Honeyd: An Unpopular Network Defense Framework

Emmanuel Ekunsumi
Department of Computer Science
University of Manitoba
umekunsu@myumanitoba.ca

ABSTRACT

Although numerous computer networks run some sort of security mechanism at their connection to the Internet, they keep on being compromised and exploited by hackers. Substantial Enterprise Networks, for example, the network for a noteworthy college, are extremely welcoming focuses to hackers who are looking to exploit networks. These substantial Enterprise Networks may comprise of numerous machines running various operating systems. Networks like these regularly have huge capacity abilities and fast/high transmission capacity connection to the Internet. Because of the necessities for Academic Freedom, system administrators are limited in what requirements they can place on the users on these networks. The high data transmission uses on these networks make it extremely hard to recognize malicious movement inside the undertaking network. This paper presents Honeyd, a framework written by Neil Provos for creating virtual honeypots that simulates virtual computer systems at the network level. The simulated computer systems seem to keep running on unallocated network addresses. To mislead network fingerprinting tools, Honeyd recreates the networking stack of various operating systems and can give subjective routing topologies and services for a large number of virtual systems. This paper talks about Honeyd's design and shows how the Honeyd framework helps in numerous regions of system security, e.g. recognizing and disabling worms, distracting adversaries, or keeping the spread of spam emails to a bare minimum.

Keywords

Honeyd; Honeynet; Honeypots; Intrusion detection.

1. INTRODUCTION

An ordinary honeynet comprises of numerous honeypots interlinked together lastly to the Internet. This setup is powerful, works adequately, and makes logging and observation basic.

On the drawback, while this choice might be reasonable for enterprises or expansive associations, it can be extremely burdened to set up for small businesses, as it obliges them to purchase a server rack or two, look after them, and keep running up the electric bill. Keep in mind that much of the time, more honeypots = better results. There is likewise some danger of malware spilling out of a traded off honeypot onto an authentic computer and devastating it (if the honeypot isn't totally segregated from your inward network, that is).

Conference'10, Month 1–2, 2010, City, State, Country.

The most ideal approach to take care of this issue is with virtual honeypots, which is fundamentally a daemon running on one or a few computers that creates virtual honeypot computers and spots them on the network. Rather than buying and set up numerous physical computers, you now just need one computer which can create and have a large number of virtual honeypots however you see fit.

This paper describes the design and implementation of Honeyd, a framework for virtual honeypots that simulates computer systems at the network level. Honeyd is an open source application that tries to meet that objective. Every honeypot is a design document that you stack and send. These honeypots are totally client adaptable through a basic content manager, where you may characterize such qualities including its base working system, port conduct, and the sky is the limit from there. Honeyd can reproduce an entire slew of port administrations for every individual honeypot, for example, HTTP, FTP, telnet, rsh, SMTP, and a lot more.

When would virtual honeypots or honeynets be used in the real world? Here is a case situation: a small organization has three servers loaded with vital information that it needs to secure. A fourth server on the same network runs Honeyd with a couple of hundred conveyed honeypots. All servers have an intrusion detection system installed. The odds of an adversary hitting one of the four legitimate computers out of two hundred are exceptionally thin.

At the point when a honeypot is attacked, all network activity and time frames are logged alongside the attacker's IP address and port listings, permitting the organization to distinguish the nearness of an intruder before any genuine harm is finished. It's the ideal trap.

Whatever is left of this paper is composed as follows. Section 2 presents foundation information on honeypots. In Section 3, we talk about the outline and usage of Honeyd. Section 3 also outlines an assessment of the Honeyd structure in which we break down the execution of Honeyd and confirm that fingerprinting and network mapping tools are deceived to report the predefined system arrangements. I depict how Honeyd can be applied to enhance system security in Section 4 and conclude in Section 5.

