Setting up a FreeBSD Wireless Access Point

Scott Gasch

scott@wannabe.guru.org

This article describes how to set up an open wireless access point on FreeBSD 4.9 including the creation of a "captive portal" and bandwidth limiting using IPFIREWALL.

Samples are included along with the descriptions of how it all works. But, because my setup here is probably different than what you have there, you'll probably have to adapt this stuff to your needs. This article assumes a basic familiarity with FreeBSD, **ipfw** and **perl** programming. Please feel free to email me (mailto:scott@wannabe.guru.org) with questions or problems. Also I'm, of course, interested in hearing about errors, omissions, improvements, and success stories!

This document is available in several formats:

- One large HTML file: http://wannabe.guru.org/scott/hobbies/wireless/wireless.html (http://wannabe.guru.org/scott/hobbies/wireless/wireless.html)
- Plain (7 bit ASCII) text: http://wannabe.guru.org/scott/hobbies/wireless/wireless.txt (http://wannabe.guru.org/scott/hobbies/wireless/wireless.txt)
- Adobe PostScript: http://wannabe.guru.org/scott/hobbies/wireless/wireless.ps (http://wannabe.guru.org/scott/hobbies/wireless/wireless.ps)
- Adobe Portable Document Format (PDF): http://wannabe.guru.org/scott/hobbies/wireless/wireless.pdf (http://wannabe.guru.org/scott/hobbies/wireless/wireless.pdf)
- Microsoft Rich Text Format (RTF): http://wannabe.guru.org/scott/hobbies/wireless/wireless.rtf (http://wannabe.guru.org/scott/hobbies/wireless/wireless.rtf)

1. Introduction

When I set up an open wireless access point at my house I was a little worried. Not because I thought I'd get hacked; my wireless subnet is totally separated from my internal network by a FreeBSD firewall. And I wasn't worried about my ISP getting pissed because are actually okay with me sharing my bandwidth. I was worried about two things:

- Random people sucking up all my bandwidth.
- People sending spam, raising hell, etc from my IP address.

This document outlines my approach to handling both problems. It is also a technical guide to setting up a "captive portal" and bandwidth limiting on FreeBSD-4.9.

2. Captive Portal

If you've ever been at a hotel where they charge for Internet access you know what a captive portal is. You plug your laptop into the network but you can't do anything until you open your browser and type in your credit card number on the only webpage you can get to. I wanted the same thing (minus the credit card number, plus an acceptable use notice) on my open wireless access point. People who want to use my bandwidth can do it for free as long as they click on the "I ACCEPT" link and agree not to spam, trade warez, etc. Whether I think this will make any difference at all is another question for a different article.

To accomplish this I used FreeBSD's IPFIREWALL kernel option and the userland **ipfw** utility. My FreeBSD machine was already set up with a firewall and was running NAT. It had one outside static IP address and one IP address on my inside private network. I added a third network card to the machine and hung the wireless access point off this new card. I decided on a range of private (rfc1918) IP addresses for the wireless subnet (different from the private range I was already using on the inside network) and set up my DHCP server to dole out IPs to clients on both networks.

I'll describe how I added a captive portal to this setup but if you need a primer on how to get your basic FreeBSD firewall setup, check out http://www.freebsddiary.org/ipfw.php.

The executive summary of what I did is:

1. install a firewall rule that diverts all wireless http traffic to a webserver listening on the wireless-subnet interface of the FreeBSD machine.
2. install a firewall rule that blocks all other wireless traffic.
3. install a CGI script on the webserver listening on the wireless-subnet that shows my acceptable use policy.
4. When someone SUBMITs on the form served by this CGI script, add a rule to the firewall for their IP address that bypasses the rules in the first two steps (above).

