Simulating Networks with Honeyd

Authors: Roshen Chandran, Sangita Pakala [http://www.paladion.net]

Date: Dec 14, 2003 Version: 0.5

Acknowledgements: Niels Provos, Lance Spitzner, Ed Balas, Laurent Oudot

Honeyd (http://www.Honeyd.org) simulates virtual hosts on a network, and is actively used in Honeynet research today; Developed and maintained by Niels Provos, it's a thin daemon with lots of interesting features. It can assume the personality of any operating system, and can be configured to offer different TCP/IP "services" like HTTP, SMTP, SSH etc. Honeyd is used in honeynet research typically for setting up virtual honeypots to engage an attacker.

One useful feature of Honeyd is its ability to simulate an entire network topology within one machine – with multiple hops, packet losses and latency. This lets us simulate complex networks in test labs; it could also present a make-believe network to an attacker who gets snared in a honeynet.

Some of the features available in Honeyd for simulating networks are:

- Simulation of large network topologies
- Configurable network characteristics like latency, loss and bandwidth
- Supports multiple entry routers to serve multiple networks
- Integrate physical machines into the network topology
- Asymmetric routing
- GRE tunneling for setting up distributed networks

This guide shows you how to simulate network topologies using Honeyd and includes sample configurations. If you are new to Honeyd, you will enjoy reading Lance Spitzner's detailed article at Security Focus on how to setup honeypots with Honeyd[1]. The Honeyd man page contains the syntax for the command line options and the usage of the configuration file. [2] Niels Provos paper titled "A Virtual Honeypot Framework" is an excellent read to understand the context and background of Honeyd.[3] In this guide, we shall focus on building a network step-by-step and look closely at each configuration line. The Appendix contains the sample honeyd.conf that simulates the network we develop in this guide.

For this guide, our physical network contains four desktops plus a system that we designate as the Honeyd host. The virtual network that we want to simulate will be hosted on the Honeyd host. Honeyd works on Unix flavors and has been ported to Windows by Michael Davis[4]. In our example, we use a Linux 7.3 machine as the Honeyd host. Our physical network has been configured for 10.0.0.0/24, and the Honeyd host has been assigned the 10.0.0.1 IP address as shown in figure 1.

[1] Open Source Honeypots: Learning with Honeyd, http://www.securityfocus.com/infocus/1659

[2] Honeyd Man Page, http://www.citi.umich.edu/u/provos/honeyd/honeyd-man.pdf

[3] A Virtual Honeypot Framework, http://www.citi.umich.edu/techreports/reports/citi-tr-03-1.pdf

[4] Honeyd for Windows, http://www.securityprofiling.com/honeyd/honeyd.shtml

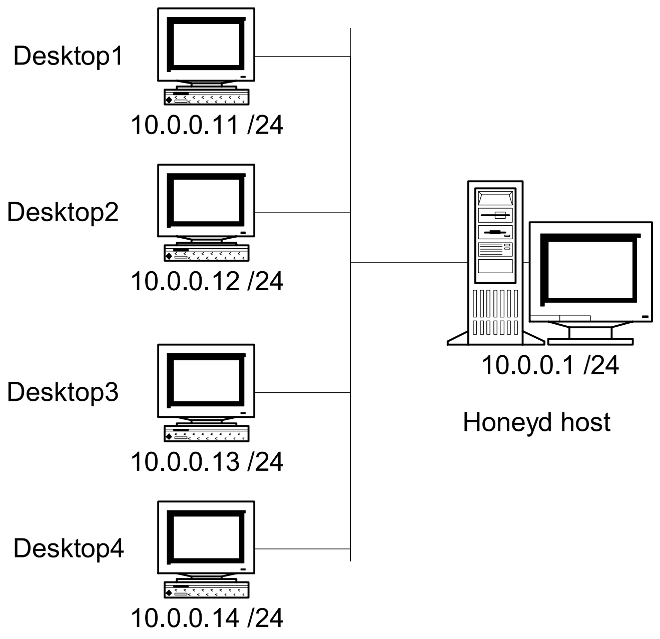

Figure 1

Setting up two honeypots

Let’s first take a quick look at how we set up two virtual honeypots on the Honeyd host. We want our two honeypots to take the 10.0.0.51 and 10.0.0.52 IP addresses and simulate Windows machines. This is shown in figure 2. The blue dotted line indicates the Honeyd host that simulates the virtual honeypots.

