Blackhole operation

Jacques Gélinas
jacques at croesus.com

September 30, 2015

Abstract

Here are some explanations about configuration and operation of the blackhole components. It will be highlighted that most of the configuration occurs in the blackhole server while very little is done in the peripheral components (horizon and wormhole). So it becomes possible to deploy the same configuration for all peripheral components.

Contents

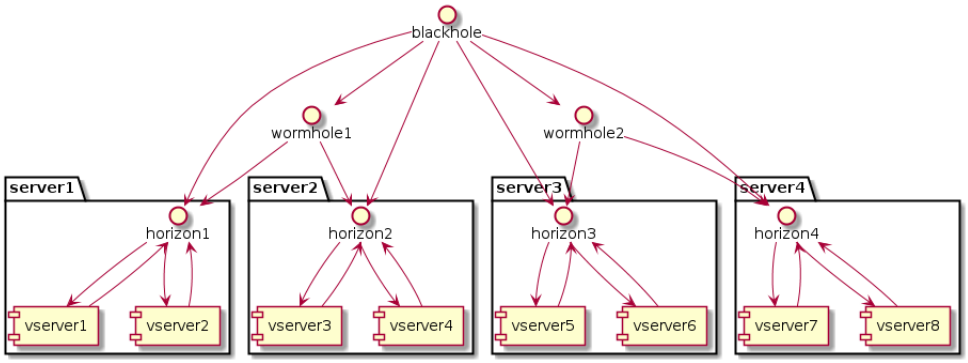
1 Rules file

Each component has a configuration with a strange name: /etc/**component-name**-rules.sh. Those configuration file are indeed little scripts talking to the component using a Unix domain socket (using the corresponding utility **component-name**-control). In general, configurations of the blackhole component may be extracted from other systems. For example, the list of vservers (and their assigned IP numbers) running of a host may be extracted using the **vipalias** utility. So the rule file can extract some configurations and specify some exceptions.

2 Connection rules

2.1 Standard mode

The standard mode is illustrated by the following diagram. This represents what is called est-west traffic, or intra datacenter.



Using the blackhole system, any vservers can talk to any vservers, yet no outside connection are made except by the wormholes and the blackholes. While this represents the bulk of the connectivity in a data center, there must be a way to reach those vservers from outside and also allow some vservers to reach the outside.

A connection request, sent to the **blackhole** server looks like this:

```
connect vserver-name ipN port linkID
```

vserver-name Name associated with the source IP of the connection made to the horizon server.

ipN The horizon server may bind on several IP numbers (generally 192.168.2.1, 192.168.2.2,...). The IPs are meaningless. The first IP is called ip0, the second ip1 and so on. Connection rules in the blackhole are built using those symbolic names. This allows a vserver to connect to several services on the same port (for example, connect to multiple database on port 4101).

port This is the actual TCP destination port.

linkID It is a number used for the rendez-vous protocol. This is also used to reject a connection.

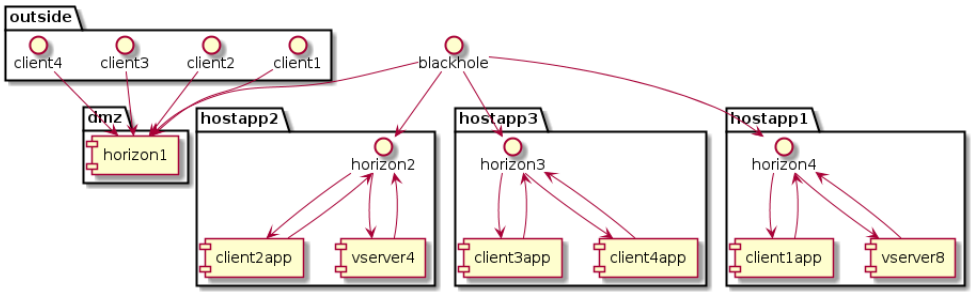
2.2 Open network mode

Normally, the **horizon** server reject all connections made on its application ports which can't be associated to a vserver. Using the **–open_network** command line option, the rule is relaxed. If the horizon server can't associate the IP number to a name, it will pass the connection request to the **blackhole** server as is.

```
connect source-IP-number ipN port linkID
```

When the blackhole server receives such a request (The first argument is not a name, but an IP number), it tries to identify the network name (see the **network** blackhole configuration command below). It rewrites the input using the network name and tries to find a matching rule.

