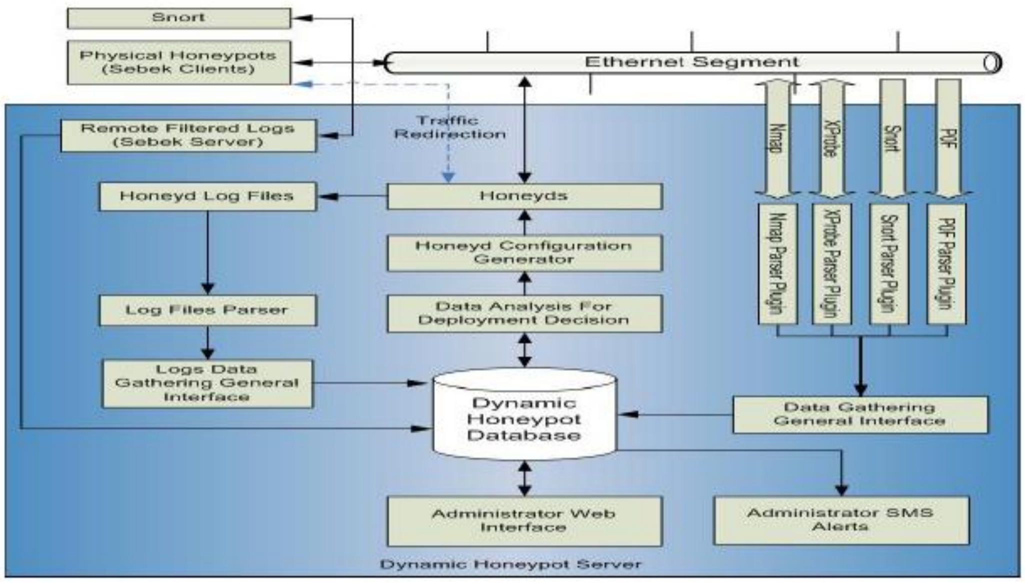
Snort
Physical Honeypots (Sebek Clients)
Ethernet Segment
Remote Filtered Logs (Sebek Server)
Traffic Redirection
Honeyd Log Files
Honeyds
Honeyd Configuration Generator
Log Files Parser
Data Analysis For Deployment Decision
Logs Data Gathering General Interface
Dynamic Honeypot Database
Nmap
XProbe
Snort
P0F
Nmap Parser Plugin
XProbe Parser Plugin
Snort Parser Plugin
P0F Parser Plugin
Data Gathering General Interface
Administrator Web Interface
Administrator SMS Alerts
Dynamic Honeypot Server

Figure 1: Dynamic Honeypot Server

The administrator has the choice of selecting the best data gathering approach to be used based on the network architecture. If the network consists of computers connected through a hub where packet sniffing is feasible, the administrator would run dynamic honeypot server in passive mode avoiding the need for generating probing packets on the shared medium. On the other hand, if network consists of switched network where the hosts connected to layer two switch, passive fingerprinting would not be as reliable as active probing. After having the complete picture of the network including hosts' operating systems and services running, dynamic honeypot server estimates the personalities and services of the fake systems to be deployed and then issues the suitable configuration parameters to honeyd to deploy the systems on the network. The network will now have both real and fake systems running together. An intruder can be detected using the connections that are made to the fake systems which are non production systems and are not supposed to receive traffic from the network. Any attempt to interact with a fake system would be redirected to a physical honeypot permitting the hacker a higher level of interaction. The interaction with the physical honeypot is logged using Honeynet Sebek client and is sent to the dynamic honeypot server which uses sebek server to capture the logs. This approach allows central collection of logs so the administrator will be presented with summarized reports of interactivity on both the fake and physical honeypots. The dynamic honeypot server constantly analyzes the data available in the database and sends an alert (such as an SMS) to the choice of selecting the best data gathering approach to be used based on the network architecture. If the network consists of computers connected through a hub where packet sniffing is feasible, the administrator would run dynamic honeypot server in passive mode avoiding the need for generating probing packets on the shared medium. On the other hand, if network consists of switched network where the hosts connected to

layer two switch, passive fingerprinting would not be as reliable as active probing. After having the complete picture of the network including hosts' operating systems and services running, dynamic honeypot server estimates the personalities and services of the fake systems to be deployed and then issues the suitable configuration parameters to honeyd to deploy the systems on the network. The network will now have both real and fake systems running together. An intruder can be detected using the connections that are made to the fake systems which are non production systems and are not supposed to receive traffic from the network. Any attempt to interact with the administrator in case of a high level of risk a need of immediate attention. An important advantage of using the approach of dynamic honeypot lies in its ability to capture malicious attacks both on the small scale (i.e. from host perspective, physical honeypot logs) and the large scale (i.e. from the network perspective ,virtual honeypot logs). For example, we can identify a worm from repeated scans on a certain port on multiple hosts. This combination of gathered logs permits the administrator to better analyze and classify attacks.
The following illustrates the function of each of the components used as part of the dynamic honeypot server:

