Abstract

A virtual honeypot is a fake network designed by computer experts to catch hackers and examine their methods of attack. It is tailored to resemble an authentic network, and is an emerging form of information technology security that actually invites hackers to perform illegal activities such as accessing computer systems for which they are unauthorized. Virtual honeypots are without production value and their sole purpose is to wait for hackers to unknowingly access them and, while doing so, for IT security professionals to learn more about hackers' activities, if not discover their identities. Honeypots are unique when it comes to IT security in that hackers can't cause any real harm to simulated networks. The flexibility and the simplicity of virtual honeypots are considered to be their primary advantages. A software program that is designed to appear to be a real functioning network but is actually a decoy built specifically to be probed and attacked by malicious users. In contrast to hotspot, which is typically a hardware device that lures users into its trap, virtual honey pot uses software to emulate a network

1. Introduction

Internet security is increasing in importance as more and more business is conducted there. Yet, despite decades of research and experience, we are still unable to make secure computer systems or even measure their security.

As a result, exploitation of newly discovered vulnerabilities often catches us by surprise. Exploit automation and massive global scanning for vulnerabilities enable adversaries to compromise computer systems shortly after vulnerabilities become known.

One way to get early warnings of new vulnerabilities is to install and monitor computer systems on a network that we expect to be broken into. Every attempt to contact these systems via the network is suspect. We call such a system a **honeypot.** If a honeypot is compromised, we study the vulnerability that was used to compromise it. A honeypot may run any operating system and any number of services. The configured services determine the vectors an adversary may choose to compromise the system.

A physical honeypot is a real machine with its own IP address. A virtual honeypot is a simulated machine with modeled behaviors, one of which is the ability to respond to network traffic. Multiple virtual honeypots can be simulated on a single system.[1]

Virtual honeypots are attractive because they require fewer computer systems, which reduces maintenance costs. Using virtual honeypots, it is possible to populate a network with hosts running numerous operating systems. To convince adversaries that a virtual honeypot is running a given operating system, we need to simulate the TCP/IP stack of the target operating system carefully, in order to deceive TCP/IP stack fingerprinting tools like **Xprobe** or **Nmap.**

2. Honeypot and its types

In computer terminology, a **honeypot** is a computer security mechanism set to detect, deflect, or, in some manner, counteract attempts at unauthorized use of information systems. Generally, a honeypot consists of data (for example, in a network site) that appears to be a legitimate part of the site but is actually isolated and monitored, and that seems to contain information or a resource of value to attackers, which are then blocked. This is similar to the police baiting a criminal and then conducting undercover surveillance, and finally punishing the criminal. [2,6]

Honeypots can be classified based on their deployment (use/action) and based on their level of involvement

Production honeypots are easy to use, capture only limited information, and are used primarily by companies or corporations. Production honeypots are placed inside the production network with other production servers by an organization to improve their overall state of security. Normally, production honeypots are low-interaction honeypots, which are easier to deploy. They give less information about the attacks or attackers than research honeypots.

Research honeypots are run to gather information about the motives and tactics of the Black hat community targeting different networks. These honeypots do not add direct value to a specific organization; instead, they are used to research the threats that organizations face and to learn how to better protect against those threats. Research honeypots are complex to deploy and maintain, capture extensive information, and are used primarily by research, military, or government organizations.

Based on design criteria, **Pure honeypots** are full-fledged production systems. The activities of the attacker are monitored by using a casual tap that has been installed on the honeypot's link to the network. No other software needs to be installed. Even though a pure honeypot is useful, stealthiness of the defense mechanisms can be ensured by a more controlled mechanism.

High-interaction honeypots imitate the activities of the production systems that host a variety of services and, therefore, an attacker may be allowed a lot of services to waste his time. By employing virtual machines, multiple honeypots can be hosted on a single physical machine. Therefore, even if the honeypot is compromised, it can be restored more quickly. In

general, high-interaction honeypots provide more security by being difficult to detect, but they are expensive to maintain. If virtual machines are not available, one physical computer must be maintained for each honeypot, which can be exorbitantly expensive. Example: Honeynet.

Low-interaction honeypots simulate only the services frequently requested by attackers. Since they consume relatively few resources, multiple virtual machines can easily be hosted on one physical system, the virtual systems have a short response time, and less code is required, reducing the complexity of the virtual system's security. Example: Honeyd.

High interaction versus low interaction

Traditionally, tools used for information security were primarily defensive. Tools like firewalls, Intrusion Detection Systems and encryption are used defensively to protect one's resources. With this strategy the attacker always has the initiative. Honeynets tend to change this. The purpose of honeynets is to gather information on threats ([Hon06]). Honeynets are a type of network architecture [Hon06]. This architecture creates a highly controlled network, in which you can control and monitor all the network activity. You setup target systems (honeypots) to attract attackers to your systems. All the actions of the attackers which they perform on the honeypots are captured and stored for analysis. A honeypot is a trap set to detect, deflect or in some manner counteract attempts at unauthorized use of information systems. It consists of a machine that is not supposed to receive any legitimate traffic and, thus, any traffic flood destined to this honeypot is most probably an ongoing attack and can be analyzed to reveal vulnerabilities targeted by attackers. Honeypots divided as Production and research Production Honeypots can be divided into two categories according to [DVK+06]: high interaction and low interaction honeypots.

High interaction:

- Provide Actual Operating Systems
- Extensive risk
- Learn extensive amounts of information
- Log every packet that enters and leave honeypot

High interaction honeypots are full fledged production like systems that host a full suite of services and allow an attacker a lot of latitude during his visit.

- Low interaction Provide simulated Services

- No operating system for attacker to access.

Information limited to transactional information and attackers activities with simulated services.

Advantages of low interaction honeypots

- Good starting point
- Easy to install, configure, deploy and maintain
- Introduce a low limited risk
- Logging and analyzing is simple

Only transactional information are available, no information about the attacks themselves,(e.g. time and date of an attack, protocol, source and destination IP)

Disadvantages of low-interaction honeypots

- No real interaction for an attacker possible
- Very limited logging abilities
- Can only capture known attacks
- Easily detectable by a skilled attacker

Low interaction honeypots have only implemented the services you think the attacker will be interested in. Advantages of low interaction honeypots are the response time, which is quicker and it is possible to add more security to it. The disadvantage is however that an attacker can easily detect an incompletely implemented service. Thus he will know he has hit a honeypot [DVK+06]. There is another type of honeypot which can be categorized in a new category: the virtual honeypot. This is actually a high interaction honeypot, running with other honeypots or production servers on one physical machine with virtualization. This is a rather new concept and has not been researched much. The biggest risk of a virtual honeypot is the host being compromised. We would like to compare them to regular high interaction and low interaction honeypots.