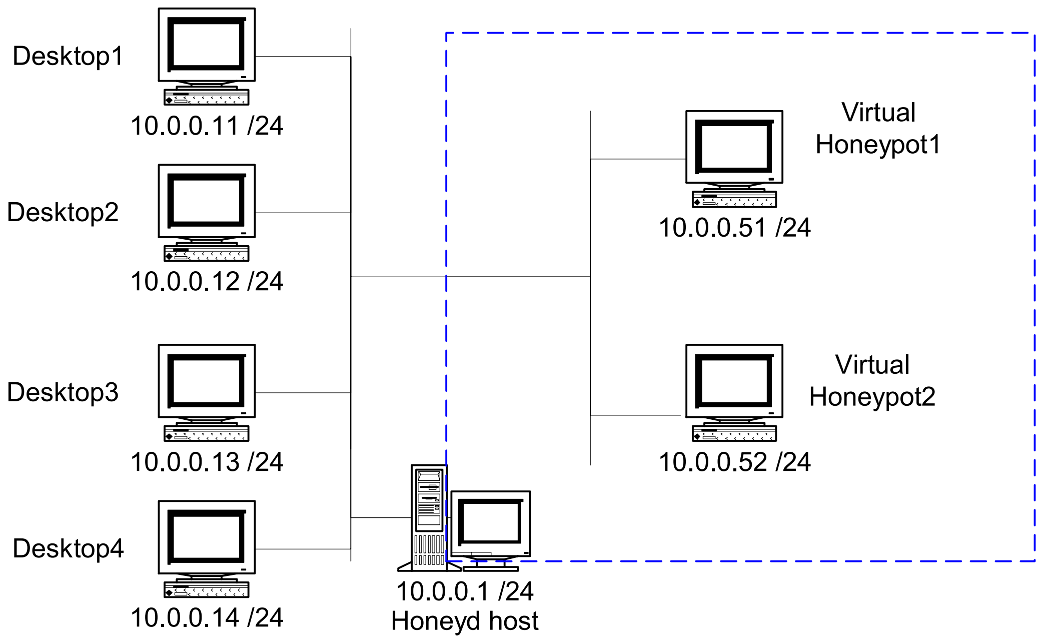

Figure 2

Before configuring and running Honeyd, we need to ensure that the Honeyd host responds to arp request for the IPs of the honeypots we are hosting. This is achieved by using the arpd software to spoof arp responses on behalf of the honeypots.

#arpd 10.0.0.0/8

arpd will now respond with the MAC address of the Honeyd host for any request to an unused IP in the 10.x.x.x address space.

Before we start Honeyd, we need to configure Honeyd using a configuration file (in our case, the honeyd.conf file) to host the two Windows machines. The configuration file follows a context-free grammar that is quite straight-forward. Here's how our conf file looks:

```
### Windows computers
create windows
set windows personality "Windows NT 4.0 Server SP5-SP6"
add windows tcp port 80 "perl scripts/iis-0.95/iisemul8.pl"
add windows tcp port 139 open
add windows tcp port 137 open
add windows udp port 137 open
add windows udp port 135 open
set windows default tcp action reset
set windows default udp action reset
bind 10.0.0.51 windows
bind 10.0.0.52 windows
```

The above lines create a template called "windows" and bind the two honeypot IP addresses to the template. The above windows template tells Honeyd to present itself as a Windows NT 4.0 SP5-SP6 when a client tries to fingerprint the honeypot with NMap or XProbe. Five ports are open on the honeypot, 80/tcp, 139/tcp, 137/tcp, 137/udp and and 135/udp. When a machine connects to port 80 of the honeypot, the honeypot will engage the client with an IIS emulator perl script. For ports that are closed, the configuration specifies that a RST be sent in the case of TCP, and an ICMP Port Unreachable message be sent for UDP.

With this configuration file, we can start Honeyd from the command line:

#./honeyd -f honeyd.conf 10.0.0.51-10.0.0.52

At this point, Honeyd starts listening and responding to packets for the two virtual systems it is hosting at 10.0.0.51 and 10.0.0.52. The IP of the Honeyd host is still reachable; if we wish to protect the Honeyd host in a honeynet, then that IP should be firewalled.

Setting up a router in the network

Now, let's look at how we simulate a simple network with Honeyd. Our simulated network uses the address space of 10.0.1.0/24; it contains two honeypots and is separated from the LAN by a Cisco router (R1) as shown in figure 3.

