Blackhole project

Jacques Gélinas
jacques at croesus.com

September 16, 2015

Abstract

The **blackhole** project is a solution to simplify routing and firewalls. It simplifies yet increases security and flexibility. It is meant for datacenters. It allows communication between independent networks using a central configuration (who can talk to who). It does so while making all routing and firewalling simple, restrictive and uniform across all servers and firewalls.

Contents

1 Introduction

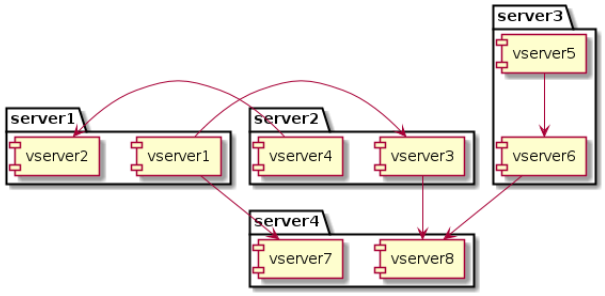
As requirements keep adding in a datacenter, it is difficult to maintain a correlation between physical location and logical role. What started as a set of specialized networks installed in their own rack, ends up with a mess: As racks fill, servers for various projects are installed in unrelated racks. We are facing with either a spaghetti of network cables running from one rack to the other, or meaningless firewall rules and routing exceptions.

The blackhole project starts with a clean state: No one talks to no one, only minimal uniform requirements (talk to DNS, NTP, and listen to SSH for management). Then from a central location, we define rules showing who can talk to who, on which TCP port. All specific configurations are centralized and all distributed configurations are uniform and minimal.

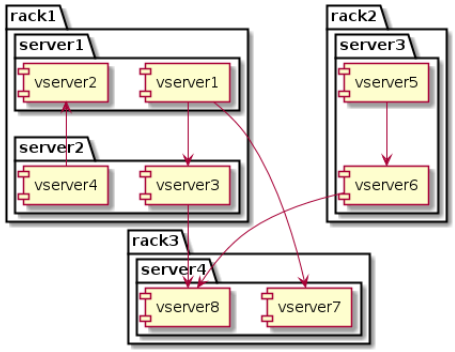
2 A quick view

Here is a schema explaining the concept. The arrows are showing the direction of the TCP connections.

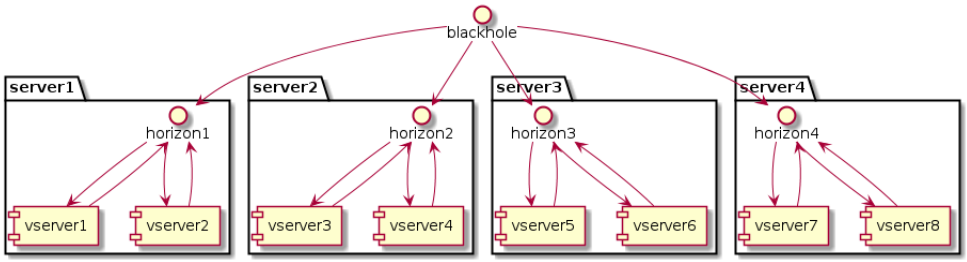
First a typical network showing relations between servers hosting vservers:



Or the same, showing the servers in racks.



Then, the same servers using the **blackhole** system. The arrows represent the directions for all possible TCP connections. Yet using the blackhole system, you can establish logically all the connections in the above drawings.



3 What is possible

Ideally, we want a controlled way to let any process talk to any other process, wherever they are on the physical network. But this makes networking more complex, introduces nightmare in firewalls and routers. Networks are generally designed logically, putting stuff talking together, together. But as the needs evolve, as the number of machines grows, we end up with all kinds of exceptions. Further, all these exceptions are carried using extra routing and by punching extra holes in firewalls. It becomes tedious not only to add those exceptions, but also to make sure those extra configurations are not creating security issues. It becomes even more tedious to remove those configurations once the exceptions are not needed.

In the end, all configurations become more and more complex and seldom return to the original state.