2. HONEYPOTS

This section presents background knowledge on honeypots. We then provide motivation for its use by comparing honeypots to firewalls and Intrusion Detection Systems (IDS). A firewall can be considered as a "traffic cop" for the network [3]. They are used to control the flow of traffic between the local network and the Internet. An IDS is used to detect and alert on possible malicious events within a network. They may also perform statistical and anomaly analysis of network traffic to detect malicious intrusions. When malicious activity is detected they can notify the system

administrator [4]. Both firewalls and Intrusion Detection Systems are tools used by system administrators to protect the security of their networks

A honeypot on the other hand is a closely monitored computing resource that we intend to be probed, attacked, or compromised. Honeypots can run any operating system and any number of services. The configured services determine the vectors available to an adversary for compromising or probing the system [1]. There are two types of honeypots; high-interaction honeypots and low-interaction honeypots. A high-interaction honeypot simulates all parts of an operating system while a low-interaction honeypots simulates only some parts, for example the network stack [5]. Low-interaction honeypots are more restricted, yet they are helpful to accumulate data at a larger amount, e.g., find out about network probes or worm action. They can likewise be utilized to investigate spammers or for dynamic countermeasures against worms; see Section 4.

Honeypots can also be grouped into physical and virtual honeypots. A physical honeypot is a real machine on the network with its own IP address. A virtual honeypot is simulated by another machine that responds to network traffic sent to the virtual honeypot. Physical honeypots are often high-interaction, so permitting the system to be compromised totally, they are costly to introduce and keep up. For large address spaces, it is impractical or impossible to deploy a physical honeypot for each IP address [1]. In that case, we need to deploy virtual honeypots.

The utilization of IDS and firewalls give a level of security assurance to the system administrator. However, there are perceived setbacks with the utilization of an IDS and firewalls to secure a network. The weaknesses connected with a firewall include the following:

1. The firewall cannot protect against attacks that bypass it, such as a dial–in or dial-out capability.

2. The firewall at the network interface does not protect against internal threats.

3. The firewall cannot protect against the transfer of virus–laden files and programs [6].

The utilization of IDS as a network security device also leads to shortcomings. It has been guessed that now and again an IDS neglects to give an extra level of security to a network and just expands the many-sided quality of the security administration issue. Shortcomings associated with an IDS include a high level of false positive and false negative alerts [7]. Because a honeypot has no production value, any attempt to contact it is suspicious. Consequently, forensic analysis of data collected from honeypots is less likely to lead to false positives than data collected by IDS [1].

I propose that the use of virtual Honeypots within a network can provide an additional layer of network security. The Honeypots can serve as a compliment to the use of the firewall and IDS and help to overcome some of the shortcomings that are inherent to these systems.

3. HONEYD

In this section, I present Honeyd, a lightweight framework for creating virtual honeypots. The framework allows us to create thousands of IP addresses with virtual machines and corresponding network services. We start by describing Honeyd's design, architecture and configuration, and then present an evaluation of Honeyd's ability to create virtual network topologies.

Figure 1 demonstrates an applied review Honeyd's operation. A main machine blocks network movement sent to the IP locations of configured honeypots and mimics their reactions. Before we portray Honeyd's architecture, we clarify how network packets for virtual honeypots come to the Honeyd host.

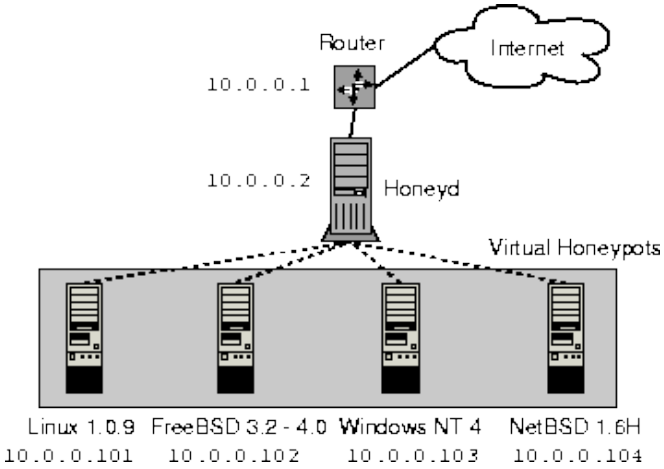

Figure 1. A conceptual overview of Honeyd's operations.