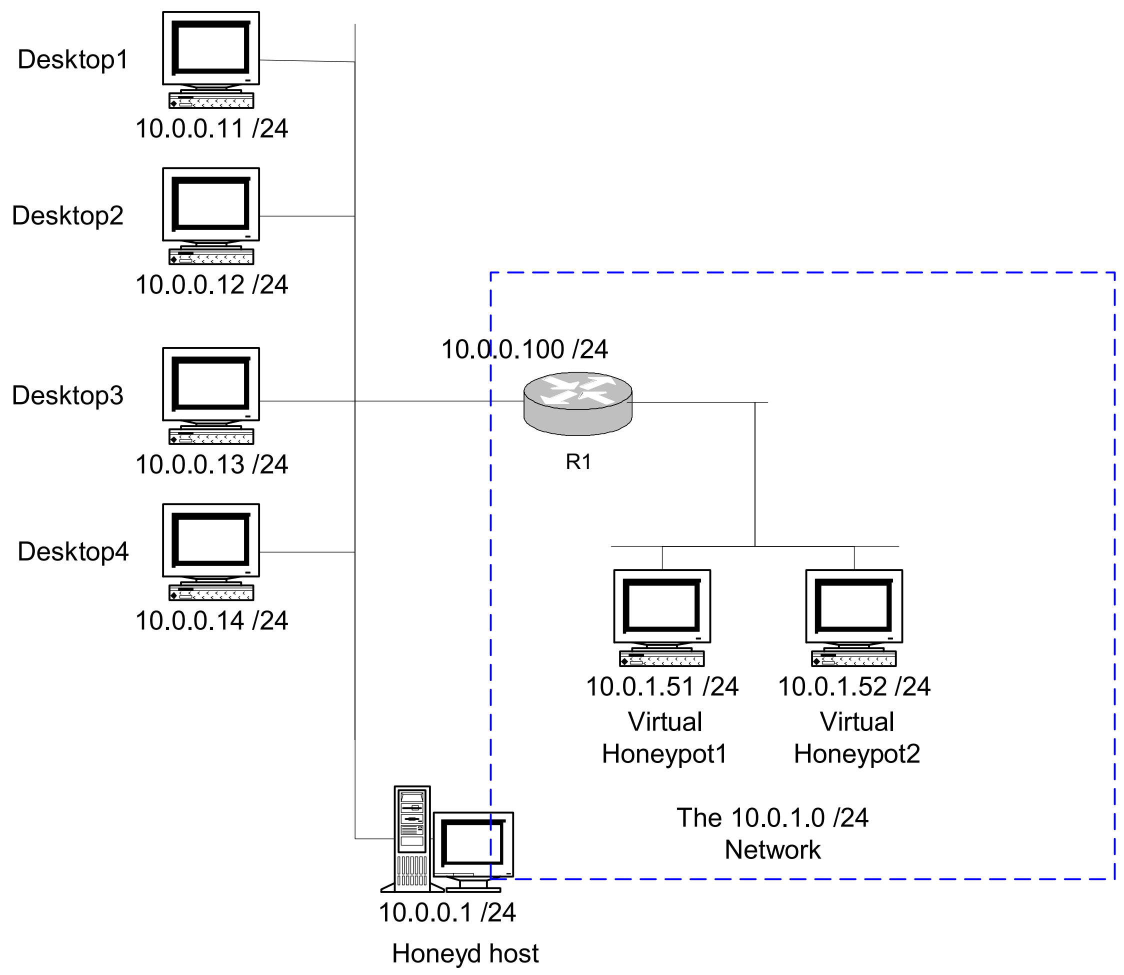

Figure 3

To simulate this network, we first create a Cisco router and bind it to the 10.0.0.100 IP address:

```
### Cisco router
create router
set router personality "Cisco IOS 11.3 - 12.0(11)"
set router default tcp action reset
set router default udp action reset
add router tcp port 23 "/usr/bin/perl scripts/router-telnet.pl"
set router uid 32767 gid 32767
set router uptime 1327650
bind 10.0.0.100 router
```

The router R1 is the entry point into the virtual network from the LAN; the “route entry” configuration is used to specify the entry point:

```
route entry 10.0.0.100 network 10.0.0.0/16
```

The above line instructs Honeyd that 10.0.0.100 is the entry point to our virtual network 10.0.0.0/16. As we shall see later, it is possible to have multiple entry routers, each serving different network ranges.

The 10.0.1.0/24 network is directly reachable from the router R1. The “route link” configuration command is used to specify which network is directly reachable and does not require further hops to be reached. In our case, the configuration line takes the form:

```
route 10.0.0.100 link 10.0.1.0/24
```

The first IP address specified above is the IP of the router. The network address specified after the link keyword defines which network is directly accessible. Multiple link commands can be used to attach multiple subnets directly to a router.

The two honeypots at 10.0.1.51 and 10.0.1.52 can be setup by binding the two IP addresses to our Windows honeypot template.

```
bind 10.0.1.51 windows
bind 10.0.1.52 windows
```

At this point, the configuration for our simple network is complete. Run the Honeyd command and point it to the configuration file to bring up our network.

Setting up a Network with Two Routers

Now that we have a simple network setup, let’s look at a slightly more complex one. In figure 4, we have added another network separated from the first network by a router R2 with an IP address of 10.0.1.100. The new network has the address range of 10.1.0.0/16 and hosts two honeypots at 10.1.0.51 and 10.1.0.52.

We first need to add a new gateway (R2) in our configuration file. The “route add net” command is used for adding a gateway, and here’s how our add net command looks:

```
route 10.0.0.100 add net 10.1.0.0/16 10.0.1.100
```

The above configuration line specifies that 10.0.0.100 (the router R1) can reach the network 10.1.0.0/16 via the gateway 10.0.1.100 (the router R2). The first IP in the line is that of R1, the last IP address is that of the new gateway, and the address range specified is that of the network reachable through the new gateway.

After we have added the router R2, we need to specify which IP addresses are reachable directly from R2. Once again, we use the route link command to achieve this. In our network, the 10.1.0.0/16 subnet is directly reachable from R2. So, the command takes the following form:

```
route 10.0.1.100 link 10.1.0.0/16
```

We next add the two honeypots by binding their IP addresses to the honeypot template.

```
bind 10.1.0.51 windows
bind 10.1.0.52 windows
```

The configuration is complete and we can launch Honeyd now to simulate the network of figure 4.

Let’s take a quick stock of where we have reached. We can specify an entry point to our virtual network with the “route entry network” configuration line. To indicate networks that are directly reachable from a gateway, we use the “route link” configuration line. We can add new gateways to reach other subnets by using the “route add net” line. These three configurations are the basic blocks for building large network topologies with Honeyd. By using a combination of these configurations, more complex networks can be simulated.

As an example, let us extend the network we have simulated one more step. In the process we shall also see a few other interesting features of Honeyd.