Here's the full sample `rc.firewall` rules. This is also included in the tarball at the end of this article:

```
#
# Some sample firewall rules
#
${fwcmd} -f flush
${fwcmd} add 100 pass all from any to any via lo0
${fwcmd} add 200 deny all from any to 127.0.0.0/8
${fwcmd} add 300 deny ip from 127.0.0.0/8 to any

#               NETWORK TOPOLOGY VARIABLE SETUP
# ------------------------------------------------------------------
# outside (untrusted) interface / netmask and ip
oif="dc0"                 # aka natd_interface
onet="217.254.31.0"       # put your outside network here
```

```
omask="255.255.255.0"      # put your outside netmask here
outside="217.254.31.0:255.255.255.0"
oip="217.254.31.172"       # put your IP address here

# inside (trusted) interface / netmask and ip
iif="xl0"                  # internal if
inet="10.10.0.0"           # my internal network
imask="255.255.0.0"        # my internal netmask
inside="10.10.0.0:255.255.0.0"
iip="10.10.10.1"           # xl0's IP addr

# wireless (untrusted) interface / netmask and ip
wif="sis0"                 # wireless if
wnet="10.30.0.0"           # wireless network
wmask="255.255.255.0"      # wireless netmask
wireless="10.30.0.0:255.255.255.0"
wip="10.30.0.1"            # sis0's IP

# ... stuff omitted for (my network) security and brevity ...

# III. INBOUND VIA WIF PRE-NAT:
# -----------------------------
#    A. Only traffic from the wireless network should arrive via
#       wif.  This stops spoofing by wireless clients.

${fwcmd} add deny all from not ${wireless} to any in via ${wif}

#    B. Wireless traffic cannot talk to the inside network

${fwcmd} add deny all from any to ${inside} in via ${wif}

#    C. Limit access to FreeBSD services for wireless clients

${fwcmd} add allow udp from any to ${wip} 53 in via ${wif}
${fwcmd} add allow tcp from any to ${wip} 8080 in via ${wif}
${fwcmd} add deny ip from any to any 25 in via ${wif}
${fwcmd} add deny ip from any to ${wip} 110 in via ${wif}
${fwcmd} add deny ip from any to ${wip} 995 in via ${wif}
${fwcmd} add deny ip from any to ${wip} 113 in via ${wif}
${fwcmd} add deny ip from any to ${wip} 139 in via ${wif}
${fwcmd} add deny ip from any to any 119 in via ${wif}

#    D. Wireless clients can't access my ISP's pages (in
#       case they are dumb and do authentication based on
#       "traffic from my IP addr == me"

${fwcmd} add deny ip from any to ${myisp} in via ${wif}

#    F. Allow wireless clients to exchange keys for PPTP

${fwcmd} add pass udp from ${wireless} to any 500 in recv ${wif}

#    G. Rate limit traffic from wireless clients -> Internet
```

```
${fwcmd} pipe 1 config bw 5kbytes/s
${fwcmd} add pipe 1 ip from ${wireless} to not ${wireless}

#
# --> DYNAMIC RULES INSERTED HERE <--
#

#    H. Forward all packets from the wireless network to our
#       wireless authorization page

${fwcmd} add 10000 fwd ${wip},8080 tcp from $wireless to any 80,8080,443 in via ${wif}

#    I. Deny anything else from wireless.

${fwcmd} add deny all from ${wireless} to any in via ${wif}

#    J. Jump past target and allow all.  When dynamic rules are
#       inserted they will jump from --> ABOVE <-- to here.

${fwcmd} add 20000 allow all from ${wireless} to any in via ${wif}

case ${natd_enable} in
[Yy][Ee][Ss])
    if [ -n "${natd_interface}" ]; then
    ${fwcmd} add divert natd all from any to any via ${natd_interface}
    fi
    ;;
esac

# ... more stuff omitted for the same reasons ...

#     C. Rate limit traffic from internet -> wireless LAN and
#        traffic from wireless -> internet

${fwcmd} pipe 2 config bw 44kbytes/s
${fwcmd} add pipe 2 ip from not ${wireless} to ${wireless}

# ...

#     Q. Everything else is denied and logged.

${fwcmd} add deny log all from any to any
```

