ERIC SORENSON

enterprise routing sinkholes using Linux and open source tools

Eric Sorenson does network administration at Transmeta in Santa Clara, California. He likes commuting by bicycle and dislikes Sarbanes-Oxley compliance audits.

eric@transmeta.com

OCCASIONALLY YOU'LL SEE THEM ON the evening news or a photoblog—a previously forgotten septic tank or abandoned sewer line has opened up and swallowed somebody's Volkswagen, and the puzzled owner is standing above his submerged car scratching his head. They're called sinkholes, and the photos are amusing precisely because it's happening to somebody else. So why would you, a sophisticated system and network administrator, want to inflict one on your enterprise?

The routing sinkhole I'll be discussing in this article has a few characteristics in common with those staples of the eleven o'clock news: They are both uncommon and unexpected, and their purpose in life is to swallow up anything that comes their way. Unlike a disused septic tank, however, a routing sinkhole has a specific useful purpose: to provide administrators with detailed statistical data about traffic that ends up falling into it. More specifically, a sinkhole is a trap for traffic that we know to be illegitimate because it's addressed to parts of RFC1918 private address space not in use by our enterprise or any other privately connected network we know about. The very existence of this traffic is evidence that *something* untoward is happening on the network. The sinkhole's job is to help us find out what that something is.

Before going much further, I should mention that this is not an invention of my own devising. I first read about routing sinkholes in Richard Bejtlich's excellent book *Extrusion Detection: Security Monitoring for Internal Intrusions* [1], and this implementation is based on his description in Chapter 5 of that book, "Layer 3 Network Access Control." However, it differs in two significant ways, which I believe make it worth describing separately. First, I use Linux on commodity hardware for the sinkhole platform, rather than the Cisco Bejtlich uses; second, I detail the setup of a NetFlow-based collection point for historical session and statistical data, whereas the book stops at enabling NetFlow on the interface and just uses IOS commands to show real-time NetFlow summaries. Sinkholes are also conceptually similar to the work CAIDA has done with Network Telescopes [2], which watch unused routable addresses for DoS traffic and backscatter.

Architectural Overview

So what, exactly, makes a sinkhole? There are really two parts to it, which I'll walk through setting up in the rest of the article. First, specialized routing rules need to be set up and propagated throughout your enterprise network to direct the known-bad traffic toward the sinkhole. Second, the sinkhole needs to generate statistics so that administrators can draw conclusions about the nature of the traffic the sinkhole sank.

The routing component has a couple of presuppositions about the network environment that ought to be explicitly stated. I'm going to assume your network:

- Uses some chunks of private address space, either in 10.0.0.0/8, 172.16.0.0/12, or 192.168.0.0/16.
- Uses some type of routing internally to direct traffic to various subnets or (minimally) toward a default gateway that has an Internet connection.
- Does not rely on your default gateway to get to private networks that are in use.

The neat hack that makes the sinkhole work in an environment that meets these criteria is this: For an IP router, the "best" path to a particular destination is the one with the longest prefix. In other words, routers prefer the most specific route. The sinkhole takes advantage of this predilection by suggesting itself as a path to various destinations but with extremely unspecific prefixes, so the route to any legitimate internal network will be preferred by all the routers. Only fall-through traffic, whose destinations are not overridden by longer prefixes, hits the sinkhole. (Though I should note that all the sinkhole destinations are more specific than your default gateway, which suggests itself as the route to 0.0.0.0/0.0.0.0.) The sinkhole then accepts the traffic that uses it, records its existence, and promptly throws it away.

As for the statistics, there are several components that need to work together. The goal is to use the large body of open source tools that use Cisco's NetFlow packet format, which is widely used to summarize conversations of traffic that pass through a router. We'll put together a flow probe, which makes your Linux box pretend to be an expensive Cisco router by recording Layer 4 session data (source and destination IP, source and destination port, protocol). The probe then forwards that data in the form of NetFlow packets to the flow collector, which receives the traffic and records it for posterity. The collector also answers queries from a flow frontend, which administrators use to check up on sinkhole activity from the safety of their browsers or terminals.

Routing Implementation

Regardless of which internal routing protocol you use, the first step in asserting your sinkhole's presence on the network is to start advertising some routes to it. There are a couple of different ways to go about this: Either set up static routes on the sinkhole's upstream router and redistribute them into your dynamic routing protocol, or have the sinkhole participate directly in dynamic routing with something like Quagga [3]. I've just redistributed the static routes from the core router immediately upstream from the sinkhole, which is easier to set up but requires that the sinkhole also have static routes back to your enterprise's subnets, or it'll sink legitimate traffic too!