This mode is shown in the following diagram.



The following table presents how the information is transformed to finally produce a connection rule.

		Rule left side				Rule right side		
Where	Step	Host	Source IP	target IP	port	host	vserver	port
horizon	1			208.10.10.1	80			
horizon	2		207.253.4.2	208.10.10.1	80			
horizon	3		207.253.4.2	client1	80			
horizon	4	Sends **connect 207.253.4.2 client1 80 linkID** to blackhole						
blackhole	5	dmz	207.253.4.2	client1	80			
blackhole	6	dmz	internet	client1	80			
blackhole	7	dmz	internet	client1	80	hostapp1	client1app	80
blackhole	8	Uses the **rendez-vous** protocol to establish the link						

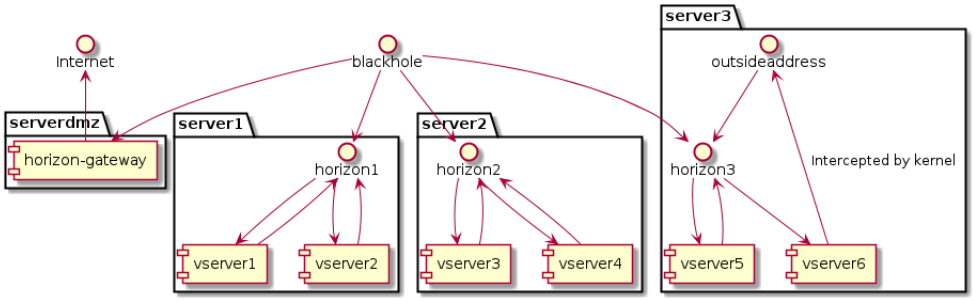
1. Using the command line option –**bind 208.10.10.1,80,client1** on the horizon server, we are telling it to setup a TCP socket on port 80, bound to the IP 208.10.10.1 and call it logically client1.
2. A connection is made on that socket from IP 207.253.4.2.
3. The horizon server looks at the target of the connection and translates the IP into the logical name provided on step 1.
4. It sends a **connect 207.253.4.2 client1 80 linkID** command to the blackhole server, using the connection made by the blackhole server.
5. The blackhole server identifies the horizon server. The horizon server does not have to tell its name.
6. The blackhole lookups 207.253.4.2 in a list of network definitions (network/netmask defined in its /etc/blackhole-rules.sh configuration file) and finds **internet**.
7. The left side is now complete. The blackhole performs a lookup and finds the corresponding right side.
8. Using the **rendez-vous protocol** it establishes the final link.

2.3 Transparent proxy mode

Normally, the blackhole system is used to control access between vservers in a network. These vservers are not reachable from the Internet and can't access it either. Sometime, you may want to let a vserver access some other networks (or the Internet) without restriction. Using **iptables** rules such as the following one, it is possible to intercept connections made outside the host and redirect it to the **horizon** running on the host.

```
iptables -t nat -A OUTPUT -s vserver1-IP -p tcp --dport 80 \
-j REDIRECT --to-port 80
iptables -t nat -A OUTPUT -s vserver1-IP -p tcp --dport 443 \
-j REDIRECT --to-port 443
```

The horizon server understands that the connection was not meant for itself, so rewrite the connect request (to the blackhole server) by entering the original target IP number instead of the symbol **ipN**. When the blackhole server receives this request, it finds out that this is not a symbolic ipN name, but a complete IP number. It tries to identify a matching network and rewrite the request using the network name. It then attempts to find a matching rules. The following diagram illustrates how it is used.



Using the iptables rules above, applied on any of the host (server1,2,3), allows any vservers to reach the internet, under complete control of the blackhole system. At least one horizon server running with –**open_client** option, usually located in the **DMZ**, acts as the final proxy to reach the internet.

You can even write a single generic rule that intercept anything originating from the vservers. We assume here that all vservers are installed using an IP in the 192.168.3.0/24 network. Even if all connections will be redirected to port 80 on the loop-back, the **horizon** will retrieve the original destination port and submit the proper request to the **blackhole**.

```
iptables -t nat -A OUTPUT -s 192.168.3.0/24  -p tcp -j REDIRECT --to-port 80
```

The listening horizon must be listening on port 80, on the loop-back. So if your vservers are on the loop-back, the horizon must be listening on 127.0.0.1. This is how the iptables **REDIRECT** feature works. The horizon must also use the –**clientbind** option to make sure it is not binding/using an IP from a vserver (when talking to the vserver), avoiding being redirected by the iptable rule above.