4.2. Active Probing

To implement the dynamic honeypot approach, one can actively probe the entire network and request responses from a targeted system in order to determine the OS and the services it uses. There are some drawbacks if active probing is extensively used. This excessive probing consumes additional bandwidth and may cause systems to shutdown. A second issue arises when the targeted systems do not respond when a local firewall with a high security policy is in place. As mentioned before, active probing needs to keep requesting answers from the machines on the network in order to work in a real time fashion. Nmap, for example, is a tool equipped with a database of known operating systems and services. It actively sends packets to the target and illicit responses.

```
TSeq(Class=RI%gcd=<8%SI=<11784E&>2CA4)
T1
(DF=Y%W=2017%ACK=S++%Flags=AS%Ops=MNWNNT)
T2 (Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
T3
(Resp=Y%DF=Y%W=2017%ACK=O%Flags=A%Ops=NNT)
T4 (DF=N%W=0%ACK=O%Flags=R%Ops=)
T5 (DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6 (DF=N%W=0%ACK=O%Flags=R%Ops=)
T7 (DF=N%W=0%ACK=S++%Flags=AR%Ops=)
PU(DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RI
PCK=E%UCK=E%ULEN=134%DAT=E)
```

Figure 2: Nmap's signature

These responses are then compared to known signatures in a database to identify the operating system and services of the remote system. For example, Figure 2 shows Nmap's signature for a Windows NT operating system [12].

4.3. Passive Fingerprinting

An alternative approach is to use passive fingerprinting which is based on sniffing packets from the network to capture network activity, analyze it, and determine the fingerprinting of the system (OS and services). This will not bring the computer down or utilize extra bandwidth. Also the firewall problem will be alleviated since passive fingerprinting will identify the system and any changes in real time can still be captured. Table 1 illustrates the pros and cons of passive and active probing. This concept of sniffing the network packets can introduce a problem when using routed or switched networks and therefore, it is more effective on a shared medium, such as a repeater hub.

Table 1: Active vs. passive probing

Active Probing [Example: Nmap]	Passive Probing [Example: p0f]
Advantages	**Disadvantages**
Able to reach distant routed network	Unable to cross beyond switched networks unless routers in between are reconfigured. More effective on LANs.
Introduces more activity on the network	No or very little traffic on the network
Not able to identify firewalled services	Able to identify firewalled services
Gives a snapshot of a point in time	Continuous real-time data gathering
Capable of identifying all open ports and their services. Assuming no firewall is used	Identifies traffic generating ports only. Ports that do not generate traffic are not detected.
Suitable for switched networks	Suitable for shared networks

P0f is a passive fingerprinting program that compares SYN packets against a set of signatures stored in its database in its attempt to determine the OS type. P0f is totally silent and does not impose traffic on the network; it uses TCP/IP fields to infer the type of OS used. Below, in Figure 3, is a list of all these fields.

```
# wwww:ttt:mmm:D:W:S:N:I:OS Description
#
# wwww - window size
# ttt - time to live
# mmm - maximum segment size
# D - don't fragment flag (0=unset, 1=set)
# W - Window scaling (-1=not present,
   other=value)
# S - SackOK flag (0=unset, 1=set)
# N - Nop flag (0=unset, 1=set)
# I - packet size (-1 = irrelevant)
```

Figure 3: TCP fields used in p0f

An example of how a field is used to determine an OS is the default window size (wwww), (see Table2):

Table 2: Window size for different OSs

Window Size	OS
5840	Linux 2.4.1-14
8760	Solaris 2.6 or 2.7
24820	SunOS 5.8
32120	Linux 2.2.x

Now consider a TCP SYN packet with the field values in Table 3. When written in p0f format we would have 512:124:1460:0:118:0:0:44. This means that this SYN packet probably originated from a Windows 2000 computer.

Table 3: TCP field values

Field	Value
Window size	512
Time To Live	124
Maximum Segment Size	1460
Don't Fragment Flag	0
Window Scaling	118
Sack OK flag	0
NOP	0
Packet Size	44

Our proposed implementation of Dynamic Honeypot uses Snort to passively gather detailed information about the services running on the current network. The network traffic can be filtered using snort rules in order to gather only that information relevant to the current services running on the network.