In this example, the sinkhole uses the two-IP subnet 10.10.15.16/30 for its link to the upstream router, the routers use OSPF among themselves, and there are two internal subnets we care about reaching from the sinkhole. The servers reside on 10.0.10.0/24 and the desktops live on 10.0.8.0/23, as shown:

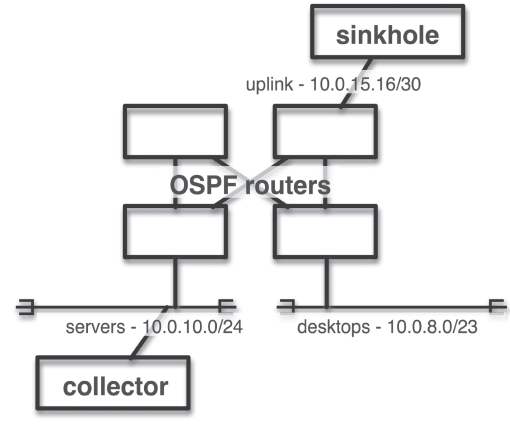

FIGURE 1: RELATIONSHIP OF THE INTERIOR ROUTERS AND DESKTOP NETWORK TO THE COLLECTOR AND SINKHOLE.

Assuming the router's end of the uplink is 10.0.15.18/30, we'd set up the RFC1918 routes to point to the sinkhole and then enable redistribution from statically configured routes into OSPF so that the rest of the OSPF-speaking routers will learn them:

```
ip route 10.0.0.0 255.0.0.0 10.0.15.17 10
ip route 172.16.0.0 255.240.0.0 10.0.15.17 10
ip route 192.168.0.0 255.255.0.0 10.0.15.17 10
router ospf
redistribution static
```

Here's a shell script snippet to configure the routes on the sinkhole box itself:

```
# known-good nets that should remain reachable
ip ro add 10.0.8.0/23 via 10.0.15.18
ip ro add 10.0.10.0/24 via 10.0.15.18
# repeat for any other networks you don't want to toss
# ...
# then add the 'blackhole' routes
ip route add blackhole 192.168.0.0/16
ip route add blackhole 172.16.0.0/12
ip route add blackhole 10.0.0.0/8
```

At this point you should be able to traceroute to nonexistent private addresses and see your packets disappear into the sinkhole. Running tcpdump on your sinkhole's Ethernet interface should provide some interesting—but probably somewhat frightening!—output. Make sure you save your changes, with `write mem` on the router and `ip route > /etc/sysconfig/network-scripts/route-eth0` (RHEL) or their equivalents. Now it's time to set up statistics collection.

Collection, Summarization, and Reporting

There are myriad different NetFlow-related resources on the Net, including several different collector implementations, plug-ins for popular packages

such as NTop, and lots of Web and CLI tools for extracting the data. I'm going to focus on just a couple of packages: fprobe [4] and nfdump [5].

Setting up fprobe should be a straightforward configure; make; make install; the only difference between our sinkhole setup and a normal fprobe-running Linux router is that we want to avoid reporting on traffic that's directly to or from the sinkhole's IP. Fortunately, fprobe uses Berkeley Packet Filter (BPF) syntax such as tcpdump, so this is easy to accomplish. I've picked an unused high port on the NetFlow collection host (named "collector") and direct fprobe's output there with the last argument:

```
/usr/sbin/fprobe eth0 -a 10.0.15.17 -l 1 -f "not host 10.0.15.17"
collector:23001
```

The recipient of the NetFlow packets sent out by fprobe is nfcapd, part of the nfdump distribution. Again, this is well-behaved open-source software, so a quick download from SourceForge and configure; make; make install cycle later and we are ready to log some flow statistics. The most important parts of starting up a NetFlow collector are, first, to make sure you have plenty of free space on the disk partition you're writing to, and, second, to match the port you're listening on with the one to which your probe will be sending. In this case, we end up with a command line like the following:

```
/usr/bin/nfcapd -w -D -l /nsm/netflow/sinkhole -p 23001
```

For the impatient or untrusting, it's easy to verify that things are working as expected. Fire up tcpdump and watch for incoming traffic to port 23001. You should see inbound traffic from your probe to the collector, and after a few minutes the byte count of the log files in the directory specified in the -l option to nfcapd should increase.

Reporting, the final piece of our sinkhole implementation, comes courtesy of another tool from the nfdump distribution. This time it's nfdump itself, which is to NetFlow capture files what tcpdump is to pcap files. nfdump was installed along with nfcapd, so it should be ready to run. After a few hours of capture, the sinkhole has probably picked up some interesting traffic. Nfdump's flow aggregation abilities let us get a quick summary of the flows we've captured; the foremost useful aggregation for our sinkhole will be on source IP and destination port. This will help find hosts that are scanning for the same service on different hosts. A side effect of the aggregation is that the Dst IP Addr in the following output is represented as zeros—we'll drill down without aggregation in the second example to see the details of each individual flow matching the pcap-style expression host 10.0.10.15. The output can be further customized to show just the relevant fields using the -o fmt: . . . option string.

```
# nfdump -o "fmt:%pr %sap - %dap %pkt %byt %fl" -a -A srcip,dstport -R .
```