3. Virtual honeypot and honeywalls

A virtual honeypot is software that emulates a vulnerable system or network to attract intruders and study their behavior.

A virtual honeypot is a fake network designed by computer experts to catch hackers and examine their methods of attack. It is tailored to resemble an authentic network, and is an emerging form of information technology security that actually invites hackers to perform illegal activities such as accessing computer systems for which they are unauthorized.

Virtual honeypots are without production value and their sole purpose is to wait for hackers to unknowingly access them and, while doing so, for IT security professionals to learn more about hackers' activities, if not discover their identities.

Honeypots are unique when it comes to IT security in that hackers can't cause any real harm to simulated networks. The flexibility and the simplicity of virtual honeypots are considered to be their primary advantages.

Virtual honeypot is a software program that is designed to appear to be a real functioning network but is actually a decoy built specifically to be probed and attacked by malicious users. It is typically a hardware device that lures users into its trap , a virtual honeypot uses software to emulate a network.

Honeypot owners can gather information about intruders and their actions that can help identify network vulnerabilities and take actions to protect weak points. The information can also be used to determine what IP addresses and requests should be blocked. Attacker information can also be turned over to the authorities.

Virtual honeypots contrast with hardware-based honeypots, which are dedicated computers, networks or network segments designed to serve the same purpose. Virtual honeypots can be thought of as virtual machines (VMs) which may exist in multiple configurations on a single computer or appliance to emulate various systems and vulnerabilities.

Virtual honeypots are cheaper to deploy and more secure than hardware-based systems. In some cases, for example, real honeypots have been infiltrated by intruders who were able to use them to attack the corporate network. However, because a virtual honeypot is an emulator, it doesn't function exactly as a real system does and hackers may be able to pick up on cues that indicate the difference.

Honeynets: Two or more honeypots on a network form a **honeynet.** Typically, a honeynet is used for monitoring a larger and/or more diverse network in which one honeypot may not be sufficient. Honeynets and honeypots are usually implemented as parts of larger network intrusion detection systems. A **honeyfarm** is a centralized collection of honeypots and analysis tools

"A honeynet is a network of high interaction honeypots that simulates a production network and configured such that all activity is monitored, recorded and in a degree, discreetly regulated."

Virtual honeynet: Virtual honeynets are one type of honeynet, specifically honeynets that run multiple operating systems on the same physical computer.

This is done using virtualization software such as VMware or User-Mode Linux

Honeywall: With Honeywall, you have a device to mitigate risk. If a cracker compromises your honeypot, you want to contain him within that honeynet. It’s a kind of intrusion-prevention system that prevents outgoing attacks.

4. **Physical versus virtual honeypots**

This section presents background information on honeypots and our terminology. We provide motivation for their use by comparing honeypots to Network Intrusion Detection Systems (NIDS) .The amount of useful information provided by NIDS is decreasing in the face of ever more sophisticated evasion techniques and an increasing number of protocols that employ encryption to protect network traffic from eavesdroppers. NIDS also suffer from high false positive rates that decrease their usefulness even further. Honeypots can help with some of these problems.

A **honeypot** is a closely monitored computing resource that we intend to be probed, attacked, or compromised. The value of a honeypot is determined by the information that we can obtain from it. Monitoring the data that enters and leaves a honeypot lets us gather information that is not available to NIDS. For example, we can log the key strokes of an interactive session even if encryption is used to protect the network traffic. To detect malicious behavior, NIDS require signatures of known attacks and often fail to detect compromises that were unknown at the time it was deployed. On the other hand, honeypots can detect vulnerabilities that are not yet understood. For example, we can detect compromise by observing network traffic leaving the honeypot even if the means of the exploit has never been seen before.

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 NIDS.

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. A **high-interaction** honeypot simulates all aspects of an operating system. A **low-interaction** honeypots simulates only some parts, for example the network stack. A high-interaction honeypot can be compromised completely, allowing an adversary to gain full access to the system and use it to launch further network attacks. In contrast, low-interaction honeypots simulate only services that cannot be exploited to get complete access to the honeypot. Low-interaction honeypots are more limited, but they are useful to gather information at a higher level, *e.g.*, learn about network probes or worm activity. They can also be used to analyze spammers or for active countermeasures against worms.

We also differentiate between **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.

When gathering information about network attacks or probes, the number of deployed honeypots influences the amount and accuracy of the collected data. A good example is measuring the activity of HTTP based worms. We can identify these worms only after they complete a TCP handshake and send their payload. However, most of their connection requests will go unanswered because they contact randomly chosen IP addresses. A honeypot can capture the worm payload by configuring it to function as a web server. The more honeypots we deploy the more likely one of them is contacted by a worm.

Physical honeypots are often high-interaction, so allowing the system to be compromised completely, they are expensive to install and maintain. For large address spaces, it is impractical or impossible to deploy a physical honeypot for each IP address. In that case, we need to deploy virtual honeypots.

5. From Botnet tracking to intrusion detection

As a security device, virtual honeypots are as effective as traditional honeypots but easier to build, deploy and maintain**.**

Botnets can cause much harm in today's Internet. For example, they are often used to mount Distributed Denial of Service attacks or to send out spam or phishing mails. Moreover, botnets can be used for mass identity theft or other abuses of the compromised machines.

Observations showed that often botnets are run by young males with surprisingly limited programming skills. These people often achieve a good spread of their bots, but their actions are more or less harmless. Nevertheless, we also observed some more advanced attackers, but these persons joined the control channel only occasionally. They use only one-character nicks, issue a command, and leave. The updates of the bots they run are very professional. Probably these people use the botnets for commercial usage and sell the services. More and more attackers use their botnets for financial gain. For example, by installing browser extensions, they are able to track/fool web surfers, click pop-ups in an automated way, or post adware as presented in the previous section. A small percentage of bot-herders seem highly skilled. They strip down the software used to run the C&C server to a non-RFC-compliant daemon, not even allowing standard IRC clients to connect.

Moreover, the data we captured while observing the botnets show that these control networks are used for more than just these attacks. Possible usages of botnets can be categorized as listed here. And since a botnet is nothing more than a tool, there are most likely other potential uses that we have not listed.

