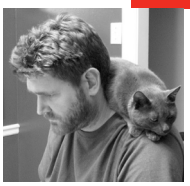
DAVE JOSEPHSEN

iVoyeur: packet-level, per-user network access control and monitoring



Dave Josephsen is the author of *Building a Monitoring Infrastructure with Nagios* (Prentice Hall PTR, 2007) and is senior systems engineer at DBG, Inc., where he maintains a gaggle of geographically dispersed server farms. He won LISA '04's Best Paper award for his co-authored work on spam mitigation, and he donates his spare time to the SourceMage GNU Linux Project.

dave-usenix@skeptech.org

WE WATCH THE LOGS SCROLL BY, USERS in the offices surrounding us using VPN to access specific ports on the specific servers they need in the server room based on business roles that apply to them. My mind shifts down one level. The gateway device allows them access to the Internet, where they loop back to the public VPN endpoint for this building. PKI and LDAP Bind authenticate them to the VPN endpoint, and it creates packet filter rules for them based on the outcome of LDAP queries, loads their rules to the VPN users' PF anchor, and hands them routes to privileged internal subnets. This is the kind of interaction that makes my frontal lobes buzz. Lots of pieces, lots of layers. Physical, logical, human, abstract; lots of things to understand—no, lots of *ways* to understand, and we do, because we built it.

It all fits so well together that it's hard to see the pieces now—like these unrelated projects were intended to be subsystems of the whole we've created. The best systems I've built work this way; the ones that are going to stick to the wall. I imagine that real scientific discovery feels something like this. When it's done and you step back and look at it, you know that you've found *the* answer to this particular question, and that when you move on it will remain. This answer is right, it is *truth*, and anything else is a compromise...a kludge.

We sit in silence, lost in thought, mulling details; routing, schema extensions, NAT, tunnels. What if . . . no, that's accounted for. But then what if . . . no, the design takes care of that too.

Finally my cohort blinks and shakes his head. "Whose idea was this?"

All I can do is chuckle and shrug. I honestly can't remember. The design has been haunting me for what feels like years, but I can't say for sure it originated in my brain. It was a progression. An artifact of our collective familiarity with these tools, our familiarity with each other, and our daily carpool brainstorms. Having a need for a network access control system probably didn't hurt, but in reality it wasn't much of a catalyst either. Solutions like this build themselves when they are ready to be built, and which of us is to blame isn't important, even

if it were answerable. This thing scrolling away before us is a product of our "us-ness," and also, it's *awesome*.

"It's a pretty good idea," he says.

"Yeah, not bad," I reply.

I should pause here for a short disclaimer: The thing I hate and dread about writing implementation articles is sharing my code. Easy there, Captain Open Source, I'm not being proprietary corporate guy. The thing I hate about sharing my code is inviting you, dear reader, into my brain. It's far easier to stay on the English composition side of this equation, where a well-placed semicolon or two might disguise the blathering idiot I truly am (unlikely, but possible). Sharing source code with the readership of *;login:*, on the other hand, is something like showing up naked to a photography convention. It's not something to be done lightly if you value the respect of your peers, and, being painfully aware of that, I wanted to make sure you knew the code herein is mine. My cohort is innocent in that regard, his kimono firmly closed as it were.

So, when did this idea coalesce? I don't know that either, but the implementation started with a fortuitous network redesign. We moved our headquarters in April, which provided us with the opportunity to redesign the network from the ground up. New numbers, new segments, new providers—the whole deal. A big part of the redesign was this VPN-based access control scheme we'd already been kicking around. The idea was that instead of having an "internal" subnet for our employees, we'd have an untrusted segment officially referred to as "public." University administrators are probably pretty familiar with this idea: I've sometimes heard them call it "the heathen zone."

It's assumed that bad things happen in the heathen zone as a matter of course, and that the systems within it should be allowed relatively unfettered (but NATed) access to the Internet and not much else. In our new corporate network, if a heathen wants to use services that don't have public external addresses (POP, printers, etc.), they should do what they'd do at Starbucks—no, not pretend to read while hoping someone will talk to them, but VPN in. When they do that, we can give them access to the exact services on the specific systems they need, and we can tie their traffic to a UID and monitor/log it. The new network was designed with this in mind.

The next step was figuring out an access control scheme (language?). LDAP was the obvious choice as a database, but how to implement it? We knew we wanted a very flexible role-based scheme that would scale and make it easy to optimize for performance during searches by limiting the search base. We also wanted to be able to consolidate a few other LDAP systems into it for things like FTP and mail authentication, and even asset control and machine inventory. I've done a lot of things in my professional career, but getting LDAP right the first time isn't one of them. In fact, I have rarely gotten a design that I like for more than a few weeks. I've also never found a design that I could move from one company to another. In this case, I think on the third or fourth try we got something that stuck.

