NAME

honeyd − Honeypot Daemon

SYNOPSIS

honeyd [**-dP**] [**-l** *logfile*] [**-p** *fingerprints*] [**-x** *xprobe*] [**-a** *assoc*] [**-f** *file*] [**-i** *interface*] [**-V|--version**] [**-h|--help**] [**--include-dir**] [**-i** *interface*] [*net ...*]

DESCRIPTION

Honeyd creates virtual hosts for IP addresses matching the specified *net*. The daemon simulates the networking stack of the configured hosts and can simulate any TCP and UDP service. ICMP is fully supported, too. By default, all UDP ports are closed and **honeyd** will generate an ICMP unreachable port message if the configured personality permits that.

Honeyd enables a single host to claim unused addresses on a LAN for network simulation. The *net* argument may contain multiple addresses and network ranges.

In order for **honeyd** to receive network traffic for IP addresses that it should simulate, it is necessary to either explicitly route traffic to it, use proxy arp or run arpd(8) for unassigned IP addresses on a shared network.

honeyd exits on an interrupt or termination signal.

The options are as follows:

-d Do not daemonize, and enable verbose debugging messages.

-P On some operating systems, it is not possible to get event notifications for pcap via select(2). In that case, **honeyd** needs to run in polling mode. This flag enables polling.

-l *logfile*
Log packets and connections to the logfile specified by *logfile*.

-p *fingerprints*
Read **nmap** style fingerprints. The names defined after the token are stored as personalities. The personalities can be used in the configuration file to modify the behaviour of the simulated TCP stack.

-x *xprobe*
Read **xprobe** style fingerprints. This file determines how **honeyd** reacts to ICMP fingerprinting tools.

-a *assoc*
Read the file that associates **nmap** style fingerprints with **xprobe** style fingerprints.

-f *file*
Read the configuration in *file*. It is possible to create host templates with the configuration file that specify which servers should run and which scripts should be started to simulate them. **honeyd** will reread the configuration file when sent a SIGHUP signal.

The syntax is as follows:

```
config  = creation | addition | binding | set |
          annotate | route [config] | option
creation= "create" template-name | "create" "default" |
  "dynamic" template-name
addition= "add" template-name proto "port" port-number action |
```

```
  "add" template-name "subsystem" cmd-string ["shared"] |
  "add" template-name "use" template-name "if" condition
binding = "bind" ip-address template-name |
  "bind" condition ip-address template-name |
  "bind" ip-address "to" interface-name |
  "clone" template-name template-name
set     = "set" template-name "default" proto "action" action |
  "set" template-name "personality" personality-name |
  "set" template-name "personality" "random"
  "set" template-name "uptime" seconds
  "set" template-name "droprate" "in" percent
  "set" template-name "uid" number ["gid" number]
  "set" ip-address "uptime" seconds
annotate= "annotate" personality-name [no] finscan |
  "annotate" personality-name "fragment" ("drop" | "old" | "new")
route   = "route" "entry" ipaddr |
  "route" "entry" ipaddr "network" ipnetwork |
  "route" ipaddr "link" ipnetwork |
  "route" ipaddr "unreach" ipnetwork |
  "route" ipaddr "add" "net" ipnetwork \
                "tunnel" ipaddr(src) ipaddr(dst) |
  "route" ipaddr "add" "net" ipnetwork ipaddr \
              ["latency" number"ms"] ["loss" percent] \
              ["bandwidth" number["Mbps"|"Kbps"] \
              ["drop" "between" number "ms" "-" number "ms" ]
proto   = "tcp" | "udp" | "icmp"
action  = ["tarpit"] ("block" | "open" | "reset" | cmd-string | \
  "internal" cmd-string \
  "proxy" ipaddr":"port )
condition = "source os =" cmd-string |
  "source ip =" ipaddr | "source ip =" ipnetwork |
  "time " timecondition
timecondition = "between" time "-" time
option  = "option" plugin option value
```

The *cmd-string* and the *personality-name* are arbitrary strings enclosed with quotation marks. Variable expansion on the tokens *$ipsrc*, *$ipdst*, *$sport* and *$dport* is performed when executing the command string or when resolving the proxy address. Additionally, the environment variables *HONEYD_IP_SRC*, *HONEYD_IP_DST*, *HONEYD_DST_PORT* *HONEYD_SRC_PORT* and *HONEYD_PERSONALITY* are available, too.