Proto	Src IP Addr:Port		Dst IP Addr:Port	Packets	Bytes	Flows
UDP	10.0.8.198:0	-	0.0.0.0:161	164	17384	42
UDP	10.0.10.15:0	-	0.0.0.0:137	520	46800	104
TCP	10.0.10.12:0	-	0.0.0.0:22	32572	1.9 M	15930

```
# nfdump -o "fmt:%pr %sap - %dap %pkt %byt"  -R . "host 10.0.10.15"
```

Proto	Src IP Addr:Port		Dst IP Addr:Port	Packets	Bytes
UDP	10.0.10.15:137	-	172.16.130.1:137	5	450
UDP	10.0.10.15:137	-	172.16.41.1:137	5	450

The capture, collection, and reporting infrastructure is up and running now, but an additional element is necessary to make the whole thing go: the application of human intelligence to analyze the data and investigate the results.

Analysis of the Results

In his book *The Tao of Network Security Monitoring* [6], Richard Bejtlich describes three categories of network traffic events:

> *Normal* traffic is anything that is expected to belong on an organization's network . . . *suspicious* traffic appears odd at first glance but causes no damage to corporate assets. . . . *Malicious* traffic is anything that could negative[ly] impact an organization's security posture. *(p. 361)*

By definition, none of the traffic that ends up at the sinkhole is normal, so our analysis will be aimed at separating the malicious traffic from the merely suspicious.

Remember from the network diagram that 10.0.10.0/24 is the server subnet and 10.0.8.0/23 are desktop subnets. In the nfdump output snippet from the previous section, there's a desktop that's doing repeated SNMP requests (UDP to port 161), a server with frequent NetBIOS name requests (UDP to port 137), and another server that is doing a huge amount of ssh scanning (TCP to port 22). The general procedure for investigation is to find out as much as possible about each of these flows from the nfdump data, and if there are still unresolved questions about the nature of the traffic, try to capture full packets from the sinkhole box itself. It's here where the advantage of a general-purpose Linux box as a sinkhole comes into play. Assuming we're proactive enough to notice suspicious traffic flows while they are still ongoing, it's easy to set up tcpdump captures on the sinkhole itself to get whole packets. This can be invaluable in determining what variant of a worm is sending out a particular scan, for instance.

In our example matrix, it turned out the ssh scans were due to an administrator's host-key gathering script that used erroneous DNS data to determine which hosts to scan. Fixing the script to limit its search by subnet rather than hostnames sped up its execution by a factor of 5. The NetBIOS requests were due to bad name registrations on the master browser, from hosts that were connected with a VPN client but registered the address on their remote LAN. Full-packet capture showed that the SNMP requests were to a branch of the host MIB used by HP printers to report status. The requests were generated automatically by a printer driver to which a user had pointed the user's home printer.

Conclusions and Future Work

Analyzing traffic that ends up in a routing sinkhole can be an enlightening experience. It turns out that there is a fair amount of background noise to various nonroutable address blocks: In addition to the three analyses shown here, I found that there was substantial bleed-through from Windows laptops that moved between our wired and wireless networks, which are on different RFC1918 blocks that are not reachable from each other. Not every packet that ends up at the sinkhole is evidence of malicious activity (unless you count NetBIOS Name Service as malicious!).

I regret that I did not have a sinkhole set up for the outbreaks of the Sasser and Welchia worms. It would have greatly aided in combating those

worms, for two reasons. First, having historical data about which IP addresses were infected first and how the worm spread would have helped the labor-intensive mop-up efforts. Second, without a sinkhole to attract the bogus addresses generated by the worm scanners, hundreds of bogus packets per second were routed toward our firewall, which collapsed under the DoS load. With the sinkhole in place I'm almost looking forward to the next outbreak. Err . . . scratch that, actually.

Further refinements include setting up the very impressive NfSen Web-based frontend [7] to the nfdump collector; fine-tuning the list of networks that are blackholed, perhaps to include all IANA-reserved blocks from RFC3330; and setting up NetFlow exports on all the router interfaces that support it. Alerting and automated response are also on the horizon, but obviously a good amount of data ought to be collected to set appropriate thresholds before we start paging people out of bed or automatically shutting down switch ports.

It may not be able to swallow a Volkswagen, but the routing sinkhole is a very useful weapon in the fight against entropy and chaos on the enterprise network.

REFERENCES

[1] Richard Bejtlich, *Extrusion Detection: Security Monitoring for Internal Intrusions* (Addison Wesley Professional, Reading, MA, 2006).

[2] CAIDA Network Telescopes: http://www.caida.org/analysis/security/telescope/.

[3] Quagga: http://www.quagga.net/.

[4] Fprobe: http://fprobe.sf.net/.

[5] Nfdump: http://nfdump.sf.net/.

[6] Richard Bejtlich, *The Tao of Network Security Monitoring: Beyond Intrusion Detection* (Addison Wesley Professional, Reading, MA, 2005).

[7] NfSen: http://nfsen.sf.net/.