5.1 Spamming: Some bots offer the possibility to open a SOCKS v4/v5 proxy —a generic proxy protocol for TCP/IP-based networking applications — on a compromised machine. After enabling the SOCKS proxy, this machine can then be used for nefarious tasks such as sending bulk e-mail (spam) or phishing mails. With the help of a botnet and thousands of bots, an attacker is able to send massive amounts of spam. Some bots also implement a special function to harvest e-mail addresses from the victims.

In addition, this can, of course, also be used to send phishing mails, since phishing is a special case of spam. Also increasing is so-called stock spam: advertising of stocks in spam e-mails. In a study we could show that stock spam indeed influences financial markets.

5.2 Spreading new malware: In many cases, botnets are used to spread new bots. This is very easy, since all bots implement mechanisms to download and execute a file via HTTP or FTP. But spreading an e-mail virus using a botnet is a very nice idea, too. A botnet with 10,000 hosts that acts as the start base for the mail virus allows very fast spreading and thus causes more harm. The Witty worm, which attacked the ICQ protocol parsing implementation in Internet Security Systems (ISS) products, is suspected to have been initially launched by a botnet because some of the attacking hosts were not running any ISS services.

5.3 Installing advertisement addons and Browser Helper Objects (BHOs): Botnets can also be used to gain financial advantages. This works by setting up a fake website with some advertisements. The operator of this website negotiates a deal with some hosting companies that pay for clicks on advertisements. With the help of a botnet, these clicks can be automated so that instantly a few thousand bots click on the pop-ups. This process can be further enhanced if the bot hijacks the start-page of a compromised machine so that the clicks are executed each time the victim uses the browser.

5.4 Google AdSense abuse: A similar abuse is also possible with Google's AdSense program. AdSense offers companies the possibility to display Google advertisements on their own website and earn money this way. The company earns money due to clicks on these ads — for example, per 10,000 clicks in one month. An attacker can abuse this program by leveraging his botnet to click on these advertisements in an automated fashion and thus artificially increment the click counter. This kind of usage for botnets is relatively uncommon but not a bad idea from an attacker's perspective.

5.5 Attacking IRC networks

Botnets are also used for DDoS attacks against IRC networks. Popular among attackers is especially the so-called clone attack. In this kind of attack, the controller orders each bot to connect a large number of clones to the victim's IRC network. The victim is overwhelmed by service requests from thousands of (cloned) bots.

5.6 Manipulating online polls/games: Online polls/games are getting more and more attention, and it is rather easy to manipulate them with botnets. Since every bot has a distinct IP address, every vote will have the same credibility as a vote cast by a real person. Online games can be manipulated in a similar way.

Currently we are aware of bots being used that way, and there is a chance that this will get more important in the future.

5.7 Sn**iffing traffic:** Bots can also use a packet sniffer to watch for interesting clear-text data passing by a compromised machine. The sniffers are mostly used to retrieve sensitive information like usernames and passwords.

But the sniffed data can also contain other interesting information: If a machine is compromised more than once and is also a member of more than one botnet, the packet sniffing allows one to gather the key information of the other botnet. Thus, it is possible to "steal" another botnet.

5.8 Key logging: If the compromised machine uses encrypted communication channels (e.g., HTTPS or POP3S), then just sniffing the network packets on the victim's computer is useless, since the appropriate key to decrypt the packets is missing. But most bots also implement functions to log keystrokes. With the help of a key logger, it is very easy for an attacker to retrieve sensitive information.

5.9 Harvesting of information: Sometimes we can also observe the harvesting of information from all compromised machines. With the help of special commands, the operator of the botnet can request a list of sensitive information from all bots.

6. Honeyd

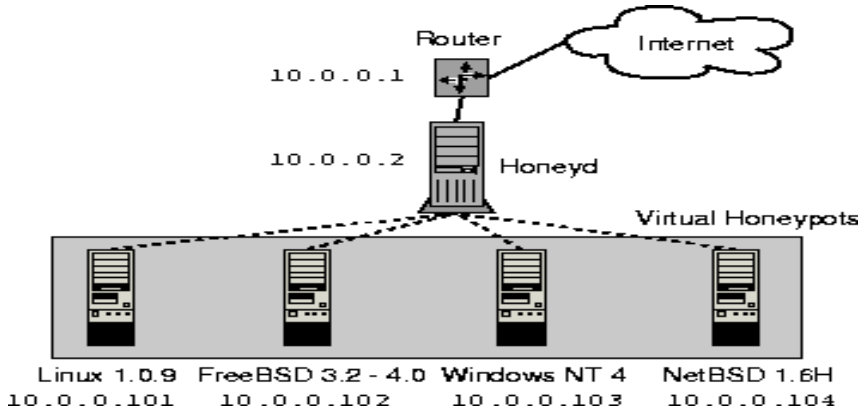
- Honeyd: Honeyd written by Neils Provos in 2002
- Honeyd, a lightweight framework for creating virtual honeypots.
- Low-interaction virtual honeypot
- Honeyd is most widely used production honeypot

6.1 Design and Implementation

In this section, we present Honeyd, a lightweight framework for creating virtual honeypots. The framework allows us to instrument thousands of IP addresses with virtual machines and corresponding network services. We start by discussing our design considerations, then describe Honeyd's architecture and implementation.

We limit adversaries to interacting with our honeypots only at the network level. Instead of simulating every aspect of an operating system, we choose to simulate only its network stack. The main drawback of this approach is that an adversary never gains access to a complete system even if he compromises a simulated service. On the other hand, we are still able to capture connection and compromise attempts. However, we can mitigate these drawbacks by combining Honeyd with a virtual machine like Vmware . For that reason, Honeyd is a low-interaction virtual honeypot that simulates TCP and UDP services. It also understands and responds correctly to ICMP messages.

Figure 6.1: Honeyd receives traffic for its virtual honeypots via a router or Proxy ARP. For each honeypot, Honeyd can simulate the network stack behavior of a different operating system.



Honeyd must be able to handle virtual honeypots on multiple IP addresses simultaneously, in order to populate the network with numerous virtual honeypots simulating different operating

systems and services. To increase the realism of our simulation, the framework must be able to simulate arbitrary network topologies. To simulate address spaces that are topologically dispersed and for load sharing, the framework also needs to support network tunneling.

A central machine intercepts network traffic sent to the IP addresses of configured honeypots and simulates their responses. Before we describe Honeyd's architecture, we explain how network packets for virtual honeypots reach the Honeyd host.