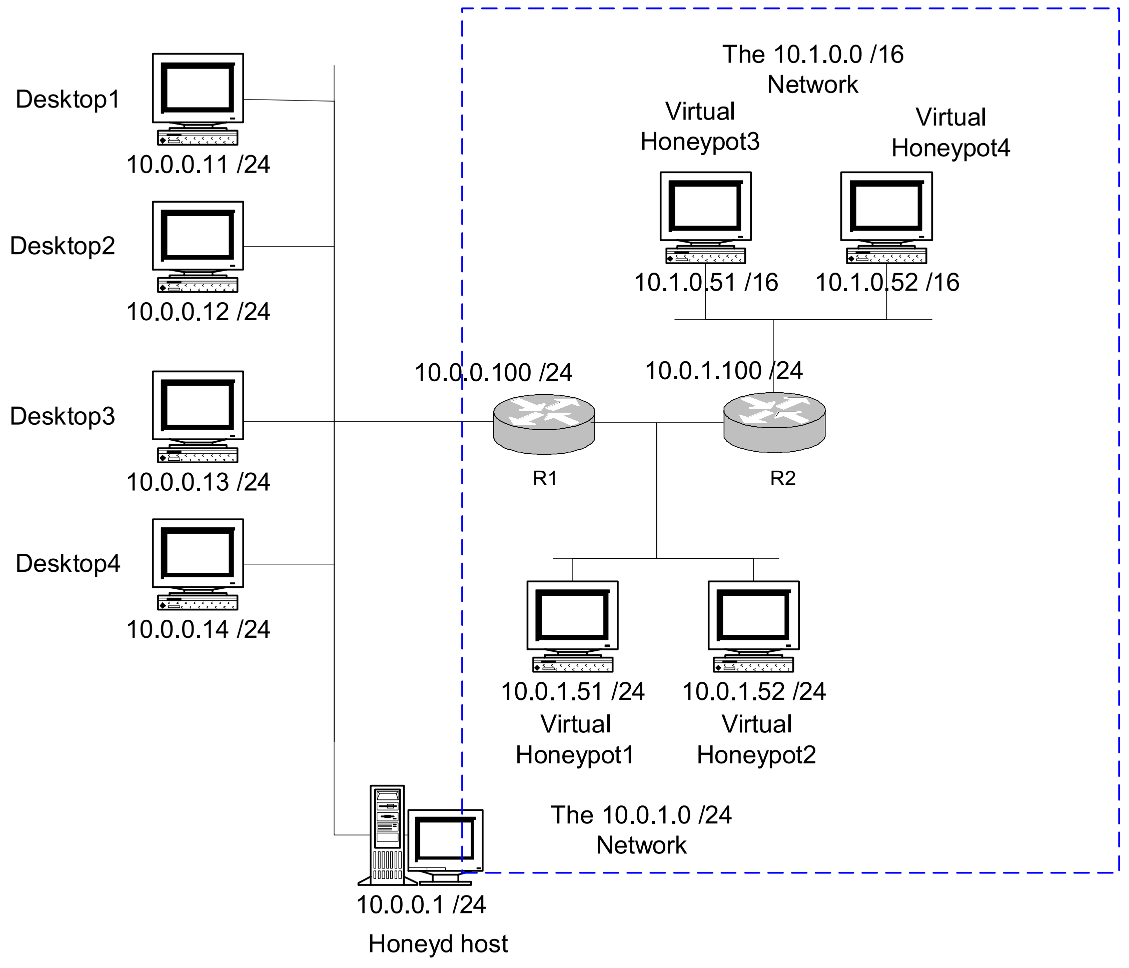

Figure 4

Configuring Latency, Loss and Bandwidth

We add a third network, one hop from R2 and deploy two virtual honeypots there as shown in figure 5.

Adding this network to our configuration file should be quite easy now:

```
route 10.0.1.100 add net 10.1.1.0/24 10.1.0.100 latency 50ms loss 0.1
bandwidth 1Mbps
route 10.1.0.100 link 10.1.1.0/24
bind 10.1.1.51 windows
bind 10.1.1.52 windows
```

The above lines add the IP address 10.1.0.100 as a gateway to reach the 10.1.1.0/24 network, and deploy two honeypots at 10.1.1.51 and 10.1.1.52. Additionally, the route add net command also specifies latency, loss and bandwidth details for the connection between routers R2 and R3.

In the real world, each additional hop adds a delay to the total time to reach the destination. This can be simulated using the latency keyword- the delay at each hop can be specified in milliseconds. Networks in the real world also tend to be less than perfect while transmitting packet - a few packets could get lost. The loss keyword can be used to model this behavior of network links by specifying the loss in %. Honeyd also queues packets if a link is occupied by a previous packet. Depending on the bandwidth available for a link, these delays can vary. The bandwidth of a link can be specified in Kbps, Mbps or Gbps with the bandwidth keyword.