3.1 Design

Honeyd must have the capacity to handle virtual honeypots on various IP addresses at the same time, keeping in mind the end goal to populate the network with various virtual honeypots simulating different operating systems and services. To build the authenticity of our simulation, the framework must have the capacity to reproduce discretionary network topologies. To simulate address spaces that are topologically scattered and for burden sharing, the framework additionally needs to bolster network tunneling.

Honeyd is designed to answer to network packets whose destination IP address fits in with one of the simulated honeypots. For Honeyd, to get the right packets, the network needs to be configured properly. To do this, we can use a Proxy ARP [8].

Let A be the IP address of our router and B the IP address of the Honeyd host. In the simplest case, the IP addresses of virtual honeypots lie within our local network. We denote them $V_1, \ldots, V_n$. When an adversary sends a packet from the Internet to honeypot V_i, router A receives and attempts to forward the packet. The router queries its routing table to find the forwarding address for V_i. There are three possible outcomes: the router drops the packet because there is no route to V_i, router A forwards the packet to another router, or V_i lies in local network range of the router and thus is directly reachable by A.

To direct traffic for V_i to B, we can use the following two methods. The easiest way is to configure routing entries for V_i with $1 \leq i \leq n$ that point to B. In that case, the router forwards packets for our virtual honeypots directly to the Honeyd host. On the other hand, if no special route has been configured, the router ARPs to determine the MAC address of the virtual honeypot. As there is no corresponding physical machine, the ARP requests go unanswered and the router drops the packet after a few retries. We

configure the Honeyd host to reply to ARP requests for V_i with its own MAC addresses. This is called Proxy ARP and allows the router to send packets for V_i to B's MAC address.

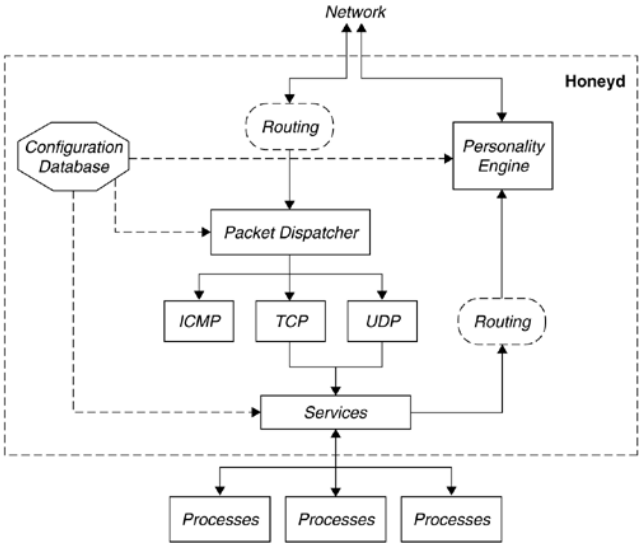

Figure 2. An overview of Honeyd's architecture.

3.2 Architecture

Approaching packets are processed by the central packet dispatcher; see figure 2. It first checks the length of an IP packet and confirms the packet's checksum. Honeyd knows about the three noteworthy Internet protocols: ICMP, TCP and UDP. Packets for other protocols are logged and disposed of.

Before it can process a packet, the dispatcher must query the configuration database to discover a honeypot configuration that compares to the destination IP address [1]. If no particular configuration exists, a default layout is utilized. Given a configuration, the packet and relating configuration is given to the protocol specific handler.

Before a packet is sent to the network, it is processed by the personality engine. The personality engine alters the packet's content so it seems to originate from the network stack of the configured operating system. Honeyd achieves this by reversing the databases used by the fingerprinting tools. When a honeypot needs to sends a network packet, it is modified by honeyd to match the fingerprint that corresponds to the configured operating system in the database.

Adversaries commonly run fingerprinting tools like Xprobe [9] or Nmap [10] to gather information about a target system. The framework uses Nmap's fingerprint database as its reference for a personality's TCP and UCP behavior; Xprobe's fingerprint database is used as reference for a personality's ICMP behavior.