4 How it works

The **blackhole** project is a very simple tool to allow communication from anywhere to anywhere in your network using a central ruleset. Because of its design where all TCP connections originate from central locations, it makes routing and firewalling simpler. Because of the one-way connection concept, it is highly compatible with NAT.

4.1 Components of a network

A network is composed of the following parts:

- Host: A host is a physical machine usually running services. Often the host is running virtual machines which in turn are running services.
- Services: Services or applications either accept TCP connections, or initiate connections to other services usually located in other hosts (or other virtual machines).
- Rack: A rack is an enclosure holding together hosts that are generally related (security wise, function wise)
- A management network: This network can reach any host. Host can't connect to the management network.

4.2 Blackhole requirements

Here are the requirements for the blackhole project. In general, any datacenter already implement this.

- Hosts may be reached on a given TCP port (default 8000) from a central location.
- Hosts may be reached on the same TCP port from some other locations to optimized bandwidth and redundancy.
- Virtual machines must be able to reach some specific TCP ports (defined as needed) on their parent host. Those ports correspond to the various services running on the datacenter (SQL, HTTP, HTTPS, ...).

While not a requirement, stricter firewalling may be implemented easily. No particular connectivity between hosts is necessary. Ideally, one may implement the following rules:

- Host can't talk to each other at all.
- One virtual machine on one host can't talk to other virtual machines, including those on the same host.

4.3 Components of the blackhole system

There are two essential components: the **blackhole** server and the **horizon** servers. An extra component may be deployed to increase performance and add redundancy: The **wormhole** servers.

4.3.1 The blackhole server

This is the center piece. It is the only one with configuration telling who is allowed to talk to who. It receives all connection requests and decides how to handle or reject them. The **blackhole** server usually runs on a server which has connectivity to all hosts. This usually is a management server.

Also, note that there could be several **blackhole** servers for redundancy. They are all sharing the exact same configuration.

The **blackhole** server does not accept any TCP connection. It has a unix-domain socket for configuration and control. It establishes connections with **horizon** and **wormhole** servers.

4.3.2 The Horizon server

This is running on all hosts (usually running virtual machines). It sits there and open two kinds of TCP ports. It listens on a port to allow the **blackhole** server(s) to connect (and optionally the **wormhole** servers). It listens on other ports (application ports) to allow virtual servers (running on the same host) to connect to it. Virtual servers do not connect anywhere else. Upon request from the blackhole or wormhole server, it connects to ports in local vserver.

The applications ports are usually bound to IP aliases on the loopback so they are not reachable from outside.

By default (unless the **–open_network** option is used), all connections on the application ports not coming from an identified IP (a local vserver) are rejected. This is the only policy it enforces.

4.3.3 The wormhole server

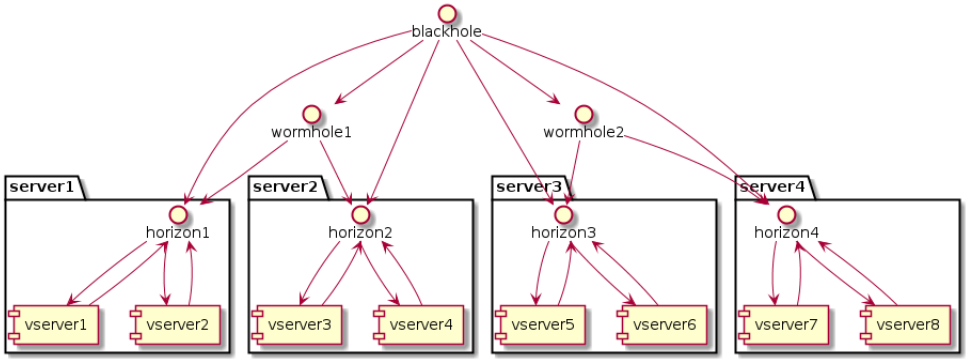
By default, all connection requests are going to the **blackhole** server. Since it has connectivity to all **horizon** servers, it can establish a link between any virtual servers in the datacenter. It acts as a proxy between anything to anything.