6.2 Receiving Network Data

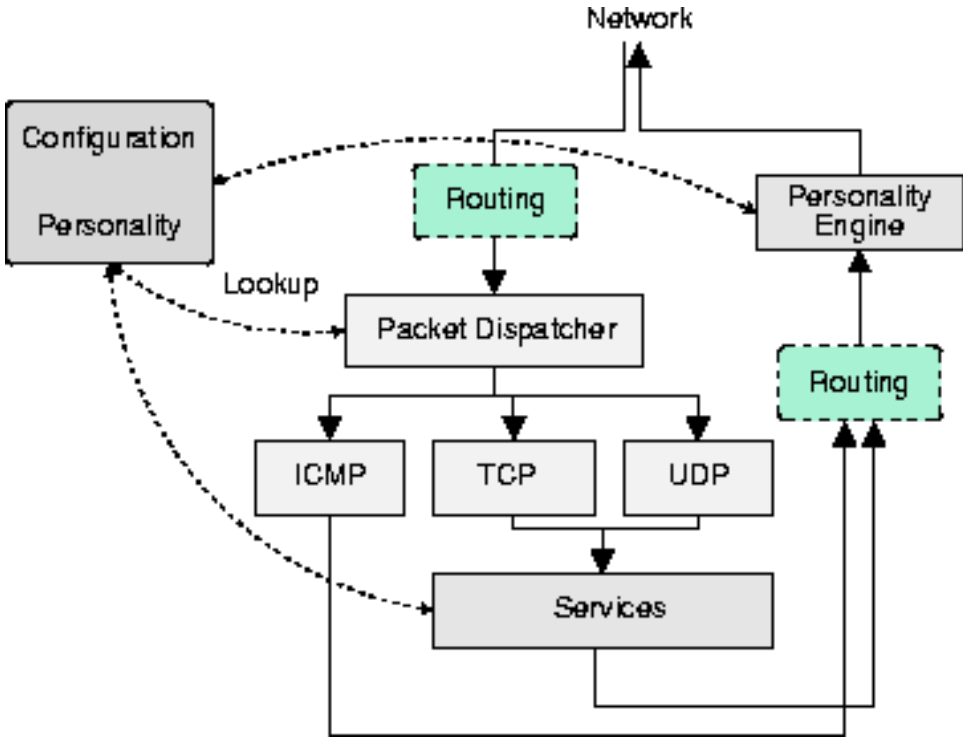
Honeyd is designed to reply to network packets whose destination IP address belongs to one of the simulated honeypots. For Honeyd, to receive the correct packets, the network needs to be configured appropriately. There are several ways to do this, *e.g.,* we can create special routes for the virtual IP addresses that point to the Honeyd host, or we can use Proxy ARP [8], or we can use network tunnels.

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 $V_1, \ldots, V_n$. 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.

In more complex environments, it is possible to tunnel network address space to a Honeyd host. We use the generic routing encapsulation (GRE) [11,12] tunneling protocol.

Figure 6.2: This diagram gives an overview of Honeyd's architecture. Incoming packets are dispatched to the correct protocol handler. For TCP and UDP, the configured services receive new data and send responses if necessary. All outgoing packets are modified by the personality engine to mimic the behavior of the configured network stack. The routing component is optional and used only when Honeyd simulates network topologies.



6.3 Architecture

Honeyd's architecture consists of several components: a configuration database, a central packet dispatcher, protocol handlers, a personality engine, and an optional routing component.

Incoming packets are processed by the central packet dispatcher. It first checks the length of an IP packet and verifies the packet's checksum. The framework is aware of the three major Internet protocols: ICMP, TCP and UDP. Packets for other protocols are logged and silently discarded.

Before it can process a packet, the dispatcher must query the configuration database to find a honeypot configuration that corresponds to the destination IP address. If no specific configuration exists, a default template is used. Given a configuration, the packet and corresponding configuration is handed to the protocol specific handler.

The ICMP protocol handler supports most ICMP requests. By default, all honeypot configurations respond to echo requests and process destination unreachable messages. The handling of other requests depends on the configured personalities.

For TCP and UDP, the framework can establish connections to arbitrary services. Services are external applications that receive data on stdin and send their output to stdout. The behavior of a service depends entirely on the external application. When a connection request is received, the framework checks if the packet is part of an established connection. In that case, any new data is sent to the already started service application. If the packet contains a connection request, a new process is created to run the appropriate service. Instead of creating a new process for each connection, the framework supports subsystems and internal services. A subsystem is an application that runs in the name space of the virtual honeypot. The subsystem specific application is started when the corresponding virtual honeypot is instantiated. A subsystem can bind to ports, accept connections, and initiate network traffic. While a subsystem runs as an external process, an internal service is a Python script that executes within Honeyd. Internal services require even less resources than subsystems but can only accept connections and not initiate them. Honeyd contains a simplified TCP state machine. The three-way handshake for connection establishment and connection teardown via FIN or RST are fully supported, but receiver and congestion window management is not fully implemented.

UDP datagrams are passed directly to the application. When the framework receives a UDP packet for a closed port, it sends an ICMP port unreachable message unless this is forbidden by the configured personality. In sending ICMP port unreachable messages, the framework allows network mapping tools like traceroute to discover the simulated network topology.

In addition to establishing a connection to a local service, the framework also supports redirection of connections. The redirection may be static or it can depend on the connection quadruple (source address, source port, destination address and destination port). Redirection lets us forward a connection request for a service on a virtual honeypot to a service running on a real server. For example, we can redirect DNS requests to a proper name server. Or we can reflect connections back to an adversary, *e.g.* just for fun we might redirect an SSH connection back to the originating host and cause the adversary to attack her own SSH server.

Before a packet is sent to the network, it is processed by the personality engine. The personality engine adjusts the packet's content so that it appears to originate from the network stack of the configured operating system.

6.4 Personality Engine

Adversaries commonly run fingerprinting tools like Xprobe [1] or Nmap [9] to gather information about a target system. It is important that honeypots do not stand out when fingerprinted. To make them appear real to a probe, Honeyd simulates the network stack behavior of a given operating system. We call this the personality of a virtual honeypot. Different personalities can be assigned to different virtual honeypots. The personality engine makes a honeypot's network stack behave as specified by the personality by introducing changes into the protocol headers of every outgoing packet so that they match the characteristics of the configured operating 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.

Next, we explain how we use the information provided by Nmap's fingerprints to change the characteristics of a honeypot's network stack.

Each Nmap fingerprint has a format similar to the example shown. We use the string after the Fingerprint token as the personality name. 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 is the most comprehensive. It 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 [2]. 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.