To change the characteristics of a honeypot's network stack, honeyd has to analyse every Nmap fingerprint. Each Nmap fingerprint has the following format; the first line is used to assign the personality name and the lines after the name describe the results for nine different tests that Nmap performs to determine the operating system of a remote host. The first test determines how the network stack of the remote operating system creates the initial sequence number (ISN) for TCP SYN segments. Nmap indicates the difficulty of predicting ISNs in the Class field. Predictable ISNs post a security problem because they allow an adversary to spoof connections [14]. The gcd and SI field provide more detailed information about the ISN distribution. The first test also determines how IP identification numbers and TCP timestamps are generated [1].

Nmap's fingerprinting is mostly concerned with an operating system's TCP implementation. TCP is a stateful, connection-oriented protocol that provides error recovery and congestion control [13]. TCP also supports additional options, not all of which implemented by all operating systems. The size of the advertised receiver windows varies between implementations and is used by Nmap as part of the fingerprint.

When the framework sends a packet for a newly established TCP connection, it uses the Nmap fingerprint to see the initial window size. After a connection has been established, the framework adjusts the window size according to the amount of buffered data.

If TCP options present in the fingerprint have been negotiated during connection establishment, then Honeyd inserts them into the response packet. The framework uses the fingerprint to determine the frequency with which TCP timestamps are updated. For most operating systems, the update frequency is 2 Hz.

Generating the correct distribution of initial sequence numbers is tricky. Nmap obtains six ISN samples and analyzes their consecutive differences. Nmap recognizes several ISN generation types: constant differences, differences that are multiples of a constant, completely random differences, time dependent and random increments. To differentiate between the latter two cases, Nmap calculates the greatest common divisor (gcd) and standard deviation for the collected differences.

The framework keeps track of the last ISN that was generated by each honeypot and its generation time. For new TCP connection requests, Honeyd uses a formula that approximates the distribution described by the fingerprint's gcd and standard deviation. In this way, the generated ISNs match the generation class that Nmap expects for the particular operating system [1].

3.3 Configuration

A virtual honeypot is configured with a template, a reference for a completely configured computer system. New templates are created with the create command. For example, typing "create newTemp" into your terminal creates a new template called newTemp.

The set command assigns a personality from the Nmap fingerprint file to a template. The set command additionally characterizes the default behavior for the supported network protocols. The default behavior is one of the accompanying qualities: block, reset, or open. Block implies that all packets for the predetermined protocol are dropped by default. Reset shows that all ports are closed by default. Open implies that they are all open by default.

The add command is used to specify the services that are remotely accessible. In addition to the template name, we have to indicate the protocol, port and the command to execute for every service. Instead of specifying a service, Honeyd also recognizes the keyword proxy that allows us to forward network connections to a different host. The framework expands the following four variables for both the service and the proxy statement: $ipsrc, $ipdst, $sport, and $dport [1]. Finally, the bind command assigns a template to an IP address. If no template is assigned to an IP address, we use the default template.

3.4 Evaluation

This subsection presents an evaluation of Honeyd's ability to create virtual network topologies and to mimic different network stacks as well as its performance.

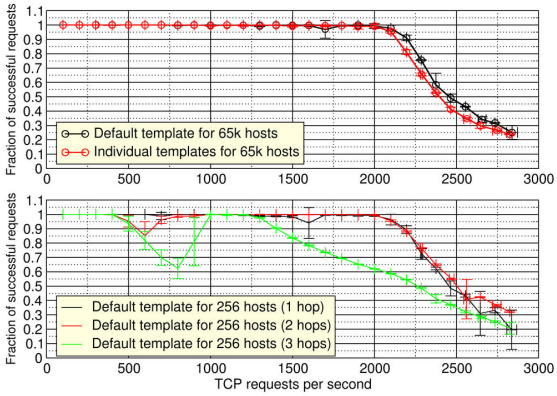

Figure 3. An overview of Honeyd's performance.

