NAME

authpf, **authpf-noip** - authenticating gateway user shell

SYNOPSIS

authpf
authpf-noip

DESCRIPTION

authpf is a user shell for authenticating gateways. It is used to change pf(4) rules when a user authenticates and starts a session with sshd(8) and to undo these changes when the user's session exits. Typical use would be for a gateway that authenticates users before allowing them Internet use, or a gateway that allows different users into different places. Combined with properly set up filter rules and secure switches, **authpf** can be used to ensure users are held accountable for their network traffic. It is meant to be used with users who can connect via ssh(1) only, and requires the pf(4) subsystem and an fdescfs(5) file system mounted at */dev/fd* to be enabled.

authpf-noip is a user shell which allows multiple connections to take place from the same IP address. It is useful primarily in cases where connections are tunneled via the gateway system, and can be directly associated with the user name. It cannot ensure accountability when classifying connections by IP address; in this case the client's IP address is not provided to the packet filter via the *client_ip* macro or the *authpf_users* table. Additionally, states associated with the client IP address are not purged when the session is ended.

To use either **authpf** or **authpf-noip**, the user's shell needs to be set to */usr/sbin/authpf* or */usr/sbin/authpf-noip*.

authpf uses the pf.conf(5) syntax to change filter and translation rules for an individual user or client IP address as long as a user maintains an active ssh(1) session, and logs the successful start and end of a session to syslogd(8). **authpf** retrieves the client's connecting IP address via the SSH_CLIENT environment variable and, after performing additional access checks, reads a template file to determine what filter and translation rules (if any) to add, and maintains the list of IP addresses of connected users in the *authpf_users* table. On session exit the same rules and table entries that were added at startup are removed, and all states associated with the client's IP address are purged.

Each **authpf** process stores its rules in a separate ruleset inside a pf(4) *anchor* shared by all **authpf** processes. By default, the *anchor* name "authpf" is used, and the ruleset names equal the username and PID of the **authpf** processes as "username(pid)". The following rules need to be added to the main ruleset */etc/pf.conf* in order to cause evaluation of any **authpf** rules:

```
nat-anchor "authpf/*"
```

```
      rdr-anchor "authpf/*"
      binat-anchor "authpf/*"
      anchor "authpf/*"
```

The "/*" at the end of the anchor name is required for pf(4) to process the rulesets attached to the anchor by **authpf**.

FILTER AND TRANSLATION RULES

Filter and translation rules for **authpf** use the same format described in pf.conf(5). The only difference is that these rules may (and probably should) use the macro *user_ip*, which is assigned the connecting IP address whenever **authpf** is run. Additionally, the macro *user_id* is assigned the user name.

Filter and translation rules are stored in a file called *authpf.rules*. This file will first be searched for in */etc/authpf/users/$USER/* and then in */etc/authpf/*. Only one of these files will be used if both are present.

Per-user rules from the */etc/authpf/users/$USER/* directory are intended to be used when non-default rules are needed on an individual user basis. It is important to ensure that a user can not write or change these configuration files.

The *authpf.rules* file must exist in one of the above locations for **authpf** to run.

CONFIGURATION

Options are controlled by the */etc/authpf/authpf.conf* file. If the file is empty, defaults are used for all configuration options. The file consists of pairs of the form name=value, one per line. Currently, the allowed values are as follows:

anchor=name
: Use the specified *anchor* name instead of "authpf".

table=name
: Use the specified *table* name instead of "authpf_users".

USER MESSAGES

On successful invocation, **authpf** displays a message telling the user he or she has been authenticated. It will additionally display the contents of the file */etc/authpf/authpf.message* if the file exists and is readable.

There exist two methods for providing additional granularity to the control offered by **authpf -** it is possible to set the gateway to explicitly allow users who have authenticated to ssh(1) and deny access to only a few troublesome individuals. This is done by creating a file with the banned user’s login name as the filename in */etc/authpf/banned/*. The contents of this file will be displayed to a banned user, thus providing a method for informing the user that they have been banned, and where they can go and how to get there if they want to have their service restored. This is the default behaviour.

It is also possible to configure **authpf** to only allow specific users access. This is done by listing their login names, one per line, in */etc/authpf/authpf.allow*. A group of users can also be indicated by prepending "%" to the group name, and all members of a login class can be indicated by prepending "@" to the login class name. If "*" is found on a line, then all usernames match. If **authpf** is unable to verify the user’s permission to use the gateway, it will print a brief message and die. It should be noted that a ban takes precedence over an allow.

On failure, messages will be logged to syslogd(8) for the system administrator. The user does not see these, but will be told the system is unavailable due to technical difficulties. The contents of the file */etc/authpf/authpf.problem* will also be displayed if the file exists and is readable.