The next seven tests determine the stack's behavior for packets that arrive on open and closed TCP ports. The last test analyzes the ICMP response packet to a closed UDP port.

The framework keeps state for each honeypot. The state includes information about ISN generation, the boot time of the honeypot and the current IP packet identification number. Keeping state is necessary so that we can generate subsequent ISNs that follow the distribution specified by the fingerprint.

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. TCP also supports additional options, not all of which implemented by all 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.

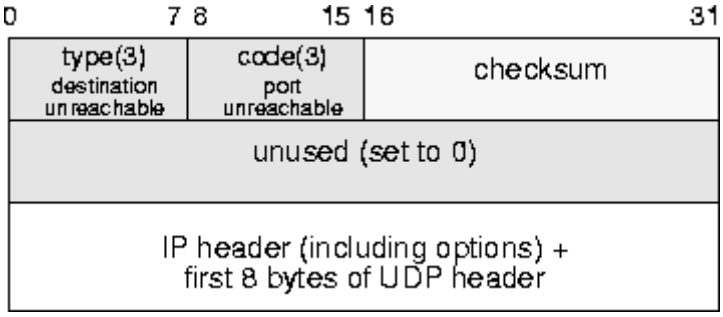
If TCP options present in the fingerprint have been negotiated during connection establishment, then Honeyd inserts them into the response packet. The framework uses the fingerprint to determine the frequency with which TCP timestamps are updated. For most operating systems, the update frequency is 2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

For the IP header, Honeyd adjusts the generation of the identification number. It can either be zero, increment by one, or random.

Figure 6.3 : The diagram shows the structure of an ICMP port unreachable message. Honeyd introduces errors into the quoted IP header to match the behavior of network stacks.



For ICMP packets, the personality engine uses the *PU* test entry to determine how the quoted IP header should be modified using the associated Xprobe fingerprint for further information. Some operating systems modify the incoming packet by changing fields from network to host order and as a result quote the IP and UDP header incorrectly. Honeyd introduces these errors if necessary. Above figure shows an example for an ICMP destination unreachable message. The framework also supports the generation of other ICMP messages, not described here due to space considerations.

6.5 Routing Topology

Honeyd simulates arbitrary virtual routing topologies to deceive adversaries and network mapping tools. This goal is different from NS-based simulators [8] which try to faithfully reproduce network behavior in order to understand it. We simulate just enough to deceive adversaries. When simulating routing topologies, it is not possible to employ Proxy ARP to direct the packets to the Honeyd host. Instead, we need to configure routers to delegate network address space to our host.

Normally, the virtual routing topology is a tree rooted where packets enter the virtual routing topology. Each interior node of the tree represents a router and each edge a link that contains latency and packet loss characteristics. Terminal nodes of the tree correspond to networks. The framework supports multiple entry points that can exit in parallel. An entry router is chosen by the network space for which it is responsible.

To simulate an asymmetric network topology, we consult the routing tables when a packet enters the framework and again when it leaves the framework. In this case, the network topology resembles a directed acyclic graph.

When the framework receives a packet, it finds the correct entry routing tree and traverses it, starting at the root until it finds a node that contains the destination IP address of the packet. Packet loss and latency of all edges on the path are accumulated to determine if the packet is dropped and how long its delivery should be delayed.

The framework also decrements the time to live (TTL) field of the packet for each traversed router. If the TTL reaches zero, the framework sends an ICMP time exceeded message with the source IP address of the router that causes the TTL to reach zero.

Figure 6.4 : An example configuration for Honeyd. The configuration language is a context-free grammar. This example creates a virtual routing topology and defines two templates: a router that can be accessed via telnet and a host that is running a web server. A real system is integrated into the virtual routing topology at IP address 10.1.0.3.

```
route entry 10.0.0.1
route 10.0.0.1 link 10.0.0.0/24
route 10.0.0.1 add net 10.1.0.0/16 10.1.0.1 latency 55ms loss 0.1
route 10.0.0.1 add net 10.2.0.0/16 10.2.0.1 latency 20ms loss 0.1
route 10.1.0.1 link 10.1.0.0/24
route 10.2.0.1 link 10.2.0.0/24

create routerone
set routerone personality "Cisco 7206 running IOS 11.1(24)"
set routerone default tcp action reset
add routerone tcp port 23 "scripts/routertelnet.pl"

create netbsd
set netbsd personality "NetBSD 1.5.2 running on a Commodore Amiga (68040 processor)"
set netbsd default tcp action reset
add netbsd tcp port 22 proxy $ipsrc:22
add netbsd tcp port 80 "scripts/web.sh"

bind 10.0.0.1 routerone
bind 10.1.0.2 netbsd
bind 10.1.0.3 to fxp0
```

For network simulations, it is possible to integrate real systems into the virtual routing topology. When the framework receives a packet for a real system, it traverses the topology until it finds a virtual router that is directly responsible for the network space that the real machine belongs to. The framework sends an ARP request if necessary to discover the hardware address of the system, then encapsulates the packet in an Ethernet frame. Similarly, the framework responds with ARP replies from the corresponding virtual router when the real system sends ARP requests.

We can split the routing topology using GRE to tunnel networks. This allows us to load balance across several Honeyd installations by delegating parts of the address space to different Honeyd hosts. Using GRE tunnels, it is also possible to delegate networks that belong to separate parts of the address space to a single Honeyd host. For the reverse route, an outgoing tunnel is selected based both on the source and the destination IP address. An example of such a configuration is described.

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.

The set and add commands change the configuration of a template. The set command assigns a personality from the Nmap fingerprint file to a template. The personality determines the behavior of the network stack. The set command also defines the default behavior for the supported network protocols. The default behavior is one of the following values: block,reset, or open. Block means that all packets for the specified protocol are dropped by default. Reset indicates that all ports are closed by default. Open means that they are all open by default. The latter settings make a difference only for UDP and TCP.

We specify the services that are remotely accessible with the add command. In addition to the template name, we need to specify the protocol, port and the command to execute for each 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.** Variable expansion allows a service to adapt its behavior depending on the particular network connection it is handling. It is also possible to redirect network probes back to the host that is doing the probing.

The **bind** command assigns a template to an IP address. If no template is assigned to an IP address, we use the **default** template. Figure shows an example configuration that specifies a routing topology and two templates. The router template mimics the network stack of a Cisco 7206 router and is accessible only via telnet. The web server template runs two services: a simple web server and a forwarder for SSH connections. In this case, the forwarder redirects SSH connections back to the connection initiator. A real machine is integrated into the virtual routing topology at IP address 10.1.0.3.