Here are some rules showing how it is done.

```
# We define two horizons
# horizon1 has a vserver named webserv
blackhole-control horizon horizon1
# hproxy is an horizon which can reach the internet
# This horizon uses the --open_client option
blackhole-control horizon hproxy
# we define a network name with ip and netmask
blackhole-control network internet 0.0.0.0 0.0.0.0
# Then the rule allowing webserv to reach the internet
blackhole-control rule horizon1 webserv internet 80 hproxy KEEP 80
```

Note the keyword **KEEP**. This tells the blackhole to rewrite the right side of the rule by duplicating the original IP number received/intercepted by **horizon1**. Then hproxy will receive a connect request asking to connect to the proxy IP on port 80.

Note that the –**dport** iptables option is ... optional. If you omit the option, then using a single iptables rule, you redirect all outgoing TCP connections to one listening horizon socket. The horizon will use the kernel to retrieve both the destination IP and port. This info will be passed up to the blackhole. It will route the connection or reject it.

2.4 Showing the original source IP to the vserver

It is sometime useful to present the original IP number to the vserver service. This makes logging more useful. Sometime, it is essential. By default, all connections made to a vserver will occur from the IP of the horizon. If you expose a vserver service on the Internet using blackhole, then the vserver won't be able to tell who is who: /etc/hosts.allow and friends become useless.

The horizon is able to **fake** the source IP when connecting to a vserver. You do this by using a special syntax in the **blackhole rule**.

```
blackhole-control rule horizon1 vserver1 internet 80 webserv1 vserver2/SOURCE 80
```

SOURCE is a special keyword. It is replaced by the source IP obtained on the left side of the rule. You can write a fixed IP after the slash if you want, forcing connections from specific rules to appear as coming from this fixed IP.

To make this work, you need some **iptables** rules on the host running the vserver. Here they are. We assume that all vservers are running on the **192.168.3.0/24** network.

```
TABLE=mangle
iptables -t $TABLE -N DIVERT
iptables -t $TABLE -A PREROUTING -p tcp -m socket --transparent -j DIVERT
iptables -t $TABLE -A OUTPUT --src 192.168.3.0/24 -p tcp -j MARK --set-xmark 0x1/0xffffffff
iptables -t $TABLE -A DIVERT -j MARK --set-mark 1
iptables -t $TABLE -A DIVERT -j ACCEPT
ip rule add fwmark 1 lookup 100
ip route add local 0.0.0.0/0 dev lo table 100
echo 0 >/proc/sys/net/ipv4/conf/lo/rp_filter
```

2.5 Using IP lists

IPlists are definitions holding list of IP numbers and networks. IP numbers are stored in an indexed set, so can hold efficiently thousands of entries. Networks are defined using network and netmask. They are sorted by the number of significant bits in the netmask, so smaller networks are first.

```
blackhole-control network network_name 192.168.1.0 255.255.255.0
```

IPlists are used in connection rules. They are either in the **vserver** part or the **target** part. In **–open-network** mode, the horizon accepts connection from unknown IPs (IP not associated with a vserver). In this mode, it passes the IP number directly in the connection request.

```
connect x.y.z.w target port ID
```

When used in transparent proxy mode, the target IP is passed directly in the connection request.

```
connect vserver x.y.z.w port ID
```

The blackhole server find a matching iplist and uses that to select the proper connection rule.

2.5.1 Large iplists

Iplist are defined using the **iplist** and **network** command of the blackhole-control utility. Executing blackhole-control several thousand times takes fairly long. Using the **–pipe** command line option, you can make this tasks quicker. Here is a script doing just that.

```
(cat file_with_ip | while read ip
        do
                echo iplist $1 $ip
        done) | blackhole-control --pipe
```

2.5.2 IP Lists and rules and the internet

Iplists are searched in alphabetical order (case sensitive). So you generally put smaller (more specific) lists first (using a name starting with an **a** for example).