It obviously may become an I/O bottleneck. While you can run several **blackhole** servers in a datacenter, all sharing the same configuration, there is another way to optimize the performance. The **wormhole** servers are logically deployed to be closer to the action and supports connectivity between virtual machines in the same network or neighbor networks.

Wormhole servers have minimal configuration. Mostly they are told who can't connect to them. This is the list of IP used by the **blackhole** servers. Only **blackhole** servers connect to **wormhole** servers.

The **blackhole** server maintain a list of **horizon** servers reachable from one **wormhole** server. Based on that, it can tell which **wormhole** may be used to connect two virtual servers. The **wormhole** have no configuration like this.

Here is a sample diagram showing the network relation



In this diagram, we see that **wormhole1** may be used whenever one vserver in **server1** connects to a vserver in **server2** (or the opposite). And **wormhole2** is used for connection between **server3** and **server4** vservers. All other cases are managed by the blackhole itself. **Wormholes** may be seen as an optimized, but optional path between vservers.

4.3.4 The conproxy server

The default behavior for each component (blackhole, horizon, wormhole) is to fork a new process to handle a connection. Each process copies back and forth between two sockets. While this is robust and performs well, it is not a good solution to handle thousands of connections. The **conproxy** is an optimised single process server handling connection in non-blocking mode. Each component knows how to use its service. It is a good idea to enable the conproxy service everywhere you are running one blackhole component. The service has no configuration. It listens on a Unix domain socket (/var/run/blackhole/conproxy.sock). It does not connect anywhere. It just receives established connection handles (pairs) through the Unix socket. It then copies back and forth between each pairs. It uses non-blocking I/O.

4.3.5 The rendez-vous principle

Network wise, the **blackhole** project only requires limited connectivity (very limited), yet delivers controlled connectivity from anywhere to anywhere. The only necessary rules are:

- The **blackhole** server(s) must be able to connect to all **horizon** servers.
- The vservers must be able to connect to the application ports of the **horizon** server on the same host.
- The **horizon** server must be able to connect to the various vservers on the same host.

That's all. Vservers do not have to go anywhere else. So the host running vservers is pretty much locked down. It can't go anywhere on its own. It can't connect to the **blackhole** server either. So how does it work ? Here are the simple steps.

- One vserver connects to the **horizon** server.
- The **horizon** server sends a connection request to the **blackhole** server using a connection already established by the blackhole. It tells the blackhole the name of the vserver, the TCP port used and a token identifying the pending connection.
- If the **blackhole** authorizes the connection, it connects back to the **horizon** and requests to link this new connection to the pending connection using the token.
- From now on, the **horizon** server just copies back and forth between the pending connection and the second **blackhole** connection.

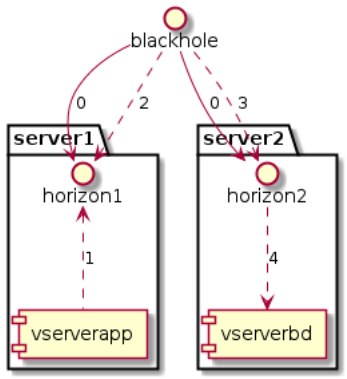
So the client application end up connected to a server somewhere without ever having done a TCP connection outside of its own box. This is one side of the process. The other side is:

- Once the **blackhole** server has authorized the connection, it connects back to the horizon server and request a link between this new connection and the pending connection.
- It also connects to another **horizon** server (controlled by the access rule) and requests it to connect to a local service.
- From now one, the **blackhole** server just copies back and forth between the two **horizon** servers.

Here is a more detailed explanation:

- The **horizon** server receives a connection from one vserver on the same host.
- It identifies the name of the vserver from its source IP number.
- It also identifies the target IP number. The **horizon** server may listen on several internal IP numbers.
- It also identifies the destination port number.
- Using all this information, it creates a request "vserver ipN dest-port". It sends a connect request using one TCP connection already established by a **blackhole** server (remember, the **horizon** can't connect to the **blackhole**. The connect request contains a token identifying the pending connection.
- The **blackhole** server prepends the name of the horizon (it knows its name because it is the blackhole which connects to the horizon). So the four inputs become the key to locate a connection rule. The left hand side is the key, the right hand side tells the host, the vserver and the port to use to complete the connection.
- If there is no rules, it replies to the **horizon** to reject the connection.
- If there is a rule, it contacts the **horizon** server on the right hand side and requests it to connect to one of its vserver on the destination TCP port.