We modified the schema to add a few objects of our own, for things like networks, servers, and a role object, with a socket-style "grant" attribute that specifies host/service tuples. The easiest way to give you a feel for how it works is to show you what the VPN endpoint does when an employee logs into it.

Step 1. Given a unique UID, look up the user's DN:

```
ldapsearch (uid=dave) dn
```

This yields something like:

```
uid=dave,ou=foo,ou=bar,dc=dbg,dc=com
```

Step 2. Given a user's DN, look up what it's been granted access to:

```
ldapsearch (&(member=uid=dave,ou=foo,ou=bar,dc=dbg,dc=com)
(objectclass=dbgRole)) dbgGrants
```

This returns a list of server/service tuples that look like this:

```
fooserver.dbg.com:login
```

Step 3. For each tuple, resolve the IP address and port number:

```
ldapsearch (&(cn=a.ig05.dc4.dbg.com)(objectclass=dbgNetwork)) dbgAddress
```

One or more IP addresses in CIDR notation may be returned.

At the moment, there is no service object in LDAP that maps the service to a port number. This is because the service definition is arguably relative, given that future consumer programs might use a different port for the same service name or might not want them mapped to TCP port numbers at all, and anyway creating 30,000+ LDAP objects that mostly won't ever be used just seems wrong. At the moment, I think it's better that the consumer application interpret the service name ("login" in this example) for itself. On the VPN gateway we do this with a slightly modified copy of the /etc/services file.

The VPN endpoint runs OpenBSD, with OpenVPN and the PF (Packet Filter) firewall. There are three OpenVPN configuration parameters that make this design possible. The first is actually optional: `--auth-user-pass-verify` allows us to authenticate the user via LDAP, which saves us from having to issue new certs every time a heathen forgets its password.

The next two go hand-in-hand: `--client-connect` and `--client-disconnect`. These allow us to call a script of our choosing when a client connects and/or disconnects, and are pretty much the bailing wire holding this all together. I wrote a shell script I call VPLDPF (VPN-LDAP-PF) that gets the user's UID from OpenVPN as `$1`, along with a bunch of other interesting variables. VPLDPF's job is to perform the necessary LDAP queries to figure out what hosts/ports the heathen gets access to, translate these into PF rules, and finally load them into PF. The script is available linked under this article at http://www.usenix.org/login/2009-10/.

PF's "anchor" feature makes this sort of automated dynamic firewall configuration safe and easy. Anchors are named sets of filter rules that can be maintained and loaded separately from the main PF rule set. VPLDPF uses LDAP searches to create PF filter rules for every user who logs in and then stores them in a file named after the user in /etc/pfanchors/vpnusers. Once we've told PF that we'll be using an anchor called `vpnusers` by adding `anchor 'vpnusers/*'` to /etc/pf.conf, VPLDPF can load, for example, Bob's rule set:

```
pfctl -a vpnusers/bob -f /etc/pfanchors/vpnusers/bob
```

Going into it, I thought the initial population of LDAP was going to be time-consuming, but the "roles" scheme we came up with didn't take much effort,and has made it pretty easy to get very granular permissions on an individual employee basis. How granular? Let's take a look at "Bob," a pretend employee modeled after a real project manager.

Bob's DN is:

```
uid=bob,ou=projectManagement,ou=staff,dc=dbg,dc=com
```

Running an ldapsearch for object class dbgRole with Bob's DN in the member attribute, we find that Bob has three roles assigned to him:

```
dn: cn=employee,ou=roles,dc=dbg,dc=com
dn: cn=HQVPNUser,ou=roles,dc=dbg,dc=com
dn: cn=PM,ou=roles,dc=dbg,dc=com
```

The employee role contains the following dbgGrants attributes:

```
dbgGrants: fileServ1.dbg.com:login          # a fileserver
dbgGrants: fileServ1.dbg.com:http
dbgGrants: fileServ1.dbg.com:https
dbgGrants: mail.dbg.com:http                #an email server
dbgGrants: mail.dbg.com:https
dbgGrants: mail.dbg.com:pop3
dbgGrants: mail.dbg.com:smtp
dbgGrants: mail.dbg.com:xmpp-client         #this is the jabber port
```

The HQVPNUser role contains the following dbgGrants:

```
dbgGrants: ns.hq.dbg.com:domain
```