Logging

The Honeyd framework supports several ways of logging network activity. It can create connection logs that report attempted and completed connections for all protocols. More usefully, information can be gathered from the services themselves. Service applications can report data to be logged to Honeyd via **stderr.** The framework uses **syslog** to store the information on the system. In most situations, we expect that Honeyd runs in conjunction with a NIDS.

Evaluation

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

Fingerprinting

We start Honeyd with a configuration similar to the one shown in Figure and use traceroute to find the routing path to a virtual host. We notice that the measured latency is double the latency that we configured. This is correct because packets have to traverse each link twice.

Running Nmap 3.00 against IP addresses 10.0.0.1 and 10.1.0.2 results in the correct identification of the configured personalities. Nmap reports that 10.0.0.1 seems to be a Cisco router and that 10.1.0.2 seems to run NetBSD. Xprobe identifies 10.0.0.1 as Cisco router and lists a number of possible operating systems, including NetBSD, for 10.1.0.2.

To fully test if the framework deceives Nmap, we set up a B-class network populated with virtual honeypots for every fingerprint in Nmap's fingerprint file. After removing duplicates, we found 600 distinct fingerprints. The honeypots were configured so that all but one port was closed; the open port ran a web server. We then launched Nmap 3.00 against all configured IP addresses and checked which operating systems Nmap identified. For 555 fingerprints, Nmap uniquely identified the operating system simulated by Honeyd. For 37 fingerprints, Nmap presented a list of possible choices that included the simulated personality. Nmap failed to identify the correct operating system for only eight fingerprints. This might be a problem of Honeyd, or it could be due to a badly formed fingerprint database. For example, the fingerprint for a **SMC Wireless Broadband Router** is almost identical to the fingerprint for a **Linksys Wireless Broadband Router**. When evaluating fingerprints, Nmap always prefers the latter over the former.

Currently available fingerprinting tools are usually stateless because they neither open TCP connections nor explore the behavior of the TCP state machine for states other than LISTEN or CLOSE. There are several areas like congestion control and fast recovery that are likely to be different between operating systems and are not checked by fingerprinting tools. An adversary who measures the differences in TCP behavior for different states across operating system would notice that they do not differ in Honeyd and thus be able to detect virtual honeypots.

Another method to detect virtual honeypots is to analyze their performance in relation to other hosts. Sending network traffic to one virtual honeypot might affect the performance of other virtual honeypots but would not affect the performance of a real host. In the following section, we present a performance analysis of Honeyd.

Performance

We analyze Honeyd's performance on a 1.1 GHz Pentium III over an idle 100 M Bit/s network. To determine the aggregate bandwidth supported by Honeyd, we configure it to route the 10/8 network and measure its response rate to ICMP echo requests sent to IP addresses at different depths within a virtual routing topology. To get a base of comparison, we first send ICMP echo requests to the IP address of the Honeyd host because the operating system responds to these requests directly. We then send ICMP echo requests to virtual IP addresses at different depths of the virtual routing topology.

To understand how Honeyd's performance depends on the number of configured honeypots, we use a micro-benchmark that measures how the processing time per packet changes with an increasing number of configured templates. The benchmark chooses a random destination address from the configured templates and sends a TCP SYN segment to a closed port. We measure how long it takes for Honeyd to process the packet and generate a TCP RST segment. The measurement is repeated 80,000 times. Figure shows that for one thousand templates the processing time is about 0.022 ms per packet which is equivalent to about 45,000 packets per second. For 250,000 templates, the processing time increases to 0.032 ms or about 31,000 packets per second.

To evaluate Honeyd's TCP end-to-end performance, we create a simple internal echo service. When a TCP connection has been established, the service outputs a single line of status information and then echos all the input it receives. We measure how many TCP requests Honeyd can support per second by creating TCP connections from 65536 random source IP addresses in 10.1/16 to 65536 random destination addresses in 10.1/16. To decrease the client load, we developed a tool that creates TCP connections without requiring state on the client. A request is successful when the client sees its own data packet echoed by the echo service running under Honeyd. A successful transaction between a random client address C_r and a random virtual honeypot H_r requires the following exchange:

1.

$\underline{|C_r \rightarrow H_r}$: TCP SYN segment

2. $\underline{|H_r \rightarrow C_r}$: TCP SYN|ACK segment

3. $\underline{|C_r \rightarrow H_r}$: TCP ACK segment

4. $\underline{|H_r \rightarrow C_r}$: banner payload

5. $\underline{|C_r \rightarrow H_r}$: data payload

6. $\underline{|C_r \rightarrow H_r}$: TCP ACK segment (banner)

7. $\underline{|H_r \rightarrow C_r}$: TCP ACK segment (data)

8. $\underline{|H_r \rightarrow C_r}$: echoed data payload

9. $\underline{|C_r \rightarrow H_r}$: TCP RST segment

The client does not close the TCP connection via a *FIN* segment as this would require state. Depending on the load of the Honeyd machine, it is possible that the banner and echoed data payload may arrive in the same segment.

.Performance decreases for honeypots with higher latency.

Our 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. The setup for studying spammers simulates two C-class networks on a 666 MHz Pentium III.

Applications

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

1. Network Decoys

The traditional role of a honeypot is that of a network decoy. Our framework can be used to instrument the unallocated addresses of a production network with virtual honeypots. Adversaries that scan the production network can potentially be confused and deterred by the virtual honeypots. In conjunction with a NIDS, the resulting network traffic may help in getting early warning of attacks for four thousand honeypots, the third for sixty five thousand honeypots and the fourth for 262,000 honeypots.

2. Detecting and Countering Worms

Honeypots are ideally suited to intercept traffic from adversaries that randomly scan the network. This is especially true for Internet worms that use some form of random scanning for new targets, *e.g.* Blaster [5], Code Red [15], Nimda [4], Slammer [16], etc. In this section, we show how a virtual honeypot deployment can be used to detect new worms and how to launch active countermeasures against infected machines once a worm has been identified.

To intercept probes from worms, we instrument virtual honeypots on unallocated network addresses. The probability of receiving a probe depends on the number of infected machines *i*, the worm propagation chance and the number of deployed honeypots h. The worm propagation chance depends on the worm propagation algorithm, the number of vulnerable hosts and the size of the address space. In general, the larger our honeypot deployment the earlier one of the honeypots receives a worm probe.