If the *internal* key word is use, **honeyd** interprets the command string as Python module. These modules are executed within **honeyd** without forking a new process. As a result, internal scripts are very fast and cheap to execute.

The special keyword *tarpit* is used to slow down the progress of a TCP connection. This is used to hold network resources of the connecting computer.

If an IP address is not bound to a template, the actions specifi ed in the *default* template are executed.

Personalities need to be annotated before they are assigned to a template or an IP address.

The default fragment policy is to accept fragment and resolve overlaps in favor of old data. If the personality returns TCP timestamps, the default uptime is a randomly chosen between zero and twenty days.

The special *include* directive may be used to include other configuration files, for example to keep all personality annotations separate from the main configuration file.

All honeyd plugins can be configured via the configuration file. Each configuration option goes in one line, indicated by the *option* keyword. It is followed by three items: the name of the plugin, the name of the configuration option, and a value. The value can be either an integer, a float, or a character string. The options are picked up when honeyd reads the configuration file and can then be queried by the plugins.

-i *interface*
Listen on *interface*. It is possible to specify multiple interfaces.

-V|--version
Print version information and exit.

-h|--help
Print summary of command line options and exit.

--include-dir
For plugin development. Reports the directory in which **honeyd** stores its header files.

net The IP address or network (specified in CIDR notation) or IP address ranges to claim (e.g. "10.0.0.3", "10.0.0.0/16" or "10.0.0.5-10.0.0.15"). If unspecified, **honeyd** will attempt to claim any IP address it sees traffic for.

ROUTING TOPOLOGY

honeyd supports the creation of a complete network topology including routing. In order to enable the simulation of a network topology, a router entry point has to be configured with

```
route entry <IP address>
```

By adding a *network* to a router entry point, **honeyd** is told about the network addresses this entry point is responsible for. This enables multiple entry points into the routing topology.

Every *route add net* directive creates the specified gateway as a new router. In the case of tunneling, no new router is created, instead packets are `gre`(4) encapsulated and sent to the tunnel destination address. It is assumed that the tunnel destination address routes the encapsulated packets to a **honeyd** machine.

The virtual machines that can be directly accessed by a router are defined as network range in the *route link* command.

A link may carry attributes like *latency*, *loss*, and *bandwidth*. The *latency* specifies a constant delay for packets travelling across the link. The *bandwidth* on the other hand tracks the bandwidth related queuing delay for each link. If a packet is still being transmitted on the link then the queue delay for another packet is the propagating delay depending on the bandwidth plus the time for the previous packet to clear the link.

Unless the link is configured to drop packets between a configurable delay threshold, **Honeyd** currently assumes infinite buffer space, so use this option with care.

An address space can be made unrouteable via the *route unreach* command.

The router entry point is the first address that inspects a packet. The packet follows a path defined by the network topology until the current router has the destination IP address on its local network.

It is possible to integrate real machines into the routing topology. **honeyd** takes care of ARP requests and replies and encapsulates packets that go to external machines into ethernet packets.

External machines can be configured with the following command:

```
bind <IP address> to <interface name>
```

SUBSYSTEM VIRTUALIZATION

Subsystem virtualization allows you to run regular network applications under a virtual IP address controlled by **honeyd**. The application's network calls are intercepted and virtualized to the honeypot that they are configured to. As a result, all network calls that subsytem applications make appear to originate from the virtual IP address of a honeypot. This includes binding ports, accepting and initiating UDP and TCP connections. Raw sockets are not supported.

Subsystem are configured as follows

```
set template subsystem "/usr/sbin/httpd"
```

and are started as a separate process for every bound template. Applications started as a **honeyd** subsystem need to be dynamically linked in order to work under **Honeyd**.

It is possible to shared subsystems across different addresses if they are created with the *shared* flag. This allows a subsystem to bind to several virtual IP addresses and may also be used to increase the performance of subsystems across addresses.

