

Fine-tuning firewall rules with Mac OS X 10.4 client

Like many things with Mac OS X, configuring the built-in firewallⁱ is a frequently frustrating combination of the “standard” way of doing things (usually owing from the *nix roots of many open source projects) and “the Apple way” (owing to the decision to make things simple to use for the widest set of users).

Like the server version of the operating system, Mac OS X client uses ipfwⁱⁱ (technically, it’s ipfw2). It uses rules to allow or deny connections for to and from the machine. But with Tiger and below (10.4 and under), it’s neither easy nor intuitive to leverage the power of ipfw to secure your workstation beyond the defaults.

The point of this guide is to modify ipfw rules so that you can, for example, prohibit connections originating from outside the campus (that is, deny connections outside 171.64.0.0/14). That’s what we’ll be doing here—but with some knowledge of ipfw rule constructionⁱⁱⁱ, you can get very fine-grained with your firewall.

We’ll be making a launchd item, which will execute a shell script; that script will then load our custom list of ipfw rules into the kernel, plus turn on logging for the purposes of auditing. The best part of this is that we’re not introducing a new, substitute firewall tool, but working with the built-in software instead, accommodating some of the quirks of Mac OS X client to make it all work.

The unique ways of ipfw on Mac OS X client

With Mac OS X 10.4 client, ipfw is normally configured using the Sharing PreferencePane in System Preferences. The interface is exceptionally simplified and limited, though. For example, the GUI lacks both the option to restrict outgoing traffic, as well as to specify hostnames or IP ranges, both which you can do easily with Mac OS X Server and the Server Admin tool.

Mac OS X is unique because it manages ipfw rules through the extensive use of XML property list (or, more commonly, plist) files, rather than a simple list of rules that get read and loaded in the kernel^{iv}. There are at least two that matter, and there are dependencies involved (which I won't get into). The first is located at

`/System/Library/PrivateFrameworks/NetworkConfig.framework/Versions/A/Resources/DefaultFirewallInfo.plist`. As the name implies, it contains the default rules, plus the services and their ports for the items displayed when you click the "New..." button in the PreferencePane.

The other plist, which we'll work extensively with, is `/Library/Preferences/com.apple.sharing.firewall.plist`.

If you have Apple's Developer Tools^v installed, you can use the Property List Editor tool to easily view the plist file contents. They're normally in binary.

An important note about hand-editing

Because you're kicking off ipfw in a way that's different from the default mechanisms, **you will not be able to use the old GUI way to modify your ipfw settings after doing all this**. Doing so will possibly result in a re-writing of one of your plists and undo your work (or worse). And as is frequently the case with Mac OS X, once you leave the GUI, you're on your own and there's no going back—it's all or nothing.

Also, you should be comfortable using a text editor, such as vi or emacs; alternatively, you can use TextWrangler (the free sibling to the expensive BBEdit). You should know how to use the plutil command line tool, which you will need to use to convert binary plist files into editable XML. You'll need to do some work in single-user mode. And naturally, you'll need administrative privileges to do this.

Setting up services, initial rules

We want ipfw to be kicked off normally with some initial, basic and universal rules, to which we'll later load our own service-related ones written out in a separate file.

The way the firewall actually gets turned on at bootup is rather unique. If you have clicked on the "Start" button in System Preferences → Sharing → Firewall, the GUI will automatically create the `/Library/Preferences/com.apple.sharing.firewall.plist` file. At startup, one of the first files to be read is a shell script, `/etc/rc`.^{vi} In that script, there's a directive that says, "if this plist exists, kick off `/usr/libexec/FirewallTool`."

As you probably know, when you enable services in Mac OS X client, and you have the firewall turned on, the OS will automatically poke the necessary holes for you.^{vii} For example, if you turn on "Remote Login" you'll have a hole created for port 22, which is SSH. You'll also notice that you can't edit most of the built-in services in the firewall configuration GUI; they're grayed out.

Once you have turned on your preferred services and have turned on the firewall, restart your computer in single-user mode.

(I find it convenient to work this way because there aren't competing system calls that may interfere with your configuration, possibly rewriting your edits for example. I've had better luck doing things this way rather than using the Property List Editor while logged in with a fully functioning system.)

Convert your plist

Once in single-user mode, you will need to mount your boot volume to make it read/writable. Normally, at the prompt, you would enter:

```
mount -uw /
```

Use caution. You have root+ privileges here, along with full ability to shoot yourself in the foot.