To detect new worms, we can use the Honeyd framework in two different ways. We may deploy a large number of virtual honeypots as gateways in front of a smaller number of high-interaction honeypots. Honeyd instruments the virtual honeypots. It forwards only TCP connections that have been established and only UDP packets that carry a payload that fail to match a known fingerprint. In such a setting, Honeyd shields the high-interaction honeypots

from uninteresting scanning or backscatter activity. A high-interaction honeypot like ReVirt [7] is used to detect compromises or unusual network activity. Using the automated NIDS signature generation proposed by Kreibich et al. [14], we can then block the detected worm or exploit at the network border. The effectiveness of this approach has been analyzed by Moore. To improve it, we can configure Honeyd to replay packets to several high-interaction honeypots that run different operating systems and software versions.

On the other hand, we can use Honeyd's subsystem support to expose regular UNIX applications like OpenSSH to worms. This solution is limiting as we are restricted to detecting worms only for the operating system that is running the framework and most worms target Microsoft Windows, not UNIX.

Moore show that containing worms is not practical on an Internet scale unless a large fraction of the Internet cooperates in the containment effort. However, with the Honeyd framework, it is possible to actively counter worm propagation by immunizing infected hosts that contact our virtual honeypots. We can model the effect of immunization on worm propagation by using the classic SIR epidemic model [13]. 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:

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.

We simulate worm propagation based on the parameters for a Code-Red like worm [15,16]. 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. This shows the worm propagation resulting from a varying number of instrumented honeypots. The graph on the left shows the results if the honeypots are brought online an hour after the worm started spreading. The graph on the right shows the results if the honeypots can be activated within

twenty minutes. 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.

Alternatively, it would be possible to scan the Internet for vulnerable systems and remotely patch them. For ethical reasons, this is probably unfeasible. However, if we can reliably detect an infected machine with our virtual honeypot framework, then active immunization might be an appropriate response.

3. Spam Prevention

The Honeyd framework can be used to understand how spammers operate and to automate the identification of new spam which can then be submitted to collaborative spam filters.

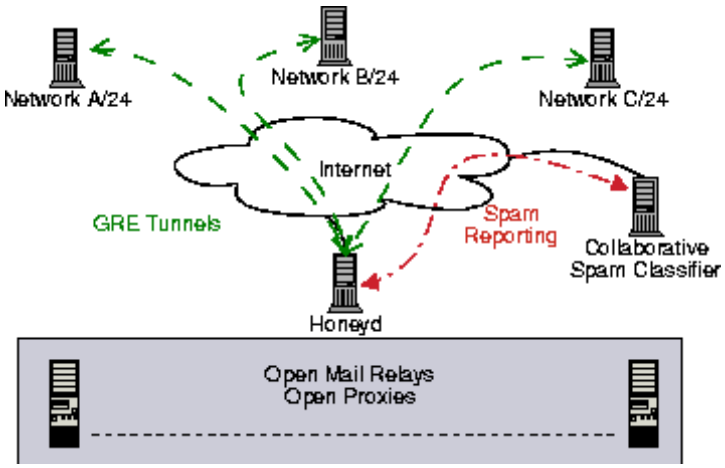
In general, spammers abuse two Internet services: proxy servers [10] and open mail relays. Open proxies are often used to connect to other proxies or to submit spam email to open mail relays. Spammers can use open proxies to anonymize their identity to prevent tracking the spam back to its origin. An open mail relay accepts email from any sender address to any recipient address. By sending spam email to open mail relays, a spammer causes the mail relay to deliver the spam in his stead.

To understand how spammers operate we use the Honeyd framework to instrument networks with open proxy servers and open mail relays. We make use of Honeyd's GRE tunneling capabilities and tunnel several C-class networks to a central Honeyd host.

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

When a spammer attempts to send spam email via an open proxy or an open mail relay, the email is automatically redirected to a spam trap. The spam trap then submits the collected spam to a collaborative spam filter.

Figure 6.5 : Using the Honeyd framework, it is possible to instrument networks to automatically capture spam and submit it to collaborative filtering systems.



Related Work

Cohen's **Deception Toolkit** provides a framework to write services that seem to contain remotely exploitable vulnerabilities [6]. Honeyd operates one level above that by providing a framework to create virtual honeypots that can run any number of services. The Deception Toolkit could be one of the services running on a virtual honeypot.

There are several areas of research in TCP/IP stack fingerprinting, among them: effective methods to classify the remote operating system either by active probing or by passive analysis of network traffic, and defeating TCP/IP stack fingerprinting by normalizing network traffic.

Fyodor's Nmap uses TCP and UDP probes to determine the operating system of a host [9]. Nmap collects the responses of a network stack to different queries and matches them to a signature database to determine the operating systems of the queried host. Nmap's fingerprint database is extensive and we use it as the reference for operating system personalities in Honeyd.

Instead of actively probing a remote host to determine its operating systems, it is possible to identify the remote operating system by passively analyzing its network packets.P0f is one such tool. The TCP/IP flags inspected by P0f are similar to the data collected in Nmap's fingerprint database.

On the other hand, Smart show how to defeat fingerprinting tools by scrubbing network packets so that artifacts identifying the remote operating system are removed. This approach is similar to Honeyd's personality engine as both systems change network packets to influence fingerprinting tools. In contrast to the fingerprint scrubber that removes identifiable

information, Honeyd changes network packets to contain artifacts of the configured operating system.

High-interaction virtual honeypots can be constructed using User Mode Linux (UML) or Vmware . One example is ReVirt which can reconstruct the state of the virtual machine for any point in time [7]. This is helpful for forensic analysis after the virtual machine has been compromised. Although high-interaction virtual honeypots can be fully compromised, it is not easy to instrument thousands of high-interaction virtual machines due to their overhead. However, the Honeyd framework allows us to instrument unallocated network space with thousands of virtual honeypots. Furthermore, we may use a combination of Honeyd and virtual machines to get the benefit of both approaches. In this case, Honeyd provides network facades and selectively proxies connections to services to backends provided by high-interaction virtual machines.

Conclusion

Honeyd is a framework for creating virtual honeypots. Honeyd mimics the network stack behavior of operating systems to deceive fingerprinting tools like Nmap and Xprobe.

We gave an overview of Honeyd's design and architecture and showed how Honeyd's personality engine can modify packets to match the fingerprints of other operating systems and how it is possible to create arbitrary virtual routing topologies.

Our performance measurements showed that a single 1.1 GHz Pentium III can simulate thousands of virtual honeypots with an aggregate bandwidths of over 30 MBit/s and that it can sustain over two thousand TCP transactions per second. Our experimental evaluation showed that Honeyd is effective in creating virtual routing topologies and successfully fools fingerprinting tools.