CONFIGURATION ISSUES

authpf maintains the changed filter rules as long as the user maintains an active session. It is important to remember however, that the existence of this session means the user is authenticated. Because of this, it is important to configure sshd(8) to ensure the security of the session, and to ensure that the network through which users connect is secure. sshd(8) should be configured to use the *ClientAliveInterval* and *ClientAliveCountMax* parameters to ensure that a ssh session is terminated quickly if it becomes unresponsive, or if arp or address spoofing is used to hijack the session. Note that TCP keepalives are not sufficient for this, since they are not secure. Also note that the various SSH tunnelling mechanisms, such as *AllowTcpForwarding* and *PermitTunnel*, should be disabled for **authpf** users to prevent them from circumventing restrictions imposed by the packet filter ruleset.

authpf will remove state table entries that were created during a user’s session. This ensures that there will be no unauthenticated traffic allowed to pass after the controlling ssh(1) session has been closed.

authpf is designed for gateway machines which typically do not have regular (non-administrative) users using the machine. An administrator must remember that **authpf** can be used to modify the filter rules through the environment in which it is run, and as such could be used to modify the filter rules (based on the contents of the configuration files) by regular users. In the case where a machine has regular users using it, as well as users with **authpf** as their shell, the regular users should be prevented from running **authpf** by using the */etc/authpf/authpf.allow* or */etc/authpf/banned/* facilities.

authpf modifies the packet filter and address translation rules, and because of this it needs to be configured carefully. **authpf** will not run and will exit silently if the */etc/authpf/authpf.conf* file does not exist. After considering the effect **authpf** may have on the main packet filter rules, the system administrator may enable **authpf** by creating an appropriate */etc/authpf/authpf.conf* file.

EXAMPLES

Control Files - To illustrate the user-specific access control mechanisms, let us consider a typical user named bob. Normally, as long as bob can authenticate himself, the **authpf** program will load the appropriate rules. Enter the */etc/authpf/banned/* directory. If bob has somehow fallen from grace in the eyes of the powers-that-be, they can prohibit him from using the gateway by creating the file */etc/authpf/banned/bob* containing a message about why he has been banned from using the network. Once bob has done suitable penance, his access may be restored by moving or removing the file */etc/authpf/banned/bob*.

Now consider a workgroup containing alice, bob, carol and dave. They have a wireless network which they would like to protect from unauthorized use. To accomplish this, they create the file */etc/authpf/authpf.allow* which lists their login ids, group prepended with "%", or login class prepended with "@", one per line. At this point, even if eve could authenticate to sshd(8), she would not be allowed to use the gateway. Adding and removing users from the work group is a simple matter of maintaining a list of allowed userids. If bob once again manages to annoy the powers-that-be, they can ban him from using the gateway by creating the familiar */etc/authpf/banned/bob* file. Though bob is listed in the allow file, he is prevented from using this gateway due to the existence of a ban file.

Distributed Authentication - It is often desirable to interface with a distributed password system rather than forcing the sysadmins to keep a large number of local password files in sync. The login.conf(5) mechanism in OpenBSD can be used to fork the right shell. To make that happen, login.conf(5) should have entries that look something like this:

```
shell-default:shell=/bin/csh

default:\
        ...
        :shell=/usr/sbin/authpf

daemon:\
        ...
        :shell=/bin/csh:\
        :tc=default:

staff:\
        ...
        :shell=/bin/csh:\
        :tc=default:
```

Using a default password file, all users will get **authpf** as their shell except for root who will get */bin/csh*.

SSH Configuration - As stated earlier, sshd(8) must be properly configured to detect and defeat network attacks. To that end, the following options should be added to sshd_config(5):

```
Protocol 2
ClientAliveInterval 15
ClientAliveCountMax 3
```

This ensures that unresponsive or spoofed sessions are terminated within a minute, since a hijacker should not be able to spoof ssh keepalive messages.

Banners - Once authenticated, the user is shown the contents of */etc/authpf/authpf.message*. This message may be a screen-full of the appropriate use policy, the contents of */etc/motd* or something as simple as the following:

```
This means you will be held accountable by the powers that be
for traffic originating from your machine, so please play nice.
```

To tell the user where to go when the system is broken, */etc/authpf/authpf.problem* could contain something like this:

```
Sorry, there appears to be some system problem. To report this
problem so we can fix it, please phone 1-900-314-1597 or send
an email to remove@bulkmailerz.net.
```

Packet Filter Rules - In areas where this gateway is used to protect a wireless network (a hub with several hundred ports), the default rule set as well as the per-user rules should probably allow very few things beyond encrypted protocols like ssh(1), ssl(8), or ipsec(4). On a securely switched network, with plug-in jacks for visitors who are given authentication accounts, you might want to allow out everything. In this context, a secure switch is one that tries to prevent address table overflow attacks.