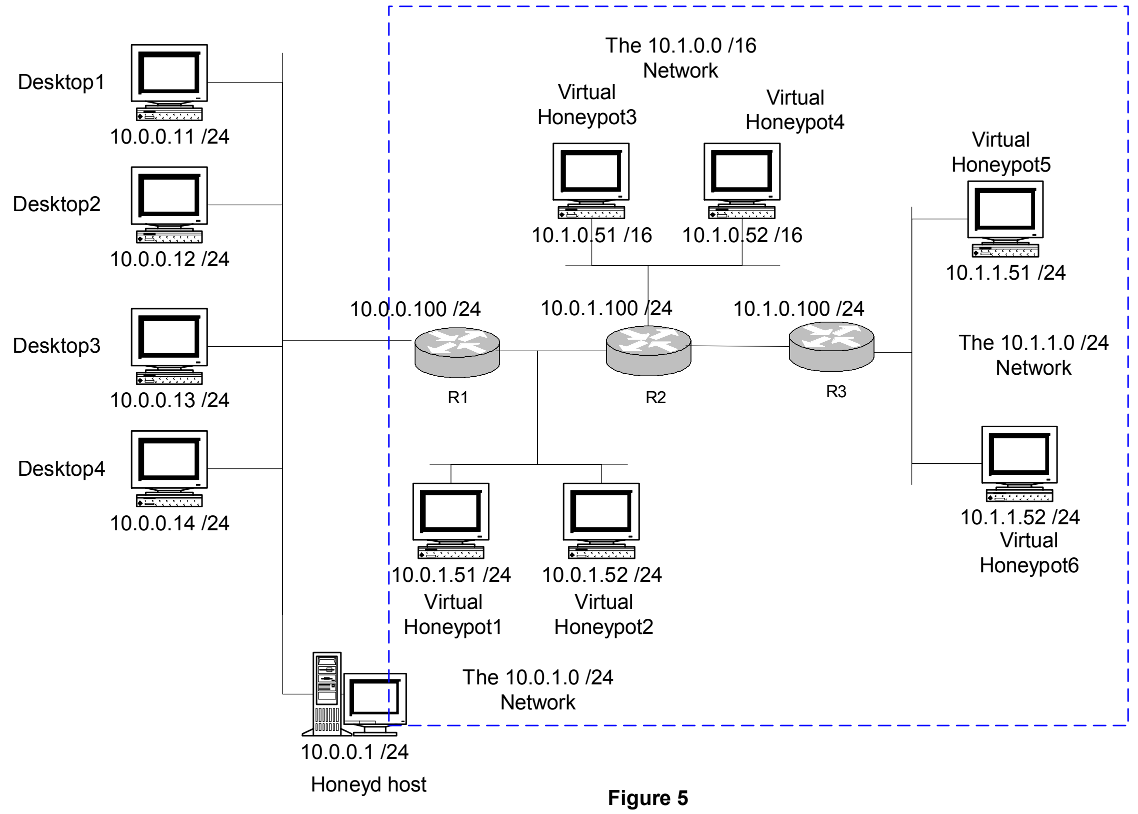

Figure 5

Integrating Physical Machines into the Topology

Honeyd also supports integrating these virtual network topologies with other physical hosts in the network. Consider the host 10.1.1.53, a physically separate host that we wish to integrate into the simulated network we have created so far. The host is physically located on the same LAN as the Honeyd host and the other desktops, but we wish to logically place it several hops inside our virtual network. Figure 6 shows how the network looks:

We use the bind command to attach an external physical host to the network. The configuration line reads:

```
bind 10.1.1.53 to eth0
```

The above line tells Honeyd that 10.1.1.53 can be reached via interface eth0; as 10.1.1.53 is logically on the segment behind R3, a packet to that IP will traverse routers R1, R2 and R3 before reaching it. To see this, let’s trace the route to 10.1.1.53 from the Desktop1 on the LAN.

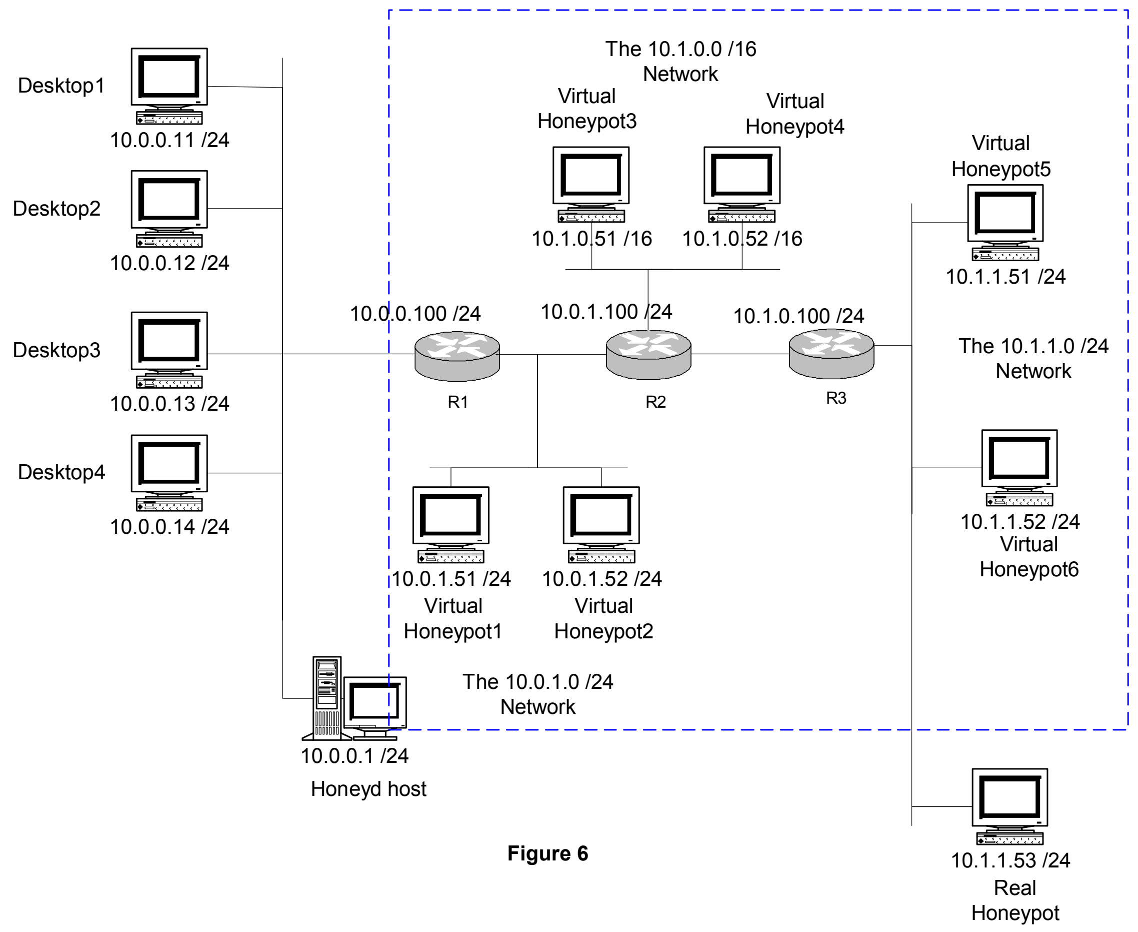

Figure 6

C:\>tracert 10.1.1.53

```
Tracing route to 10.1.1.53 over a maximum of 30 hops
  1     *        *        *     Request timed out.
  2   <10 ms    10 ms    10 ms  10.0.1.100
  3    10 ms    20 ms    20 ms  10.1.0.100
  4    10 ms    20 ms    20 ms  10.1.1.53
Trace complete.
```

As the above output shows, the 10.1.1.53 machine is reached through 3 virtual intermediate hops, even though it is on the same physical network as the other machines.