Figure 10 shows the results from Neil Provos's TCP performance measurement [1]. The measurements is repeated at least five times and shows the average result including standard deviation. The upper graph demonstrates the performance when using the default template for all honeypots compared with the performance when utilizing an individual template for every honeypot. Performance diminishes somewhat when each of the 65K honeypots is configured individually. In both cases, Honeyd can manage more than two thousand TCP transactions for each second. The lower diagram demonstrates the performance for reaching honeypots at various levels of the routing topology. The performance diminishes most noticeably for honeypots that are three hops away from the sender.

The measurements show that a 1.1 GHz Pentium III can simulate thousands of virtual honeypots. However, the performance depends on the complexity and number of simulated services available for each honeypot.

4. APPLICATIONS

In this section, I describe how the Honeyd framework can be used in different areas of system security.

4.1 Detecting and Countering Worms

Moore et al. show that containing worms is not practical on an Internet scale unless a large fraction of the Internet cooperates in the containment effort [11]. However, with the Honeyd framework, it is possible to effectively counter worm propagation by immunizing infected hosts that contact our virtual honeypots. According to Moore et al. [11], we can model the effect of immunization on worm propagation by using the classic SIR epidemic model [12]. The model states that the number of newly infected hosts increases linearly with the product of infected hosts, fraction of susceptible hosts and contact rate. The immunization is represented by a decrease in new infections that is linear in the number of infected hosts:

$$\frac{ds}{dt} = -\beta i(t)s(t)$$

$$\frac{di}{dt} = \beta i(t)s(t) - \gamma i(t)$$

$$\frac{dr}{dt} = \gamma i(t),$$

where at time t, i(t) is the fraction of infected hosts, s(t) the fraction of susceptible hosts and r(t) the fraction of immunized hosts. The propagation speed of the worm is characterized by the contact rate β and the immunization rate is represented by γ.

According to studies by Niels Provos, if we use 360,000 susceptible machines in a 2^{32} address space and set the initial worm seed to 150 infected machines. Each worm launches 50 probes per second and we assume that the immunization of an an infected machine takes one second after it has contacted a honeypot. The simulation measures the effectiveness of using active immunization by virtual honeypots. The honeypots start working after a time delay. The time delay represents the time that is required to detect the worm and install the immunization code. We expect that immunization code can be prepared before a vulnerability is actively exploited. If we wait for an hour, all vulnerable machines on the Internet will be infected. Our chances are better if we start the honeypots after twenty minutes. In that case, a deployment of about 262,000 honeypots is capable of stopping the worm from spreading to all susceptible hosts. Ideally, we detect new worms automatically and immunize infected machines when a new worm has been detected [1].

4.2 Spam Prevention

In general, spammers misuse two Internet services: proxy servers [10] and open mail relays. Open proxies are frequently used to connect to different proxies or to submit spam email to open mail relays. Spammers can utilize open proxies to anonymize their character to avert following the spam back to its root. An open mail relay acknowledges email from any sender location to any beneficiary location. By sending spam email to open mail relays, a spammer causes the mail relay to convey the spam on his behalf.

To see how spammers operate we utilize the Honeyd structure to instrument networks with open proxy servers and open mail relays. We make use of Honeyd's GRE tunneling capacities and tunnel a few C-class networks to a central Honeyd host.

We populate our network space with randomly picked IP addresses and a random selection of services. Some virtual hosts may run an open proxy and others may simply run an open mail relay or a mix of both.

At the point when a spammer endeavors to send spam email by means of an open proxy or an open mail relay, the email is consequently diverted to a spam trap. The spam trap then presents the gathered spam to a cooperative spam channel.