You will eventually have a list called **z_internet** with this network definition:

```
blackhole-control network z_internet 0.0.0.0 0.0.0.0
```

This list matches everything. So by naming it z_internet, you make sure it is checked last. Here is how blackhole works with iplists. When it finds an IP number in a **connect** request, it searches through all iplists. It finds everyone that matches: Not only the first one, everyone. Then it searches a connection rule using the first matching iplist, then the second, ... until it finds one rule.

2.5.3 IP list example

Here is an example showing how you can implement the **host allow** feature using blackhole. You have serverdmz reachable from the internet and listening on two public IP numbers. We will call them publicA and publicB. When we connect on port 22 to IP publicA, we expect to end on vserverA. If we connect to IP publicB, we expect to end on ververB. But this is true only if we originate from IP 100.0.0.1 or 100.0.0.2 All other IPs end up in the devnull service.

```
blackhole-control iplist lista 100.0.0.1
blackhole-control iplist listb 100.0.0.2
blackhole-control network z_internet 0.0.0.0 0.0.0.0
blackhole-control rule serverdmz lista publicA 22    server1 vserverA 22  # rulea
blackhole-control rule serverdmz listb publicB 22    server1 vserverB 22  # ruleb
blackhole-control rule serverdmz z_internet publicA 22   server1 devnull 8022 # rulec
blackhole-control rule serverdmz z_internet publicB 22   server1 devnull 8022 # ruled
```

Here are different cases supported:

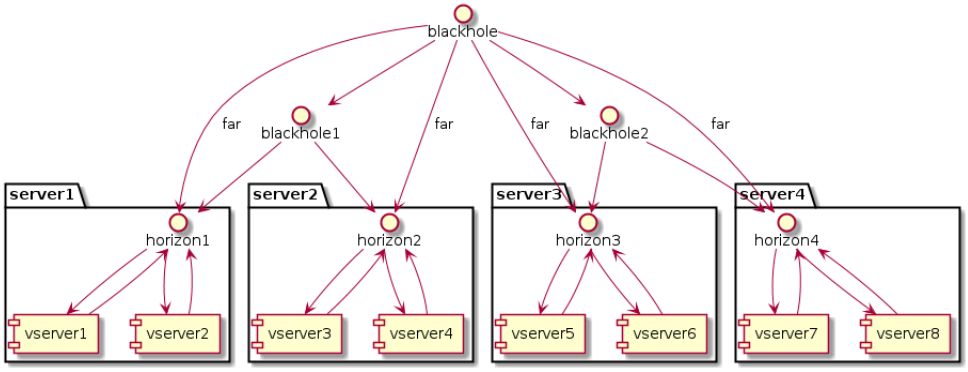
Case	Connect from	Connect to	Matching iplists	Using rule
	Typical usage			
A	100.0.0.1	publicA	lista,z_internet	rulea
B	100.0.0.2	publicB	listb,z_internet	ruleb
C	100.0.0.3	publicA	z_internet	rulec
D	100.0.0.3	publicB	z_internet	rulec
	Also expected, see explanation			
E	100.0.0.1	publicB	lista,z_internet	ruled
F	100.0.0.2	publicA	listb,z_internet	rulec

For cases A and B, we have two matching iplists and two matching rules. The first one is selected.

For cases E and F, we have two matching iplists, but only one matching rule (**rulec** or **ruled**). In case E the IP 100.0.0.1 matches **lista**, but there is no rules using **lista** and **publicB** together. So the only matching rule is **ruled**.

2.6 Sub-blackholes

It is possible to install **blackhole** servers closer to the action. Those sub-blackhole are managing connection rules for the horizon they are connecting to. Whenever they get a connection they can't resolve, they pass the request to the upper blackhole. Ultimately, this one will either reject or establish the connection.



The upper **blackholes** connect to the horizon servers specifying they are **far**. It means the horizon server won't send **connect** requests to them, unless there are no **near** blackhole servers available. In general, when installing a network like this, you make sure the connection rules in the sub-blackhole are a subset of the rules in the upper blackhole. This way, connections are always serviced equally if a lower blackhole goes away.

3 The blackhole server

All configurations of the blackhole server are entered dynamically. It does not have to be restarted: All you need is a reload on the service. It runs as user **blackhole**. It does not have to be root. When used in daemon mode, it changes to user blackhole and disconnect itself from the console. The blackhole service may run in a vserver as well: There is no need to have it on the host.