Multiple Entry Routers

Honeyd also allows multiple entry points into the virtual network. For instance, in figure 7 we add a new network that is reachable via the router R4 at 10.0.0.200. Creating a new entry point is quite simple- we use the route entry command again to define the new router. The rest of the network can then be built the same way with a combination of “route add net” and “route link” configurations. For this network, here’s the configuration for the second entry point and the network behind it:

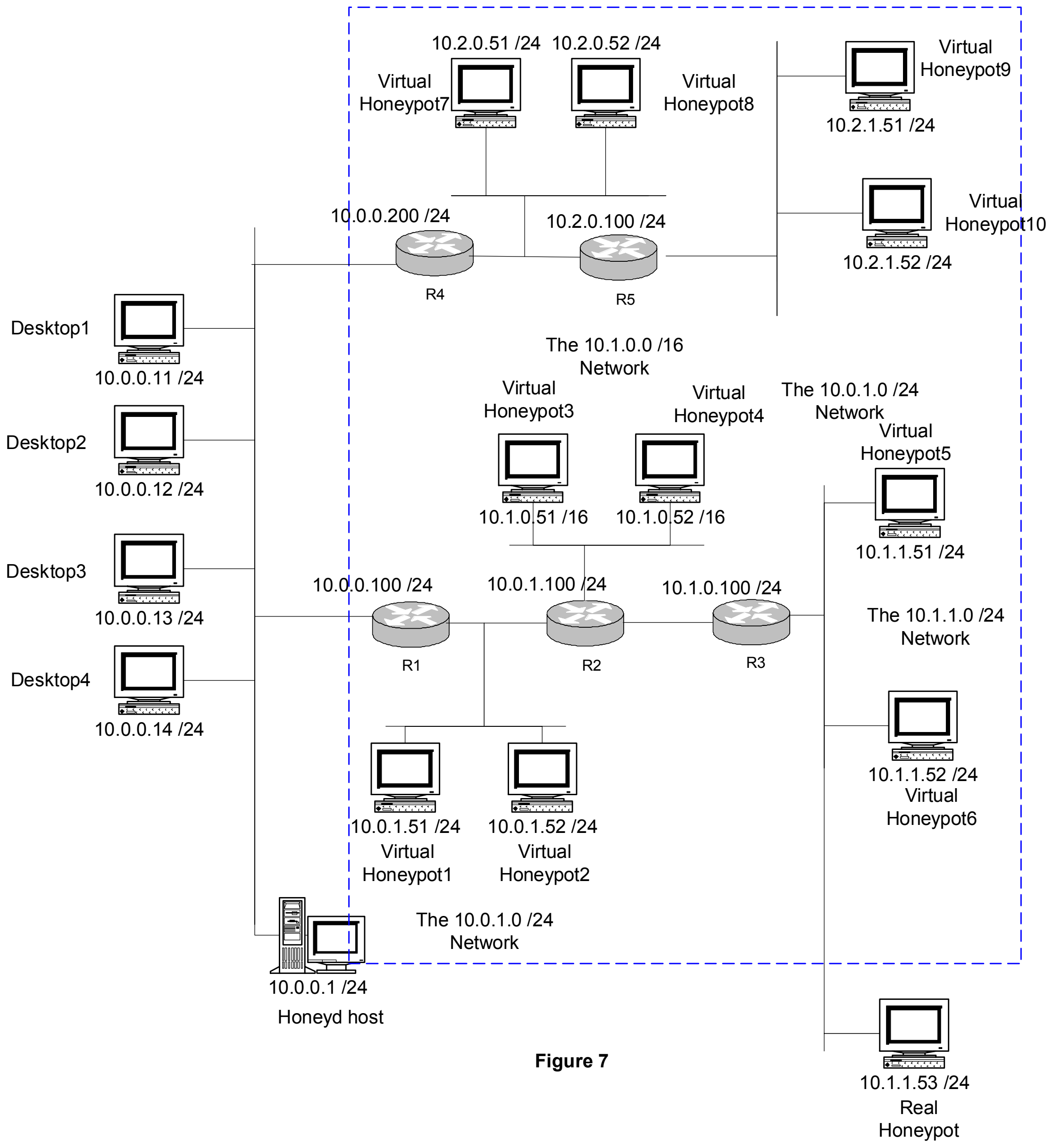

Figure 7

```
route entry 10.0.0.200 network 10.2.0.0/16
route 10.0.0.200 link 10.2.0.0/24
route 10.0.0.200 add net 10.2.1.0/24 10.2.0.100
route 10.2.0.100 link 10.2.1.0/24
bind 10.0.0.200 router
bind 10.2.0.100 router
bind 10.2.0.51 windows
bind 10.2.0.52 windows
bind 10.2.1.51 windows
bind 10.2.1.52 windows
```

The route entry adds 10.0.0.200 as a new router to serve the 10.2.0.0/16 network; the route link then specifies that the 10.2.0.0./24 network is directly reachable from this router, R4. The route add net then adds a new gateway at 10.2.0.100 that servers the 10.2.1.0/16 network. The next route link indicates that the 10.2.1.0/24 network is directly reachable from this new router. We then bind the new router IP addresses to the router template, and the 4 honeypot addresses to the windows template.