DYNAMIC TEMPLATES

Dynamic templates give **Honeyd** the ability to change networking behavior based on several different conditions:

source address	The source address of the network connection determines which template is going to be used.
operating system	The operating system as determined by passive fingerprinting needs to be matched for the template to be activated.
time	The template is only being used between a certain time interval. This allows Honeyd to simulate machines being turned on and off.

A dynamic template can be created with the following command:

```
dynamic magichost
add magichost use windowsxp if source os = windows
add magichost use linux if source ip = 192.168.0.0/16
add magichost use invisible if time between 12:00am - 5:00am
add magichost otherwise use default
```

As an alternative, it is possible to use a short cut in the bind command to create dynamic templates:

```
bind source ip = 192.168.0.0/16 10.0.0.5 cisco
bind source ip = 10.0.0.0/8 10.0.0.5 juniper
```

In this example, the host on *10.0.0.5* behaves like a cisco router if it is contacted from IP addresses in the *192.168* network. If it is contacted from IP addresses in the *10* network, it behaves like a juniper router.

LOGGING

Honeyd has two different logging modes. The syslog facility is used to log connection establishment and termination including other relevant packet events. Most messages can be disabled when configuring `syslog.conf`(5) to drop all messages for the `LOG_DAEMON` facility if the log level is below `LOG_NOTICE`.

Services started by **honeyd** can cause the daemon to log data by sending information to *stderr*.

The second way of logging network activity is by using the **-l** flag. This causes **honeyd** to log all received packets in a human readable format. For UDP and TCP connections, **honeyd** logs the start and end of a flow including the amount of data transfered.

For logging any other information, it is suggested to run a separate intrusion detection system.

SCRIPTING WITH PYTHON

Honeyd supports internal service scripts that have been written in Python. To improve the performance of these services, **Honeyd** provides an event-driven model. The services need to indiciate when they are ready to read and when they are ready to write data. **Honeyd** keeps track of state that is provided to the Python scripts on every invocation.

The folowing example uses a Python script to implement a simple echo server:

```
import honeyd
import sys

def honeyd_init(data):
  mydata = {}
  honeyd.read_selector(honeyd.EVENT_ON)
  return mydata

def honeyd_readdata(mydata, data):
  honeyd.read_selector(honeyd.EVENT_ON)
  honeyd.write_selector(honeyd.EVENT_ON)
  mydata["write"] = data
  return 0

def honeyd_writedata(mydata):
  data = mydata["write"]
  del mydata["write"]
  return data

def honeyd_end(mydata):
  del mydata
  return 0
```

EXAMPLES

A sample configuration file looks as follows:

```
# Example of a simple host template and its binding
include annotations

# Set up the hosts
create template
set template personality "OpenBSD 2.6-2.7"
add template tcp port 80 "sh scripts/web.sh"
add template tcp port 22 "sh scripts/test.sh $ipsrc $dport"
add template udp port 53 proxy yournameserver:53
set template default tcp action reset
set template uid 32767 gid 32767

bind 10.11.69.2 template
set 10.11.69.2 uptime 1327650
```

A simple example of a routing topology:

```
route entry 10.0.0.1
route 10.0.0.1 link 10.2.0.0/24
route 10.0.0.1 add net 10.2.1.0/24 10.2.0.10 latency 10ms loss 3.4
route 10.2.0.10 link 10.2.1.0/24
```

FILES

`/var/run/honeyd.pid`	The PID of the current daemon.
`{prefix}/lib/honeyd/libhoneyd.so`	A shared library that can be preloaded to virtualize applications within **honeyd**.
`{prefix}/share/honeyd/nmap.assoc`	An association file to match **xprobe2** fingerprints against **nmap**.
`{prefix}/share/honeyd/nmap.prints`	**Nmap** fingerprints used by **honeyd** to impersonate operating system stacks.
`{prefix}/share/honeyd/xprobe2.conf`	**Xprobe** fingerprints used by **honeyd** to impersonsate the ICMP section of operating system stacks.

SEE ALSO

`arpd`(8)

AUTHORS

Niels Provos <provos@citi.umich.edu>