The following diagram shows the connection order when a vserver named **vserverapp** tries to connect to a vserver named **vserverbd**.



0. Connections 0 are pre-established from the blackhole server to the horizon servers.
1. The vserverapp connects to port 4101 of the horizon server on IP 192.168.2.1.
2. The blackhole connects back to the horizon server using the **link** command. It requests to link the connection 1 with connection 2. From now on, the horizon server (a sub-process actually) copies back and forth between the two connections (raw copy).

3. The blackhole connects to the second horizon server and request it connects to vserver vserverbd on port 4101.

4. The horizon server connects to port 4101 of the vserver vserverbd. From now on the horizon server will copy everything back and forth between connection 3 and 4.

If the blackhole finds out that the target horizon server is the same as the source, it connects back to the source horizon and request it to do everything:'It connects to a local vserver and link to the pending connection. In that case, the blackhole is out of the loop.

5 Best practices

The blackhole provides a way to establish connections between any hosts on a complex network. Yet it does not require direct connectivity between hosts. The interest of the blackhole project security wise is that very little can operate on the network without its control. To enforce this, here are simple rules to make your data center much more resilient to external treats:

- All horizon servers should listen on the same IP for application ports. This IP should be installed as an IP alias on the loopback. It becomes (by default) unreachable from the outside. Only local vservers may reach it. We suggest **192.168.2.1**. If more than one IP is needed, pick the following (192.168.2.2, 192.192.168.2.3, ...).
- All vservers should use the same IP range. The first vserver in a host could use **192.168.3.1** and the second **192.168.3.2** and so on. Install those addresses on the loopback as well. You will end up with many vservers on your network using the same IP number. Note that all access rules in the blackhole server are expressed using vserver name. The horizon server does the translation to IP number when connecting to an application port.
- Install the following iptables rules. Note that this is the exact same rules on all/most hosts. Host operating in **open_network** mode will need different rules.
  - Deny access to network 192.168.2.0 from outside
  - Deny access to network 192.168.3.0 from outside
  - Deny connections to the outside from 192.168.3.0 (few exceptions maybe, like DNS). If you need direct access (without blackhole control) to the outside from the vserver, you will need to use **DNAT** since all vservers are using the same addresses.
  - Deny connections from 192.168.3.0/24 to 192.168.3.0/24. Vservers won't be able to talk to each other. A vserver won't be allowed to talk to itself, unless it uses the loopback (localhost, 127.0.0.1)

6 About vservers

This project is not tied to **vservers**, but uses one extremely powerful idea: A vserver can't lie when doing networking. Most people who look for the first time at vservers think they are simply less capable virtual machines. Less capable in the sens that there are few things they can't do. Actually, there are few things they **won't** do. By default, they can't change their network configuration. They can't change routing nor their assigned IP numbers. Further, they can't lie. All outgoing connections have to be bound to their assigned IP numbers. And they can't do (by default) raw packet.

So vservers are a very very safe way to run application services. If a vserver has a security flaw, the attacker end up in a very unfriendly place to fool around. Actually, ping does not even work (again by default) inside a vserver.

So the **blackhole** project takes advantage of this: A vserver may be identified from its source IP address.

7 Implementation

The source code is available using **subversion** at

```
http://svn.solucorp.qc.ca/repos/solucorp/blackhole/trunk
```

You can build it simply by issuing

```
make
make install
```

or on fedora

```
make buildrpm
```

To compile it, you need the linuxconf-devel and linuxconf-lib package. You can grab the latest source here

```
http://svn.solucorp.qc.ca/repos/solucorp/linuxconf/trunk
```

then you can build an rpm for it

```
make buildrpm
```

Linuxconf is not maintained anymore, but the library is. At some point it will be renamed...