GRE Tunneling for Distributed Setups

One of the interesting features of Honeyd is its ability to tunnel data from the virtual network within a Honeyd host to other remote networks for distributed setups. This feature can be used to connect together a distributed network of Honeyd hosts and also to serve different distributed networks from a single Honeyd host. In the latter case which we shall show in our guide, the Honeyd host can serve as a low interaction honeypot farm, a kind of centralized site. Participants can monitor a section of their IP space by routing traffic to that IP space over a tunnel to the Honeyd host. Figure 8 illustrates this concept.

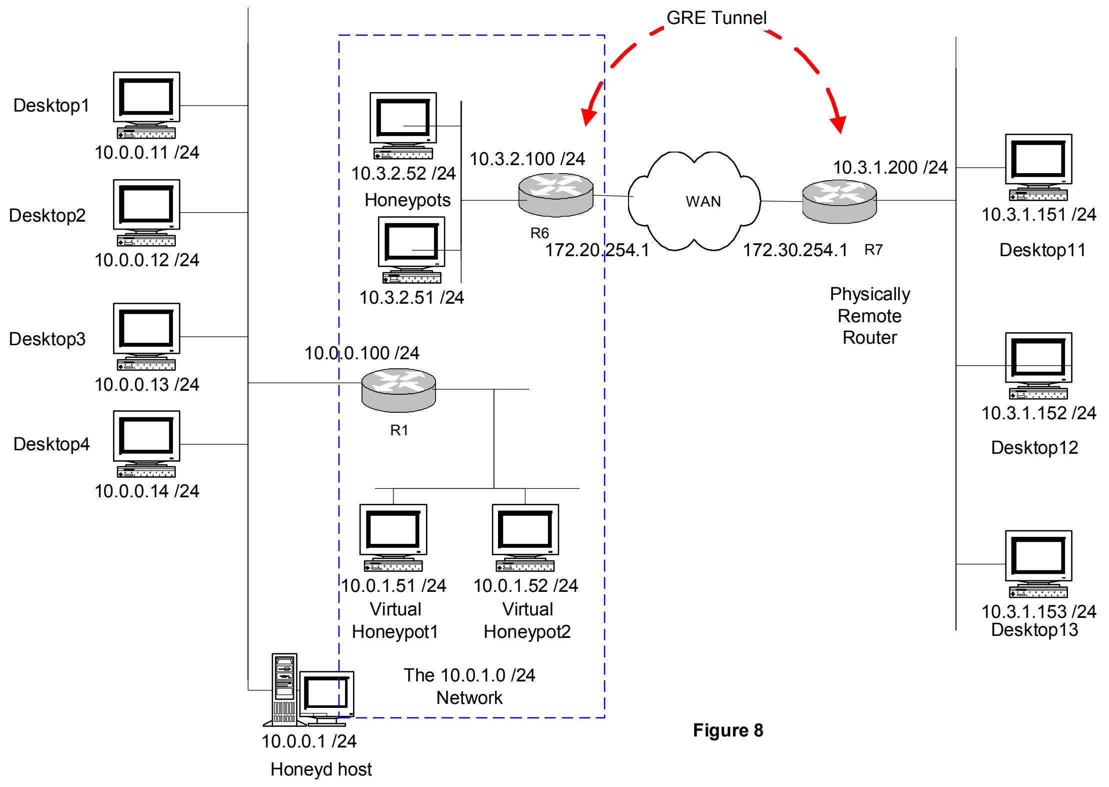

Figure 8

To see how GRE tunneling can be configured, let's take a simplified version of the network we have been developing so far. In figure 8 the 10.3.1.0/24 network is located across the WAN, say across the country. Using the GRE tunneling feature of Honeyd, it is possible to tunnel the data from the router R7 to the virtual router at R6 as if the two routers were connected directly. To a user on the network across the country it would seem that the network simulated within the Honeyd host is just one hop away, as the traffic gets tunneled through the virtual link we setup between the two routers. Our Honeyd host could thus be used to serve the remote network as well.

To configure our Honeyd host to setup GRE tunnels as in figure 8, we first add R6 whose external interface is 172.20.254.1 as a virtual router which we shall dedicate for the tunnel. R6 is a new entry router that enables traffic to reach the 10.3.2.0/24 network. While it is not required to have a dedicated router for the tunnel, it is easier to understand the configuration with a dedicated virtual router.

```
route entry 172.20.254.1 network 10.3.2.0/24
```

We then indicate that the 10.3.2.0/24 network is directly reachable from the above router with the "route link" configuration.

```
route 172.20.254.1 link 10.3.2.0/24
```

To setup the tunnel, the "route add net tunnel" configuration. The first termination point for the tunnel is the external IP address of R6, ie. 172.20.254.1. The remote physical router's IP address 172.30.254.1 will be the second terminating point for the tunnel. In the configuration line, we also specify the virtual router, and the remote network range that will be using the tunnel, as follows:

```
route 172.20.254.1 add net 10.3.1.0/24 tunnel 172.20.254.1 172.30.254.1
```

Please note that the router R7 across the WAN should also be configured to terminate the GRE tunnel from the virtual router and route the packets to the destination.

At this point, any traffic from the 10.3.1.0/24 network to the honeypots hosted within Honeyd will go through a GRE tunnel between the routers R6 and R7. With this feature, we can have a distributed setup and create a virtual network topology across them hiding the underlying physical network from attackers.

Summary

We have seen how to setup virtual network topologies in a box using Honeyd. By using a combination of few commands, it is possible to simulate complex networks and model typical network behavior. The network we setup included the following features:

- Multiple entry points
- Multiple hops to reach destinations
- Links with latency, loss and bandwidth specifications
- Integration of external physical machines to the topology
- GRE tunneling for a distributed Honeyd setup.