Example */etc/pf.conf*:

```
# by default we allow internal clients to talk to us using
# ssh and use us as a dns server.
internal_if="fxp1"
gateway_addr="10.0.1.1"
nat-anchor "authpf/*"
rdr-anchor "authpf/*"
```

```
binat-anchor "authpf/*"
block in on $internal_if from any to any
pass in quick on $internal_if proto tcp from any to $gateway_addr \
      port = ssh
pass in quick on $internal_if proto udp from any to $gateway_addr \
      port = domain
anchor "authpf/*"
```

For a switched, wired net - This example */etc/authpf/authpf.rules* makes no real restrictions; it turns the IP address on and off, logging TCP connections.

```
external_if = "xl0"
internal_if = "fxp0"

pass in log quick on $internal_if proto tcp from $user_ip to any
pass in quick on $internal_if from $user_ip to any
```

For a wireless or shared net - This example */etc/authpf/authpf.rules* could be used for an insecure network (such as a public wireless network) where we might need to be a bit more restrictive.

```
internal_if="fxp1"
ipsec_gw="10.2.3.4"

# rdr ftp for proxying by ftp-proxy(8)
rdr on $internal_if proto tcp from $user_ip to any port 21 \
      -> 127.0.0.1 port 8021

# allow out ftp, ssh, www and https only, and allow user to negotiate
# ipsec with the ipsec server.
pass in log quick on $internal_if proto tcp from $user_ip to any \
      port { 21, 22, 80, 443 }
pass in quick on $internal_if proto tcp from $user_ip to any \
      port { 21, 22, 80, 443 }
pass in quick proto udp from $user_ip to $ipsec_gw port = isakmp
pass in quick proto esp from $user_ip to $ipsec_gw
```

Dealing with NAT - The following */etc/authpf/authpf.rules* shows how to deal with NAT, using tags:

```
ext_if = "fxp1"
ext_addr = 129.128.11.10
```

```
int_if = "fxp0"
# nat and tag connections...
nat on $ext_if from $user_ip to any tag $user_ip -> $ext_addr
pass in quick on $int_if from $user_ip to any
pass out log quick on $ext_if tagged $user_ip
```

With the above rules added by **authpf**, outbound connections corresponding to each users NAT'ed connections will be logged as in the example below, where the user may be identified from the ruleset name.

```
# tcpdump -n -e -ttt -i pflog0
Oct 31 19:42:30.296553 rule 0.bbeck(20267).1/0(match): pass out on fxp1: \
129.128.11.10.60539 > 198.137.240.92.22: S 2131494121:2131494121(0) win \
16384 <mss 1460,nop,nop,sackOK> (DF)
```

Using the authpf_users table - Simple **authpf** settings can be implemented without an anchor by just using the "authpf_users" *table*. For example, the following pf.conf(5) lines will give SMTP and IMAP access to logged in users:

```
table <authpf_users> persist
pass in on $ext_if proto tcp from <authpf_users> \
        to port { smtp imap }
```

It is also possible to use the "authpf_users" *table* in combination with anchors. For example, pf(4) processing can be sped up by looking up the anchor only for packets coming from logged in users:

```
table <authpf_users> persist
anchor "authpf/*" from <authpf_users>
rdr-anchor "authpf/*" from <authpf_users>
```

Tunneled users - normally **authpf** allows only one session per client IP address. However in some cases, such as when connections are tunneled via ssh(1) or ipsec(4), the connections can be authorized based on the userid of the user instead of the client IP address. In this case it is appropriate to use **authpf-noip** to allow multiple users behind a NAT gateway to connect. In the */etc/authpf/authpf.rules* example below, the remote user could tunnel a remote desktop session to their workstation:

```
internal_if="bge0"
workstation_ip="10.2.3.4"

pass out on $internal_if from (self) to $workstation_ip port 3389 \
        user $user_id
```

FILES

/etc/authpf/authpf.conf
/etc/authpf/authpf.allow
/etc/authpf/authpf.rules
/etc/authpf/authpf.message
/etc/authpf/authpf.problem

SEE ALSO

pf(4), fdescfs(5), pf.conf(5), securelevel(7), ftp-proxy(8)

HISTORY

The **authpf** program first appeared in OpenBSD 3.1.

BUGS

Configuration issues are tricky. The authenticating ssh(1) connection may be secured, but if the network is not secured the user may expose insecure protocols to attackers on the same network, or enable other attackers on the network to pretend to be the user by spoofing their IP address.

authpf is not designed to prevent users from denying service to other users.