Use the `plutil` command to convert the plist from binary to editable XML text. Normally, that's something like:

```
plutil -convert xml1 /Library/Preferences/com.apple.sharing.plist
```

And open the file using `vi` or `emacs`. In this example, we'll be using `vi`.

```
vi /Library/Preferences/com.apple.sharing.plist
```

You will want to change some settings in the XML file from 0 to 1 (from "off" to "on" in other words). On the services you've enabled, you'll search for the "editable" keys and change the corresponding integer. To search for "editable," type `/editable` and your cursor will move to the first instance in the file. Use your arrow keys to move down a line and look for `<integer>0</integer>`. If it's a zero, use the "x" key to delete it, then use the "i" key to go into "insert mode." Now you can enter the numeral one "1" in its place. In other words, change:

```
<key>editable</key>  
<integer>0</integer>
```

to:

```
<key>editable</key>  
<integer>1</integer>
```

When you have changed the zero to a one, press the "esc" key to get out of insert mode.

Search for the next instance where `<integer>0</integer>` exists and make the necessary changes. As a tip, you can just type a forward-slash and enter to re-invoke your last search query, speedily finding the next instance of "editable." When you have made all the changes, make sure you save the file by typing ZZ. (Ignore any possible messages about writing viminfo.)

Restart your Mac by typing:

```
shutdown -r now
```

Closing the open ports using the GUI

Once you have rebooted and logged in, go to System Preferences → Sharing → Firewall and actually turn off the previously grayed-out (but enabled) services. This will have the effect of telling Mac OS X to turn on ipfw at startup, even though you have no ports open or configured.^{viii}

Why are we doing it this way? If you left these ports open, you'll have services open to the world, whereas our effort is to close them to everyone but SUNet computers. Likewise, if you just turned off the firewall and tried to invoke it later with your own rules and startup mechanisms, you'd probably have to edit your rc file to kick off the FirewallTool differently, or create your own (now deprecated) StartupItem, which is probably a more difficult path to achieving the same solution. With this method, we're fooling the system into creating and maintaining com.apple.sharing.firewall.plist, just so that ipfw runs at boot, just as it normally would.

Create your ipfw rules

Now comes the fun part. Here's where we create the allow/deny rules that ipfw loads into the kernel—where we can really overcome the limitations of the Mac OS X client GUI.

First, let's look what our rules are currently. Launch the Terminal application and enter this command:

```
sudo ipfw list
```

You should see a list of basic common rules that are common in just about any ipfw configuration. Here's what you might see:

```
02000 allow ip from any to any via lo*
02010 deny ip from 127.0.0.0/8 to any in
02020 deny ip from any to 127.0.0.0/8 in
02030 deny ip from 224.0.0.0/3 to any in
02040 deny tcp from any to 224.0.0.0/3 in
02050 allow tcp from any to any out
02060 allow tcp from any to any established
02065 allow tcp from any to any frag
12190 deny log tcp from any to any
65535 allow ip from any to any
```

There is one rule per line, starting with an increasing leading integer, and are read from top to bottom, in a “first to satisfy” fashion.

When you create your own rules set, don't bother including these lines. They are already included in the plist files created automatically by the PreferencePane interface; they will load along with the rules you construct in another file.

The file is a simple text file that you will probably want to store in the /etc directory (which is the default directory for firewall rules in most other *nix systems). You will need to know how to construct your own ipfw rules to suit your environment. Below is what my example file looks like:

```
add 03070 allow tcp from 171.64.0.0/14 to any dst-port 3283 in # ARD
add 03075 allow udp from 171.64.0.0/14 to any dst-port 3283 in # ARD
add 03080 allow tcp from 171.64.0.0/14 to any dst-port 5432 in # ARD-DB
add 03085 allow tcp from 171.64.0.0/14 to any dst-port 5900 in # ARD
add 03090 allow udp from 171.64.0.0/14 to any dst-port 5900 in # ARD
add 03095 allow udp from 171.64.0.0/14 to any dst-port 5988 in # ARD-WBEM
add 03100 allow tcp from 171.64.0.0/14 to any dst-port 22 in # SSH
add 03548 allow tcp from 171.64.0.0/14 to any dst-port 548 in # AFP
add 03880 allow tcp from any to any dst-port 80 in # HTTP
```

As you can see, my rules are constructed to allow connections from the SUNet. The “deny from everyone else” directive is rule 12190 above, from the default configuration list. If you are not connecting from the SUNet, you’re rejected and logged. The exception is port 80, which is for Apache. Also note that I’ve added a comment to the end of my rules just so I know what they’re for (mostly Apple Remote Desktop, but also SSH and Apple Filing Protocol, aka “Personal File Sharing”). Finally, I started my rules around 03000 just because I wanted to make sure they were above the default 02000 rules, but well below the “final rules” in the 65000 range.

The script to merge your rules

Next, we’ll need a method to merge your rules with those default rules so that the kernel has all the right directives. For this, I used a shell script provided by the author of [WaterRoof](#), which is a great GUI ipfw program used to write rules and learn about the built-in firewall. You can use this application to make both the rules files and the launchd item (see the next section), though you may need to modify it slightly for your environment. **Make sure your permissions are restrictive!** I set my permissions on this file to 744 and make it owned by root:wheel.