And with each new version of Honeyd, newer and more powerful features are being added.

The appendix gives a sample template for simulating network topologies with Honeyd.

Appendix – honeyd.conf for a Sample Network

```
#################################################
#File: honeyd.conf for a sample network
#Date: Dec 14, 2003
#Ver: 0.5
#
#Prepared by:    Roshen Chandran, Sangita Pakala
#                Paladion Networks [http://www.paladion.net]
#
#Thanks to:      Niels Provos, Lance Spitzner, Ed Balas, Laurent Oudot
#
#This sample network configuration template builds
#a virtual network step-by-step. The network we simulate
#has multiple hops, two entry points, a GRE tunnel to a remote location
# and integrates external physical hosts to the virtual network.
#
#################################################


#To create the router at the entry point, use the
#route entry command and specify the IP address of
#the router and the network reachable through it.

route entry 10.0.0.100 network 10.0.0.0/16

#To specify the IP addresses directly reachable from
#a router, use the route link configuration. In the
#example below, we specify that the 10.0.1.0/24
#network is directly reachable from the 10.0.0.100 router.

route 10.0.0.100 link 10.0.1.0/24

# Add a new router connected to an existing router
#in the network by using the route add net
#directive. Specify the network range that can be
#reached by the new router and the IP address of the
#new router. In the example below, we add
#10.0.1.100 as a new router that serves the
#10.1.0.0/16 network and connected to the first
#router 10.0.0.100

route 10.0.0.100 add net 10.1.0.0/16 10.0.1.100

#Specify the range of IP addresses that are directly
#reachable from the new router with the route link
#configuration. Here, we indicate that 10.1.0.0/16
#is directly accessible from the router 10.0.1.100 we
#newly added

route 10.0.1.100 link 10.1.0.0/16

#Here we add another router connected to 10.0.1.100
#that can reach the 10.1.1.0/24 network. The new
#router takes the IP 10.1.0.100. Additionally, we
#also specify the network characteristics of that
```

```
#link using the latency, loss and bandwidth keywords.

route 10.0.1.100 add net 10.1.1.0/24 10.1.0.100 latency 50ms loss 0.1 bandwidth 1Mbps

#With the route link configuration, we next
#specify that the 10.1.1.0/24 network is directly
#accessible from the 10.1.0.100 router.

route 10.1.0.100 link 10.1.1.0/24

#External physical machines can be integrated into the
#virtual network topology of the honeynet. The bind
#to interface configuration is used to attach external
#machines into the network. In our example here,
#the external machine at 10.1.1.53 is integrated
#into the virtual network through eth0.

bind 10.1.1.53 to eth0

#Multiple entry points may be defined in Honeyd for the
#virtual network by using additional route entry
#configurations. Here we add 10.0.0.200 as a new entry
#router and then define an entire network behind it.

route entry 10.0.0.200 network 10.2.0.0/16
route 10.0.0.200 link 10.2.0.0/24
route 10.0.0.200 add net 10.2.1.0/24 10.2.0.100
route 10.2.0.100 link 10.2.1.0/24

# We can setup GRE tunnels to other networks located across
#a WAN or the Internet by using the tunnel keyword.
#For simplicity, we first create a dedicated virtual router 172.20.254.1
#for the GRE tunneling. The 10.3.2.0/24 network containing honeypots
#is directly connected to this virtual router.
#
#To setup a tunnel to the 10.3.1.0/24 network
#located across the WAN, we setup a tunnel with 172.20.254.1 and
# 172.30.254.1 as the two points of termination. The destination
#router should know how to decapsulate the GRE packets and
#route them to the 10.3.1.0/24 network. The source and
#destination are specified after the tunnel keyword of the
#route add net configuration line as follows.

route entry 172.20.254.1 network 10.3.2.0/24
route 172.20.254.1 link 10.3.2.0/24
route 172.20.254.1 add net 10.3.1.0/24 tunnel 172.20.254.1 172.30.254.1

#IP addresses are assigned to virtual hosts that we
#want to simulate within Honeyd with the bind
#configuration. Here, we bind the honeypot IPs
#to a template called windows that we have defined.

create windows
set windows personality "Windows NT 4.0 Server SP5-SP6"
add windows tcp port 80 "perl scripts/iis-0.95/iisemul8.pl"
add windows tcp port 139 open
add windows tcp port 137 open
add windows udp port 137 open
add windows udp port 135 open
set windows default tcp action reset
set windows default udp action reset

bind 10.0.1.51 windows
bind 10.0.1.52 windows
bind 10.1.0.51 windows
bind 10.1.0.52 windows
bind 10.1.1.51 windows
bind 10.1.1.52 windows
bind 10.2.0.51 windows
bind 10.2.0.52 windows
bind 10.2.1.51 windows
bind 10.2.1.52 windows
bind 10.3.2.51 windows
bind 10.3.2.52 windows

#The routers we have created in the virtual network
#also need to be bound to templates to model their
#behavior. We have created a template called router
#and bound the router IP addresses to that template.

create router
set router personality "Cisco IOS 11.3 - 12.0(11)"
set router default tcp action reset
set router default udp action reset
add router tcp port 23 "/usr/bin/perl scripts/router-telnet.pl"
set router uid 32767 gid 32767
set router uptime 1327650

bind 10.0.0.100 router
bind 10.0.1.100 router
bind 10.1.0.100 router
bind 10.0.0.200 router
bind 10.2.0.100 router
bind 172.20.254.1 router
```