We showed how the Honeyd framework can be deployed to help in different areas of system security, *e.g.*, worm detection, worm countermeasures, or spam prevention.

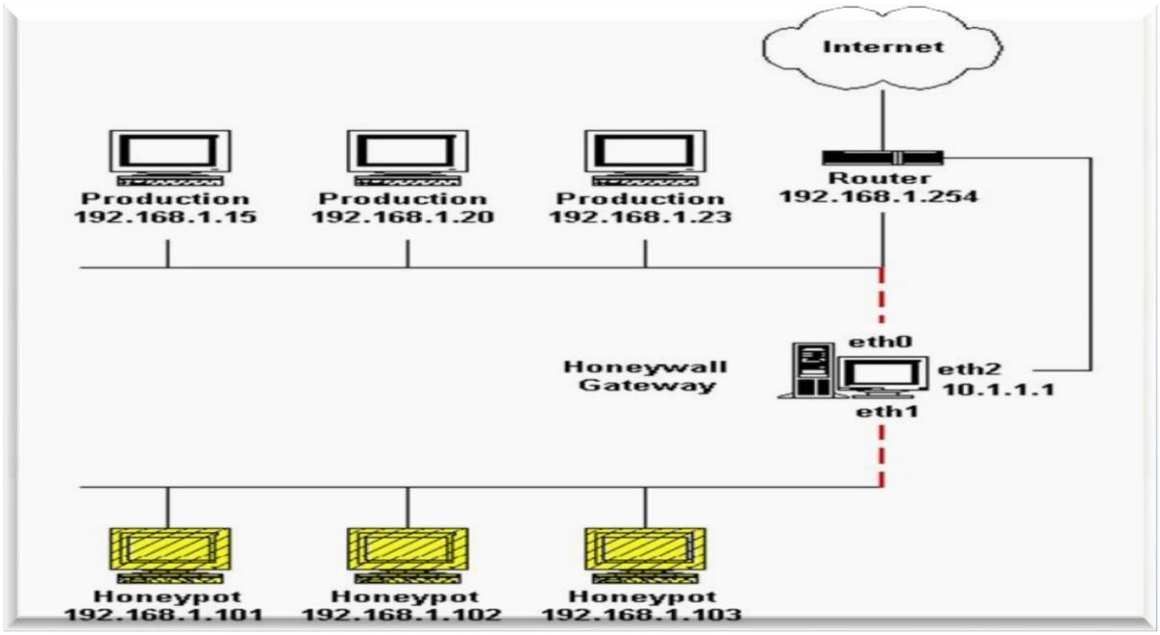
Honeyd is freely available as source code and can be downloaded from http://www.citi.umich.edu/u/provos/honeyd/.

7. Honeynet project:

- A honeynet is one type of high interaction honeypot
- Started in 2000 by a group of volunteer security professionals.

Allows full access to OS of honeypot

Fig 7.1 Honeynet Project



Advantages

- Provide Actual Operating Systems.
- Learn extensive amounts of information.

Log every packet that enters and leave honeypot

Virtual honeynets

- Virtual honeynets are one type of honeynet, specifically honeynets that run multiple operating systems on the same physical computer.

This is done using virtualization software such as VMware or User mode Linux

8. Conclusion and Future Scope

- Honeypots are a highly flexible security tool that can be used in a variety of different deployments.
- A virtual honeypot is tailored to resemble an authentic network, and is an emerging form of information technology security that actually invites hackers to perform illegal activities to learn more about hackers' activities and discover identities.
- Honeypots are unique when it comes to IT security in that hackers can't cause any real harm to simulated networks.
- Our performance measurements showed that a single 1.1 GHz Pentium III can simulate thousands of virtual honeypots with an aggregate bandwidths of over 30 M Bit/s and that it can sustain over two thousand TCP transactions per second.
- Our experimental evaluation showed that Honeyd is effective in creating virtual routing topologies and successfully fools fingerprinting tools.
- In the near future, clustering will enable organizations to quickly deploy honeypot technology globally.
- In the future, honeypots will be able to learn about networks and configure themselves, making them a lot easier to deploy in large numbers.

9. References

1. "Know Your Enemy: GenII Honeynets Easier to deploy, harder to detect, safer to maintain.".Honeynet Project. Retrieved 14 March 2016.

2. Lance Spitzner (2002). Honeypots tracking hackers. Addison-Wesley. pp. 68–70. ISBN 0-321-10895-7

3. Litke, Pat "Crypto currency-Stealing Malware Landscape" Secureworks.com. Secure Works, Retrieved 9 March 2016.

4."Bitcoin Vigil: Detecting Malware Through Bitcoin" http://cryptocoinsnews.in May 5, 2016

5. Edwards, M. "Antispam Honeypots Give Spammers Headaches".Windows IT Pro. Retrieved 11 March 2016.

6."Sophos reveals latest spam relaying countries". Help Net Security Retrieved 24 March 2016.

7."Honeypot Software, Honeypot Products, Deception Software", Intrusion Detection, Honeypots and Incident Handling Resources, Honeypots.net Retrieved on 14 March 2016.

8. "Spam hole – The Fake Open SMTP Relay Beta". SourceForge, Dice Holdings, Inc. Retrieved 14 March 2016.

9. Ec-Council (5 March 2016). Certified Ethical Hacker: Securing Network Infrastructure in Certified Ethical Hacking. Cengage Learning. pp. 3–. ISBN 978-1-4354-8365-1.

10.Kaushik, Gaurav; Tyagi, Rashmi (2016). "Honeypot : Decoy Server or System Setup Together Information Regarding an Attacker" (PDF). VSRD International Journal of Computer Science & Information Technology **2**: 155–166.

11."Secure Your Database Using Honeypot Architecture". www.dbcoretech.com. May 1, 2016. Archived from the original on March 8, 2016.

12. "Deception Toolkit". www.allnet.com 2013. Retrieved 14 March 2016.

13.Nicholas Weaver, Vern Paxson, Stuart Staniford (2003). Wormholes and a Honeyfarm: Automatically Detecting Novel Worms. The ICSI Networking and Security Group. . pp. 68–74. ISBN 0- 821-20895-7.

14. Honeynets a Honeynet Definition (PDF) by Ryan Talabis from Philippine Honeynet. Retrieved on 14 March,2016.

15. http://resources.infosecinstitute.com/virtual-honeypots/ cited on March 30, 2016.

16. "The word for "bear" www.pitt.edu. Retrieved 12 March 2016.