3. The Webserver

The first step in this scheme is to divert all http traffic from wireless clients with unregistered IP addresses to a webserver running on the FreeBSD machine's inside wireless IP address that can send them the acceptable use form by means of a CGI script. I did this via a "fwd" firewall rule. In the sample `rc.firewall` rules above, this is the rule to which I'm referring:

```
ipfw add 10000 fwd ${wip},8080 tcp from $wireless to any 80 in via ${wif}
```

When a packet comes in from the wireless subnet via the wireless interface and is destined for `www.whatever.com` port 80 it will be forwarded to the FreeBSD machine's wireless IP address port 8080. This section of the ipfw(8) ipfw(8) manpage about forwarding is vital:

> "The fwd action does not change the contents of the packet at all. In particular, the destination address remains unmodified, so packets forwarded to another system will usually be rejected by that system unless there is a matching rule on that system to capture them. For packets forwarded locally, the local address of the socket will be set to the original destination address of the packet. This makes the netstat(1) netstat(1) entry look rather weird but is intended for use with transparent proxy servers."

In other words, the webserver at port 8080 on my system is going to be getting TCP/IP packets that look like they are to `www.whatever.com`. Moreover the HTTP requests in those packets are going to be trying to GET the original path of the request. Clearly this webserver has to be able to handle crazy URLs without sending back a 404 File not found.

I'm not an apache guru so maybe what I'm about to tell you is wrong. That said, I could not get my stock http server to do this stuff. Instead I made a new instance of apache just to listen on port 8080 of the wireless IP address and handle the crazy wireless URLs. I include the full `httpd.conf` file in a tarball at the end of this article but the important part is this:

```
AliasMatch .* /usr/local/www/data/wireless/notice.cgi
```

The apache manual on mod_alias describes the AliasMatch directive as follows:

> "This directive is equivalent to Alias, but makes use of standard regular expressions, instead of simple prefix matching. The supplied regular expression is matched against the URL-path, and if it matches, the server will substitute any parenthesized matches into the given string and use it as a filename. For example, to activate the /icons directory, one might use:
>
> ```
> AliasMatch ^/icons(.*) /usr/local/apache/icons$1"
> ```

My regular expression, `.*`, matches anything and directs it to `notice.cgi`, a little **perl** script. Since scripts are enabled by extension on this server the CGI script runs and instead of `www.whatever.com` the unregistered wireless user gets my portal page.

4. The CGI Script

The `notice.cgi` script has two jobs: to display a page full of acceptable use mumbo-jumbo and to handle when people accept it.

The first job is simple, it just throws some HTML out. People probably were surfing to `www.whatever.com` so they don't expect this page. It's friendly, tells them what's up, and gives them a chance to accept my policy.

Then, the same `notice.cgi` script has to handle the acceptance because the wireless client's IP is not unblocked yet -- whatever URL the submit goes to will get redirected to the apache server and matched by the AliasMatch rule... and end up back at the script anyway.

When they accept the policy the script says thanks and adds their IP address to the list that are to be unblocked. A separate daemon process handles unblocking the wireless IP address by modifying the firewall rules. Specifically it adds a rule to bypass the "forward http to the webserver" and "deny everything else" entries in the ruleset. I use a the daemon process because I want to be able to re-block the IP address after some amount of time has elapsed. Since the CGI script is only invoked as a response to to a web hit it is not able to re-block an IP after a timeout.

The last thing the CGI script does is redirect the wireless user to the original website to which they surfed.