3.1 The /etc/blackhole-rules.sh file

The rules.sh file is a configuration file that is indeed a shell script. This was done because most of the blackhole configuration may be extracted from other sources. So instead of replicating the information in the rules file, you simply perform some extractions and formatting. The rules file ends up issuing a bunch of **blackhole-control** commands to load the configuration in a running blackhole server.

Note that it is possible to erase and load new rules in the blackhole server without disrupting the service. See the **pause** and **resume** command.

3.2 The blackhole-control utility

The blackhole-control utility simply copies its command line arguments to the blackhole server using a Unix domain socket (usually /var/rum/blackhole/blackhole.sock). Here are the supported blackhole commands:

- Basic configuration

  allow Add one IP number to the list. Blackhole servers from that list are allowed to connect. The list is empty at startup.

  blackhole Defines a sub-blackhole. You must specify the name of the sub-blackhole server. It must resolves to an IP number. The blackhole server will try to establish a control connection with this sub-blackhole every 5 seconds

  hole Name one horizon server reachable from one wormhole server. To be useful, you must provide at least 2 **hole** commands for one wormhole server. Note that you can also associate a wormhole with a network name. This is used in the transparent proxy mode.

  horizon Define an horizon server. You must specify the name of the horizon server. It must resolves to an IP number. The blackhole server will try to establish a control connection with this horizon every 5 seconds.

wormhole Define a wormhole server. You must specify the name of the wormhole. This name has to resolve to an IP number. The blackhole server will try to establish a control connection with this wormhole every 5 seconds. A wormhole definition is useless unless some **hole** commands are issued.

- Rule definitions

arule Add an alternative target to a connection rule. It is used for clustering.

atemprule Like arule, but add a temporary alternative.

rule Definition of a connection rule. The definition is made of a left side (the connection request) and the right side (where to connect).

Left side	Horizon name (origin of the connect request) Vserver-name or IP number (–**open_network** mode) ipN or IP number (transparent proxy mode) TCP port
Right side	Target horizon-name Target vserver-name Target TCP port

temprule Same format as a **rule** command, but adds the expiration date and time. The date is supplied in **yyyy/mm/dd** format. The time is supplied in **hh:mm:ss**. Note that **temprule** are not erased from the blackhole server. They just become expired. Connection attempt matching those rules will be rejected. The **status** command is explicit about expired rules.

- Control/Reloading

erase-rejects Erases one entry in the reject list.

pause This stops the blackhole server. It won't process connection requests until the **resume** command is sent. All configurations may be changed without creating invalid behavior in the mean time. Active connections (vserver to vserver) are not impacted. If the blackhole server is paused, it will show in the **statuserr** output.

readsecrets Tells the blackhole server to re-read the secrets in the file defined by the –secretfile command line option (of the blackhole server).

rejectstat server vserver ipname port nbtry lasttry
is used to re-inject statistics about rejected requests. This is normally used when you restart (not reload) the blackhole service. It allows you to keep the statistics around.

reset-blackholes Erases all sub-blackhole server definitions. Connections to all sub-blackhole servers will be closed. Active connections (from vserver to vserver) are not impacted.

reset-horizons Erases all horizon server definitions. Connections to all horizon servers will be closed. Active connections (from vserver to vserver) are not impacted.

reset-rules Erases all the connection rules (**rule** and **temprule**), the list of allowed blackhole IPs, the **hole** and the **network** definitions.

reset-wormholes Erases all wormhole server definitions. Connections to all wormhole servers will be closed. Active connections (from vserver to vserver) are not impacted.

resume Resumes normal operation.

rulestat Inject connection statistics on rules. This is used when we perform a reload. All rules are flushed and new ones are put in place. The statistics are lost. So before doing so we extract the statistics using the **status** command and then we inject them back here. The command requires the seven parameters describing a rule plus the number of connections and the last connection date.

- IP Lists (black list)

iplist adds an IP number in a list. You must provide

  – The iplist name

- An IP number

list-iplist prints the content of an **iplist**. You must provide the iplist name.

network Add/defines a network in an **iplist**. For each network, you provide

- The iplist name
- A network ipv4 address
- A netmask

reset-iplist erases the content of an **iplist**. You must provide the iplist name.

- Load balancing

setaltcon tells the blackhole how many connections are currently active on other blackholes for this target.

setweight sets the relative weight of this target (default 100).