4.4. Honeyd

This is a low-interaction honeypot. Developed by Niels Provos, Honeyd is Open Source and designed to run primarily on UNIX systems. Honeyd works on the concept of monitoring unused IP space. Anytime it sees a connection attempt to an unused IP, it intercepts the connection and then interacts with the attacker, pretending to be the victim. By default, Honeyd detects and logs any connection to any UDP or TCP port. When an attacker connects to the emulated service, not only does the honeypot detect and log the activity, but it captures all of the attacker's interaction with the emulated service. In the case of the emulated FTP server, we can potentially capture the attackers' login and password, the commands they issue, and perhaps even learn what they are looking for or their identity. Honeyds can be used to simulate networks, and provides an IP stack simulated response of the OS they represent. They do not provide full application layer response such as is the case with a high interaction honeypot. They can redirect network activities that require a high level of interaction to a physical honeypot. For example, they can redirect DNS requests to a proper name server. If someone uses active fingerprinting measures to determine the OS type of the honeyd most honeyds respond with the IP stack of whatever OS they are configured to mimic. This is accomplished by spoofing the needed replies. The Database The database contains information about the real and virtual systems; it contains mainly 5 tables as shown in (Table 4).

1- IP Information Table: It contains the description of the systems having a certain IP address. The description of the system includes information about the operating system, the time the data was collected, in addition to a flag indicating whether the IP is used by a real system or a virtual system. This table also includes a code for the probing tool used, since some probing tools are more reliable than others.
2- Ports Table: Contains more detailed description of a certain IP address. The data includes a list of all ports that are open and the services. In addition there are entries for the probing tool used and the time the data was collected.
3- Operating Systems Table: Contains a unique ID given to each operating system together with a description of that operating system.
4- Services Scripts Table: Contains information about the honeyd script and the service it is used to simulate.
5- HoneydLog Table: In case a honeyd is accessed, the log data is filled into this table. The table contains information about the packets received by the honeyds, including the source and destination addresses and their respective ports. In addition there are entries for information about time and possibly some information about the intruder's operating system or his scanning tool.
6. Sebek Logs Table: This is used as part of Sebek Server to capture interactivity with physical honeypots.

Table 4: Dynamic Honeypot DB Tables

Dynamic Honeypot DB
IP Information
Ports
Operating Systems
Services Scripts
HoneydLog
Sebek Logs

4.5. Dynamic Honeypot Engine

This module performs 4 main functions:

1. Gather the needed information required for honeyd deployment and perform necessary calculations:
   In order to be able to deploy the right combination of low interaction honeypots, with the right characteristics, we need to gather information about the real systems. This is done using the IP Information table. The calculations are made such that the virtual systems reflect the ratio of the real systems' operating systems. Reflecting the ratio means that the number of used IPs running a certain OS to the total number of used IPs must match the number of deployed virtual honeypots simulating the same OS to the total number of the deployed virtual honeypots. For example if we have 5 used IPs with 2 Windows NT and 3 Red Hat Linux, and we have 10 free IPs these should be filled with 4 virtual Windows NT and 6 virtual Red Hat Linux

2. Issue configuration commands for deployment of honeyds: Now that the engine knows what low interaction honeypots should be deployed, configuration commands are sent to honeyd to deploy the virtual systems.

3. Collect logs and generate output reports: The administrator is alerted if there is any attack. The data in the HoneydLog should also be reported, this allows the administrator to view how often honeyds are accessed and the corresponding details of attack.

4. Delete Expired database entries: This functionality basically checks for the time field for entries in the database and compares them with the default expiration time or to the custom period set by the administrator.

The network on which tests were carried on was composed. The computers run various Windows OS as well as Linux systems. We have tested our system using the Normal scan option of NMAP whereby an average scan takes 5 seconds for Windows and 60 seconds for Linux. This option runs multiple threads to communicate with multiple ports at the same time thus a large amount of data is generated and received by the Dynamic Honeypot system. The tests shown in table 5 illustrate the time it takes to completely probe a class C network. However it should be taken into account that the time it would take to scan a certain computer may vary from test to test because the targeted computers may be communicating over the network or the window size may not accommodate the incoming traffic which will result in retransmissions. Depending on the prior case the data received and sent by the dynamic honeypot may reach an extremely high level almost 1 million bytes (see figure 4). The fact that we have used a time interval (D) between each time we issue a set of scans to run in parallel is to mitigate the case when we may have all the scans running at the same time, thus reducing I/O traffic. Thereafter this interval allows the system to complete some scans before issuing others, it is also configurable and depends on the system's performance under the network's traffic load. Another Test has been carried out to decrease the overhead induced on the network and the dynamic honeypot system; whereby Nmap's polite mode was used, which serializes the port scans with a delay of 400ms between each scan. This allows the system to run multiple instances of this scan on different IPs which increases the overall convergence time, yet it would decrease the I/O traffic, to a large extent, of the dynamic honeypot system.