The CGI script is included here for your reading pleasure but is also included again (with support HTML files) in the tarball at the end of this article.

```
#!/usr/bin/perl

use CGI qw(:standard);
use MLDBM::Sync;
use MLDBM qw(DB_File Storable);
use MLDBM qw(MLDBM::Sync::SDBM_File);
use Fcntl qw(:DEFAULT);

$file = "/var/auth";
if (param('auth') eq "YES")
{
    &AuthIp($ENV{REMOTE_ADDR});
    &Accepted;
}
else
{
    &Must_Accept;
}
exit(0);


sub Must_Accept
{
    if (!open(FILE, "index.html"))
    {
        print "Content-type:text/html\n\n";
        print "Page is broken, come back later. :(\n";
    }
    else
    {
        print "Content-type:text/html\n\n";
        while(<FILE>)
        {
            print;
        }
        close(FILE);
        print "<A href=" . url() . "?auth=YES>I ACCEPT THESE TERMS</a>";
        print "</CENTER><BR><P></BODY></HTML>\n";
    }
```

```
}


sub Accepted
{
    $foo = url();
    $foo =~ s/\?auth=YES$//;
    print "Content-type: text/html\n\n";
    print "<HTML><HEAD>\n";
    print "<META HTTP-EQUIV=\"Refresh\" CONTENT=\"5;URL=$foo\">";

    if (!open(FILE, "accepted.html"))
    {
        print "Thanks, sorry this is broken a little...\n";
    }
    else
    {
        while(<FILE>)
        {
            print;
        }
        close(FILE);
    }
}

sub AuthIp
{
   my ($ip) = @_;
   my %cache;
   my $sync_dbm_obj = tie (%cache,'MLDBM::Sync',$file,O_CREAT|O_RDWR,0644);
   $sync_dbm_obj->Lock;
   my $time = time();
   $cache{$ip} = $time;
   $sync_dbm_obj->UnLock;
   untie (%cache);
}
```

5. The Daemon

Once the CGI script has decided to unblock an IP address, that IP is placed in a DBM file that is tied to a hash array in a **perl** script. This **perl** script is a daemon process in that it is always running in the background on the FreeBSD machine. Every few seconds it wakes up and...:

- Sees if there are any new IP addresses to unblock.
- Sees if any of the unblocked IP addresses have an expired time limit and must be reblocked.

The script (which I called `wireless-auth`) is listed in this section but is also included in the constantly-referred-to tarball at the end of the article. The rules it adds to unblock a new wireless client's IP address look like this:

```
ipfw add $start_range set 1 skipto 20000 ip from $ip to any in via ${wif}
```

Rules like this are part of one ruleset (set 1) so that they can be deleted easily in one **ipfw** command when the firewall is being rebuilt. All that they do is allow one particular wireless IP address to skip around this part of the firewall rules:

```
# ---> THE RULES SKIP FROM HERE ...

#    I. Forward all packets from the wireless network to our
#       wireless authorization page

${fwcmd} add 10000 fwd ${wip},8080 tcp from $wireless to any 80,8080,443 in via ${wif}

#    J. Deny anything else from wireless.

${fwcmd} add deny all from ${wireless} to any in via ${wif}

# ---> TO HERE

${fwcmd} add 20000 allow all from ${wireless} to any in via ${wif}
```