- Stats

connectload shows current target connections and configuration. A target is the right end side of a configuration rule. The 3 numbers are: connections, weight and alt_connections. The connections is the number of active connections through this blackhole. The weight is used for load balancing. The default weight for a target is 100. A lower weight means that proportionally, the target will receive less connections.

connections Prints all the active connections (from vserver to vserver).

rejects Prints all the rejected attempts. The blackhole server logs 1000 different attempts (the left side of a connection rule). It logs the occurrence and the last date.

status Prints a status, human readable.

statuserr Prints all error condition. Normally, it prints nothing. It is a good way to monitor the health of the whole system.

4 The horizon server

4.1 The /etc/horizon-options.conf file

This file provides the basic option of the horizon server. It is not possible to change those options without restarting the horizon server.

–**master IP,port** Tells on which IP and port to bind. This is used by the blackhole and wormhole servers to reach the horizon server.

–**bind IP,port** This option may be repeated. It is used to enumerate the application ports. The horizon server will listen for connection on these IP,port.

4.2 The /etc/horizon-rules.sh file

The rules file is a configuration script. It is mainly used to associate IP number to vserver name. It generally parses the output of the vipalias command. It specifies the IP number of the wormhole and blackhole servers. The script simply runs the horizon-control command to pass configurations to the horizon server.

4.3 The horizon-control utility

The horizon-control sends commands to the horizon server. They are

allow ip-number Tells the IP number of a blackhole or wormhole server

connections Prints the current connections

nameip ip name
: associates a name to an IP number. By default, horizon assigned ip0, ip1 and so on to the various IP it is listening to. Using the nameip command, you can assign meaningful names, which will be used in the blackhole rules.

reseterror clears the last error recorded. This last error is transmitted using the ping/pong protocol to the blackhole.

readsecrets reads the content of the file defined by the –**secretfile** command line option of the **horizon** server.

status Human readable status dump of the horizon server

vserver vserver-name ip-number Associates an IP number to a vserver name

n_clear clears the temporary list of vserver names in the horizon service.

n_commit commits the temporary list of vserver names to the running list.

n_vserver name ip
: does the same as the vserver command, but write to a temporary list. Once the new list is completed, the n_commit command make this list effective. This is a way to update the configuration without causing service outage.

5 The wormhole server

5.1 The /etc/wormhole-rules.sh file

This file is a configuration scripts. It is used to specify the IP numbers of blackhole servers. All other connections will be rejected.

5.2 The wormhole-control utility

The wormhole-control sends commands to the wormhole server. They are

allow ip-number Tells the IP number of a blackhole

connections Prints the current connections

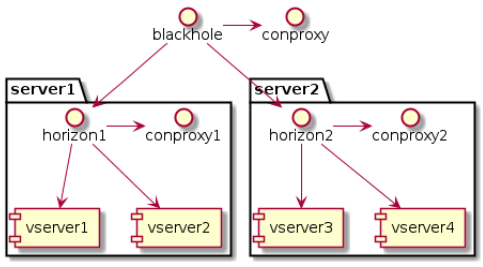
status Human readable status dump of the wormhole server

6 The conproxy server

There is no configuration file for this service. The conproxy server is a high performance single thread non-blocking connection proxy. It does not establish any connection itself. It simply copies back and forth between two sockets. It can handle thousands of connections. It normally operates as a companion to the blackhole, horizon and wormhole servers. It is optional. If not running, the blackhole system works, but is less efficient: Each connection is handled by a sub-process.

The conproxy listens on a unix domain socket. The various servers (blackhole, horizon and wormhole) pass new connection handles using this unix socket. It is used by default by the blackhole, horizon and wormhole server. There is nothing to configure.

This is a typical installation.



The conproxy has the ability to log the communication. It can log new connections as well as existing connections. This is useful when one is trying to figure what is going on between two hosts...

The conproxy seldom needs to be restart. In general, you do it to install a new version. Restarting the conproxy is safe. The old conproxy process will continue to run until all the connections it is handling have ended.