And the PM role contains the following dbgGrants:

```
dbgGrants: pm.dbg.com:http        # the project management server
```

Using the server name as a CN to resolve the IP address and looking in the services file for the port, VPLDPF created the following PF rules for Bob:

```
pass in inet proto tcp from 10.253.21.10 to 10.21.1.2 port = ssh flags S/SA
    keep state
pass in inet proto tcp from 10.253.21.10 to 10.21.1.2 port = https flags S/SA
    keep state
pass in inet proto tcp from 10.253.21.10 to 10.21.1.2 port = www flags S/SA
    keep state
pass in inet proto tcp from 10.253.21.10 to 10.21.64.101 port = https flags S/
    SA keep state
pass in inet proto tcp from 10.253.21.10 to 10.21.64.101 port = www flags S/
    SA keep state
pass in inet proto udp from 10.253.21.10 to <__automatic_adb98624_0> port
    = domain flags S/SA keep state
pass in inet proto tcp from 10.253.21.10 to 10.21.1.4 port = www flags S/SA
    keep state
```

The first thing to note is that the "login" service was translated to port 22. A different LDAP client program might have used RDP, or "login" might have been used as a requirement in something like nsswitch.conf if we were doing LDAP Auth on a Linux box, for example. This is why we choose to interpret the service name in the app instead of in LDAP.

Next, note the weird-looking PF destination address for the DNS rule: <__automatic_adb98624_0>. This is a dynamic table that was generated for us by PF. The server object whose CN is ns.hq.dbg.com has multiple address attributes associated with it. This caused VPLDPF to generate a slew of PF rules, one per destination address for that server object. When those rules were loaded into PF, PF saw that everything but the destination address was redundant, so it optimized these rules down to a single rule by creating a table for all of ns.hq.dbg.com's destination addresses. If we wanted to see the contents of this table, we could ask PF with the command:

```
pfctl -a vpnusers/bob -t __automatic_adb98624_0 -T show
```

To make it easy to track the current state of things, I wrote another shell script that parses OpenVPN's status log for the currently connected users and runs the pfctl commands necessary to dump the PF details on each of them. This script, called vpninfo.sh and also available linked under this article at http://www.usenix.org/login/2009-10/, gives the following output for Bob:

```
################## bob ######################
localIP: 10.253.21.10, remoteIP: 67.16.87.60:13771
Connected since: Sat Jul 25 17:24:19 2009

PF Rules for user bob
pass in inet proto tcp from 10.253.21.10 to 10.21.1.2 port = ssh flags S/SA
    keep state
pass in inet proto tcp from 10.253.21.10 to 10.21.1.2 port = https flags S/SA
    keep state
pass in inet proto tcp from 10.253.21.10 to 10.21.1.2 port = www flags S/SA
    keep state
pass in inet proto tcp from 10.253.21.10 to 10.21.64.101 port = https flags S/
    SA keep state
pass in inet proto tcp from 10.253.21.10 to 10.21.64.101 port = www flags S/
    SA keep state
pass in inet proto udp from 10.253.21.10 to <__automatic_adb98624_0> port
    = domain flags S/SA keep state
pass in inet proto tcp from 10.253.21.10 to 10.21.1.4 port = www flags S/SA
    keep state
```

PF Dynamic Table Contents for user Bob:

```
#### __automatic_adb98624_0 ###
  10.21.0.1
  10.21.16.1
  10.21.32.1
  10.21.48.1
  96.26.18.66
```

Since users each get their own set of firewall rules, we can associate every packet they send with their username by, for example, tagging their packets with their name or logging them to a special pflog interface. We can monitor and otherwise collect usage information on particular heathens with access to sensitive systems (Holt-Winters forecasting anyone?), and we get the happy side effect of encrypting all traffic in the heathen zone that's destined for privileged networks, whether the protocols in use are encrypted or not. These are the sorts of things that make auditors go all giggly. The impact on our users, most of whom were already used to using VPN from home, was pretty minimal, and I'm far happier with this than any of the real NAC solutions we've tried, though that's arguably apples and oranges.

As far as caveats go, there are two that spring to mind: First, I wish PF had an iptables-style log-prefix feature. That would make auditing far easier (time to delve into PF's source code perhaps). Second, this solution makes it somewhat tricky for systems in the privileged networks to reliably initiate connections to heathen workstations, which may or may not be a problem in your environment. We have a few people who forward their mail from the mail system to the smtp daemon on their workstation by way of a .qmail file, and we're having to find some workarounds for that. Otherwise, it's been all smiles and giggly auditors for us. Whoever's idea this was, it was not bad at all.

Take it easy.