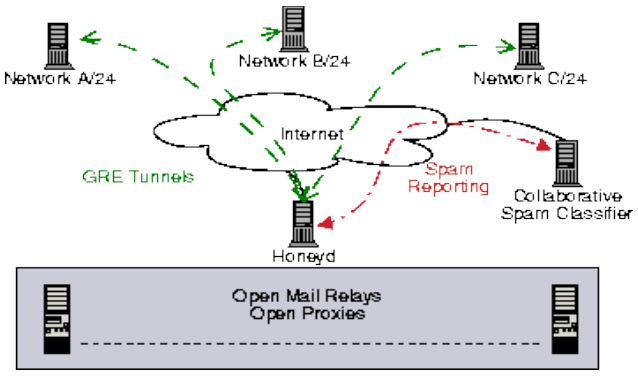

Figure 4. An overview of how Honeyd handles spam.

5. CONCLUSIONS

Honeyd is a framework for making virtual honeypots. Honeyd imitates the network stack behavior of operating systems to bamboozle fingerprinting tools like Nmap and Xprobe.

I gave an outline of Honeyd's design and architecture and demonstrated how Honeyd's personality engine can alter packets to match the fingerprints of other operating systems.

The performance measurements demonstrated that a solitary 1.1 GHz Pentium III can be used to create thousands of virtual honeypots and can manage more than two thousand TCP exchanges for every second. The assessment demonstrated that Honeyd is viable in effectively tricks fingerprinting tools.

I also indicated how the Honeyd framework can be used to help in various ranges of system security, e.g., worm detection, worm countermeasures, or spam prevention.

I trust that Honeyd is basically an extraordinary all-around honeypot program. It can mimic any operating system or port service you toss at it, it has a lot of good elements, and is easily obtainable from various repository sources. It is an intense, adaptable, and cost-effective alternative for physical, hardware-based honeypots, which makes it ideal for growing system administrators, specialists, or truly jumpy clients.

6. ACKNOWLEDGMENTS

I thank my instructor, Dr. Noman Mohammed for giving me his approval to do my research on this topic.

7. REFERENCES

[1] Niels Provos. *A Virtual Honeypot Framework.* Google, Inc. niels@google.com

[2] John Levine, Richard LaBella, Henry Owen, Didier Contis, Brian Culver, 2003. *The use of Honeynets to Detect Exploited Systems Across Large Enterprise Networks* Proceedings of the 2003 IEEE Workshop on Information Assurance United States Military Academy, West Point, NY

[3] E. Skoudis, Counter Hack, Upper Saddle River, NJ: Prentice Hall PTR, 2002, p. 47.

[4] S. Northcut, L. Zeltser, S. Winters, K. Kent Fredericks, R. Ritchey, Inside Network Perimeter Security. Indianapolis, In: New Riders, 2003, p.5.

[5] Lance Spitzner. Honeypots: Tracking Hackers. Addison Wesley Professional, September 2002.

[6] W. Stallings, Network Security Essentials, Upper Saddle River, NJ: Prentice Hall PTR, 2000, p. 322.

[7] R. Stiennon, M. Easley, Intrusion Prevention Will Replace Intrusion Detection, Gartner Research Notes, 30 August 2002, available at http://www.gartner.com/reprints/intruvert/109596.html, Dec 2002.

[8] Smoot Carl-Mitchell and John S. Quarterman. Using ARP to Implement Transparent Subnet Gateways. RFC 1027, October 1987.

[9] Ofir Arkin and Fyodor Yarochkin. Xprobe v2.0: A “Fuzzy” Approach to Remote Active Operating System Fingerprinting. www.xprobe2.org, August 2002

[10] S. Glassman. A Caching Relay for the World Wide Web. In Proceedings of the First International World Wide Web Conference, pages 69–76, May 1994.

[11] David Moore, Colleen Shannon, Geoffrey Voelker, and Stefan Savage. Internet Quarantine: Requirements for Containing Self-Propagating Code. In Proceedings of the 2003 IEEE Infocom Conference, April 2003.

[12] Herbert W. Hethcote. The Mathematics of Infectious Diseases. SIAM Review, 42(4):599–653, 2000.

[13] Jon Postel. Transmission Control Protocol. RFC 793, September 1981.

[14] Steven M. Bellovin. Security problems in the TCP/IP protocol suite. Computer Communications Review, 19:2:32–48, 1989.