I called my rules-merge script “stanford_ipfwrules_merge.sh” which made it necessary to edit the original script to reflect that change wherever the author refers to “waterroof.sh”.

You can view and use my edited rules-merge script here:

http://www.stanford.edu/group/macosexsig/stanford_ipfwrules_merge.sh

You can download the source for the open Xcode project WaterRoof here:

http://www.hanynet.com/waterroof_1.5src.zip

The launchd item to kick off the rules-merge script

You could kick this off using a StartupItem, but Apple has deprecated method in favor of launchd items. This is just a simple XML property list placed in /Library/LaunchDaemons. You can download mine at:

http://www.stanford.edu/group/macosexsig/edu.stanford.ipfw_rulemerge.plist

You must make sure the permissions are strict on this item! Because launchd items are kicked off by root (and run with root privileges) any unauthorized edits could jeopardize your system. Set it to 644 and make sure it's owned by root:wheel.

Here's the simple code:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<!-- This was originally written by hany@hany.net.com for his WaterRoof ipfw editor.
Modified by nbfa@stanford.edu 20070321 -->
<plist version="1.0">
<dict>
    <key>Label </key>
    <string>edu.stanford.ipfw_rulemerge</string>
    <key>ProgramArguments</key>
    <array>
        <string>/etc/stanford_ipfwrules_merge.sh</string>
    </array>
    <key>RunAtLoad</key>
    <true/>
    <key>ServiceDescription</key>
    <string>Stanford ipfw rule merge: load firewall rules</string>
</dict>
</plist>
```

Wrapping it up, final considerations

By now, you have a couple components in place that should have the effect of making your ipfw configuration on your Mac OS X client machine more granular.

- 1) Your modified `/Library/Preferences/com.apple.sharing.firewall.plist`
- 2) Your new launchd item at `/Library/LaunchDaemons`
- 3) Your new rules-merge script, likely in `/etc`
- 4) Your new set of ipfw rules, likely in `/etc`

In my testing, this has been a pretty resilient configuration. I can actually turn on services later and not have my rules re-written. Of course, if I turn on a service for which I haven't opened the corresponding ports in my rules-merge file, connectivity will be denied.

I can also *try* to open the Firewall tab in System Preferences → Sharing, but I get the expected message stating that some other system is managing the firewall and I'm denied access. It's important, then, to make your ipfw rule changes only in your rules-merge file.

Helpful references

<http://www.macdevcenter.com/pub/a/mac/2005/03/15/firewall.html> is a terrific article on ipfw and Mac OS X.

http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/firewalls-ipfw.html is an authoritative, comprehensive guide for ipfw, which is typically associated with BSD-flavored OSes (like Mac OS X). This may not apply 100% to the Apple distribution, but it's a good read.

<http://www.hmug.org/man/8/ipfw.php> is the HTML version of the man page for ipfw, posted by the Huntsville Macintosh User Group.

<http://docs.info.apple.com/article.html?artnum=107846> talks about how Mac OS X 10.3 and 10.4 *Server* manages its firewall settings and logging preferences.

http://www.nsa.gov/snac/downloads_macOSX10_4Server.cfm?MenuID=scg10.3.1.1 is a newly revised guide from the National Security Agency on securing Mac OS X 10.4, with instructions for both server and client systems.

<http://www.hmug.org/HowTos/tcpwrappers.html> a how-to guide for using TCP Wrappers, another way to restrict hosts.

ⁱ People can get really picky about what Apple calls a “firewall” being really a “packet filter” —heated, nerdy arguments over such semantics are not uncommon. For my purposes in this document, I’m going to side with Apple, be contentiously indiscriminating and use the term “firewall.”

ⁱⁱ ipfw is defined in its man page as “IP firewall and traffic shaper control program.” Consult the man page for ipfw; it’s a wealth of information. Just type “man ipfw” in your Terminal application or visit the link in the “Helpful resources” section.

ⁱⁱⁱ See the end of this document for some resources on ipfw rule construction.

^{iv} Because the ipfw rules are loaded into the kernel, you will not see discrete ipfw or firewall processes managing security.

^v See <http://developer.apple.com/tools/> to download these free and nifty tools.

^{vi} For an excellent overview of how Mac OS X starts up, check out this article on the web: http://www.kernelthread.com/mac/osx/arch_startup.html.

^{vii} It’s not clear to me what process is responsible for this hot, automatic configuration action, but I suspect NetCfgTool calling /usr/libexec/FirewallTool.

^{viii} You have some basic rules still in effect, though, even though there aren’t any specific services. We’ll examine those later.