6.1 The conproxy-control utility

The conproxy-control utility sends commands to the conproxy server. They are

connections It lists all connections. Each line outputs the following information:

N,M It copies from handle N to M. Expect to find a line with M,N in the output as well.

bytes Number of bytes copied from N to M.

start Start time of the connection (in seconds since 1970).

description A description of the connection.

flushlog executes a fflush() on each log file.

log off Forget all logging strings.

log off some string ...
Forget the logging definition for "some strings ...". Logging is stopped for connection matching this string.

log on some string ...
Turn logging on (copy the bytes received on that socket to a log file). Logging will be done on connection with a description holding "some string ...". Logging starts immediately on already opened connections.

quit requests the conproxy server to end immediatly.

quitlast requests the conproxy server to end when the last connection ends.

stamplog writes a time stamp in active log files.

status reports the number of active connections and the active "log on" strings.

7 The blackhole-devnull service

This is a service used to discard connections. Normally, all connection rules point to a server/vserver. If there is no matching rule, the connection is simply rejected and you can see a trace using the **rejects** command of the blackhole-control utility.

But you may want to keep the **rejects** feature to detect un-managed (no matching rule) connections. Connection not matched by any rule is either a flaw (something missing), or something new (You have been hacked). Using the blackhole-devnull service, you explicitly throw away what you don't want to serve. By default, this service does not do much. It simply wait for one minute and then hang up the connection. It maintains some stats about connections and it can execute a script when a new IP number is seen. This script may be used to maintain a blacklist for example.

Here is a typical application:

- You allow access to ssh (port 22) for IP 100.0.0.1 and 100.0.0.2.
- You want to block access to every other IP.
- You want to collect a list of IP attempting connection to port 22.
- You want to use this list to block access to port 80.

Using this scheme anyone who knock on port 22 is blacklisting itself. Here are the rules to do that

```
blackhole-control iplist a_sshok 100.0.0.1
blackhole-control iplist a_sshok 100.0.0.2
blackhole-control network z_internet 0.0.0.0 0.0.0.0
blackhole-control rule serverdmz a_sshok publicA 22 server1 vservera 22 # Allow access to ssh
blackhole-control rule servervmz z_internet publicA 22 server1 devnull/SOURCE 8022 # Reject and collect st
# Allow control to port 80, except for blacklisted IP
blackhole-control rule serverdmz a_black publiciA 80 server1 devnull/SOURCE 80 # Reject and collect stats
blackhole-control rule servervmz z_internet publicA 80 server1 vservera/SOURCE 80 # Serve normally
```

In the devnull service, you use the –**notify** option to assign a script called whenever a new IP number is seen. This script records the offending IP and immediately register it in the **a_black** iplist. The script looks like this.

```
#!/bin/sh
if [ "$2" = "8022" ]; then
        echo $1 >>/var/lib/devnull/devnull.log
         /usr/sbin/blackhole-control iplist a_black $1
fi
```

8 The findproc service

This is a service used by the **horizon** to identify the process (program name/path), user and group at the origin a connection. When a connection is accepted by the horizon, if the –**findproc** option is active (normally set in the **/etc/horizon-options.conf** file), then the horizon connects to the findproc service using a unix domain socket to find information about the owner of the socket (the client).

For now, the findproc service is **vserver** aware. It does not support the other container types.

When you use this option, the horizon will include in the request made to the blackhole the information about the client. Instead of just sending the vserver name in the connect request, it sends a line looking like that:

```
connect vservername,program-path,user,group ipN port
```

When you write connection rules, you are free to make them as precise as you want. For example, you may write a rule allowing connection from a vserver by any user, but only done using one program. The blackhole takes the request as is and perform a search. If it can't find a rule, it remove one part, starting from the end (it removes the group first) and try again. It removes part after part until there is only the vserver name remaining.

The ability to limit connection to a specific program is especially useful when the vserver runs on a read-only filesystem: The vserver can't lie. It is a good way to catch errand vservers (abused/hacked). For example, a vserver connected to a database will normally only connect to it using one program. Once abused, the intruder will use other programs to access the database. These accesses will fail and could easily be monitored using the **blackhole-control rejects** command line.

9 Running multiple blackhole servers

You can run multiple blackhole servers on a network. Since the blackhole is the key to authorize and route a connection, it is essential that it runs at all time. Using wormholes, it won't be an I/O bottleneck. But it has to be there for any new connections. So you can run several blackholes. Here are the rules:

- They must share the same configuration. Exact same configuration.
  - Same list of horizon servers
  - Same rules
  - Same network definitions
  - But they may use different wormholes
- Blackhole servers are **pinging** the horizon servers every 5 seconds (a message on the control connection). The horizon has to replied.
- When an horizon server performs a connection requests, it will send the request to the last blackhole server having pinged it. So if a blackhole server dies, within 5 seconds, another one will become the preferred one (as seen by horizon servers).
- You can run as many as you want.