And here's the daemon itself:

```
#!/usr/bin/perl

use MLDBM::Sync;
use MLDBM qw(DB_File Storable);
use MLDBM qw(MLDBM::Sync::SDBM_File);
use Fcntl qw(:DEFAULT);
use Config;

my $prefix = "10.0.30";      # wireless net prefix
my $gw = "10.0.30.1";        # my wireless IP addr
my $timeout = 60 * 60;       # firewall opening timeout
my $file = "/var/auth";      # IPC file
$ipfw = "/sbin/ipfw";        # path to ipfw
my %current;
$verbose = 1;

open(LOG, ">/var/log/wireless_auth.log") || die $!;

&gen_firewall;
while (true)
{
    my %ips;

    my $time = time();
    $time = $time - $timeout;

    my %cache;

    my $sync_dbm_obj = tie (%cache,'MLDBM::Sync',$file,O_CREAT|O_RDWR,0666);
```

```
$sync_dbm_obj->Lock;

#
# Create a cache hash based on the IPs in the file at this time
#
$cache_hash = 0;
foreach my $ip (keys %cache)
{
    if ($cache{$ip} < $time)
    {
        delete ($cache{$ip});
        print LOG "$ip has expired.\n" if ($verbose);
    }
    else
    {
        if ($ip =~ /\d+\.\d+\.\d+\.(\d+)/)
        {
            $cache_hash ^= $1;
        }
        $ips{$ip} = 1;
    }
}

#
# Create a current hash based on the rules in the firewall at this
# time.
#
$current_hash = 0;
if (open(IPFW, "$ipfw show |"))
{
    while(<IPFW>)
    {
        if (/\d+\s+\d+\s+\d+\s+skipto\s+20000\s+ip\s+from\s+\d+\.\d+\.\d+\.(\d+)\s/)
        {
            $current_hash ^= $1;
        }
    }
    close(IPFW);
}
else
{
    foreach my $ip (keys %current)
    {
        if ($ip =~ /\d+\.\d+\.\d+\.(\d+)/)
        {
            $current_hash ^= $1;
        }
    }
}

if ($current_hash != $cache_hash)
{
    #
```

```
        # Rebuild the firewall...
        #
        print LOG "Detected a change on the firewall...\n" if ($verbose);
        gen_firewall(keys %ips);

        #
        # Update current
        #
        foreach my $key (keys %current)
        {
            delete $current{$key};
        }
        foreach my $key (keys %ips)
        {
            $current{$key} = 1;
        }
    }
    $sync_dbm_obj->UnLock;
    untie(%cache);
    sleep (5);
}


sub gen_firewall
{
    my (@ips) = @_;
    my $start_range = 9000;
    my $date = `date`;

    system "$ipfw -q delete set 1";
    foreach my $ip (@ips)
    {
        print LOG "$date: $ip is authorized.\n" if ($verbose);
        $ips{$ip} = 1;
        system "$ipfw -q add $start_range set 1 skipto 20000 ip from $ip to any in via sis0";
        $start_range++;
    }
}
```

6. Bandwidth Limiting

You can use the **ipfw** to limit the amount of bandwidth that your wireless clients can use by setting up some pipes. Here's an example:

```
pipe 1 config bw 5kbytes/s
add pipe 1 ip from ${wireless} to not ${wireless}

pipe 2 config bw 44kbytes/s
add pipe 2 ip from not ${wireless} to ${wireless}
```

These rules limit your wireless client's outgoing bandwidth to 5k/sec and incoming bandwidth to 44k/sec. They get reasonable performance that doesn't ever swamp my internal subnet.

7. Miscellaneous

I have to give a plug to my ISP because it's a rare thing in this age of fascist service providers to find one with a liberal and progressive policy towards bandwidth sharing. I've also found Speakeasy (http://www.speakeasy.net) to be pretty responsive and customer oriented when I've had to deal with them.

Part of the deal with bandwidth sharing is that I am responsible for what happens from my IP address. So that means that part of the acceptable use policy I serve out with that CGI script is copied from the policy that I have agreed to abide by. In addition I added some stuff I dug out of a tool called **OpenSplash** that seems to have come from a wireless acceptable use policy in San Francisco.

And, of course, here is the promised tarball (http://wannabe.guru.org/scott/hobbies/wireless/wireless.tar.gz) that contains all these sample files along with this document in various forms.

8. Other Tools

I didn't figure this out on my own; in fact large parts of the daemon are lifted from some other tools I found that do captive portals. Some of these other packages only work on Linux. Some of them are for FreeBSD too. In my opinion, all of them purport to be "something you install that makes it work" rather than what I tried to do with this article (namely, tell you how it works and give you samples that you can custom fit to your own system). Nonetheless, you might want to google **NoCatAuth**, **WiCap** and **OpenSplash**. And, of course, thank you to the authors of those tools.