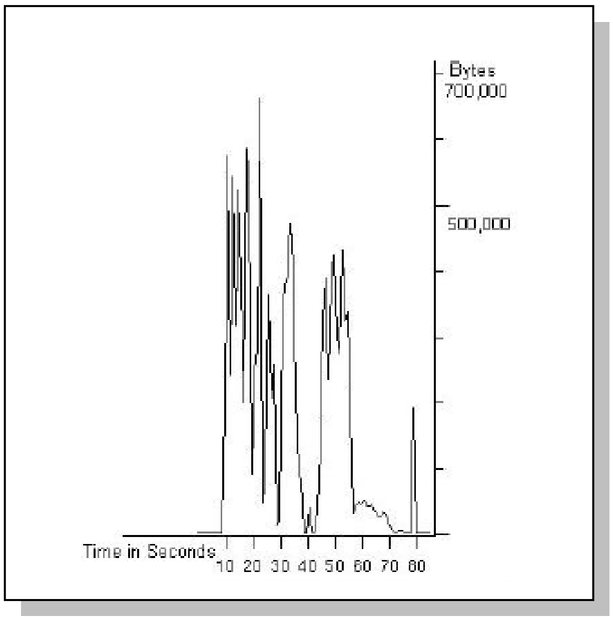

Figure 4: Normal Mode for N = 70 and D = 30

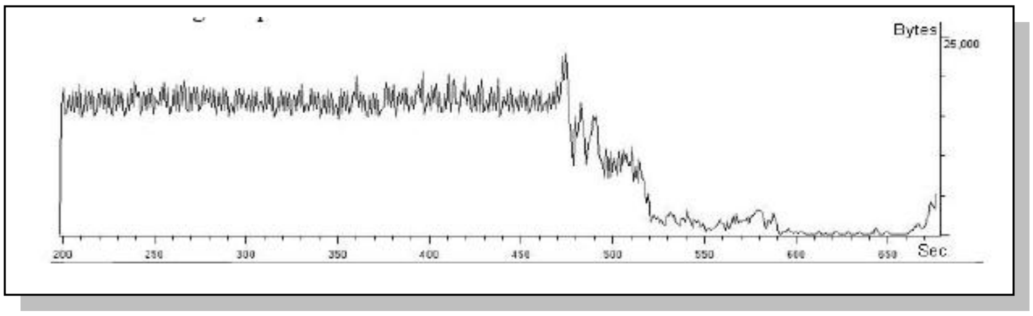

Figure 5: Polite scanning with N = 60 and D = 1200 (Parallel Scan I)

Table 5: Test results using Nmap's Normal mode

N=Number of Parallel Scans every D (seconds)	Time to converge (seconds)	Average Mbits/second
70	70.8	1.487
60	95.6	1.013
50	99.33	0.963
40	94.63	1.032
30	160.58	0.5802
20	165.33	0.614

We performed a number of parallel scans, the case when N is larger than the available devices on the network the system would finish deploying the honeyds within 1300s (taking into account the Linux scan convergence time which is about 1100s, opposed to windows which took almost 650s). Figures 5 shows the results of using the polite option with D=1200 and N=60 scans at a time. Figure 5 shows the system at the first scan iteration with two scan times the first is comprised of 60 scans. The traffic is significantly less and almost constant. Convergence of data to the system as well as deployment is at most 20 minutes for this case and can increase as N or D.

5. Conclusion

In this paper we introduced the of dynamic honeypots and our proposed design for implementing such systems in real network environments. Our design includes a dynamic honeypot engine that integrates data collected from passive fingerprinting tools such as p0f and active probing tools such as Nmap to dynamically configure Honeyds. Further improvements include analyzing the gathered data to generate suitable countermeasures.

6. REFERENCES

[1] CSI/FBI Computer Crime and Security Survey 2003. http://www.gocsi.com/forms/fbi/pdf.jhtml

[2] Paul Innella and Oba McMillan. An Introduction to Intrusion Detection Systems. http://www.securityfocus.com/infocus/1520

[3] Snort Website. http://www.snort.org/about.html

[4] Lance Spitzner. Definitions and Value of Honeypots. http://www.trackinghackers.com/papers/honeypots.html

[5] Specter Website. http://www.specter.com/

[6] Niels Provos. A Virtual Honeypot Framework. http://citi.umach.edu/techreports/reports/cititr-03-1.pdf

[7] KFSensor website. http://www.keyfocus.net/kfsensor/overview. php

[8] Know Your Enemy: Honeynets. http://www.honeynet.org/papers/honeynet/

[9] Lance Spitzner. Dynamic Honeypots. http://www.securityfocus.com/infocus/1720

[10] Mark Wolfgang. Host discovery with Nmap. http://moonpie.org/writings/discovery.pdf

[11] Michal Zalewski and William Stearns. Passive OS Fingerprinting Tool. http://www.stearns.org/p0f/README

[12] Contributed by Vilius beneti@sc.ktu.lt