10 Extra tools

10.1 The udpproxy utility

This utility was created to proxy UDP packet using the blackhole system (which only handle TCP connection). So each UDP request opens one TCP connection, send the data and wait for the answer. It is mostly use to proxy DNS request. Some servers are completely isolated, but once in a while, may need to access a DNS...

11 Experimenting with blackhole

The **blackhole** project is a game changer. You certainly don't have to change everything in a datacenter to make use of it. But experimenting with it is a good idea. It is easy to fiddle with it on a single workstation. A script named **test.sh**, part of the source distribution, is provided to help you create the setup and then play with it. To test the project, just open a terminal windows with a bunch of tabs (10 actually).

Note that you do not need to be root to run the test, except to setup some IP aliases and modify /etc/hosts. Here is what you must do in each tab:

11.1 To prepare the test environment

Enter the blackhole source directory and do

```
make
su
./test.sh root
edit /etc/hosts as suggested
exit
```

You do not need to be root anymore. Some IP aliases will be installed on the loopback (Make sure it is not incompatible with your network).

11.2 The tabs

horizon1 The following command starts an horizon server in debug mode, setting its UNIX domain socket as /tmp/hozizon1.sock

```
./test.sh horizon1
```

horizon1 control The following command tells the first horizon to allow connections from various IP. It also provides the IP numbers associated with vservers (you don't need virtual machines to perform the tests). It simply associates names with IP aliases.

```
./test.sh horizon-control1
```

horizon2 The command starts another horizon server using /tmp/horizon2.sock socket. It is started with the option **–conproxyport none** to prevent the use of the conproxy server. It is also started using the **–open_network** option.

```
./test.sh horizon2
```

horizon2 control Like the horizon1 control.

```
./test.sh horizon-control2
```

blackhole Starts the blackhole server in debug mode

```
./test.sh blackhole
```

blackhole control Provides the rules to the blackhole server. This includes:

- The horizon server definitions
- The wormhole definitions
- The wormhole connectivity
- The sub-blackhole definition
- The connection rules

```
./test.sh blackhole-control
./test.sh blackhole-resume
```

Once in a while, you can do the following commands:

```
./test.sh blackhole-rejects
./test.sh blackhole-status
./test.sh blackhole-statuserr
```

conproxy Starts the conproxy server with the /tmp/conproxy.sock socket.

```
./test.sh conproxy
```

It is interesting to note that conproxy is not required. Each component (blackhole,wormhole,horizon) may operate without it. You can stop this service at any time and new connection may be established: Try it! If you kill it (control-C), existing connections will die.

tcpecho This is a small multi-user echo service. It simply echoes back whatever it receives. The following command starts two tcpecho servers, one listening on TCP port 18001 and the other on TCP port 18002.

```
./test.sh tcpecho
```

Once connected to it (see the **tests** tab), you can do the following

quit ends this session.

status prints the list of active connections. It shows the handle numbers. This is the argument needed with the start and stop commands.

stop N stops listening.

start N re-starts listening.

Type anything and see it echoed to you, with the prefix **echo**.

allconnections Keep this tab to execute the following command:

```
./test.sh allconnections
```

This connects to all components and requests the list of active connections.

tests You perform your various tests here. Uses the **allconnections** tab above to understand what is going on. You can easily track the path followed.

./test.sh testok1 Connects to the tcpecho server on port 18001. This should pass through horizon1, blackhole and horizon2 server.

./test.sh testok2 Connects to the tcpecho server on port 18001. This should pass through horizon2, blackhole and horizon1 server.

./test.sh testok3 Connects to the tcpecho server on port 18002. This should pass through horizon2, blackhole and horizon1 server. This tests the **−open_network** mode.

./test.sh testok4 connects to the tcpecho server on port 18001. This tests the ability for one horizon server (horizon1) to connect vservers on the same machine. The connection is accepted by the blackhole. It finds out the source and destination are on the same physical server. It requests the horizon to establish the link.

./test.sh fail1 Connection is rejected. the **./test.sh blackhole-rejects** is used to see which connections were rejected.

./test.sh fail2 same.