User’s Guide to nftfw

User’s Guide to nftfw

What does nftfw do?

nftfw provides a simple-to-use framework generating rules for the latest flavour of packet filtering for Linux, known as *nftables*. It generates a set of incoming rules, outgoing rules, supports a whitelist for ‘friends’ and a blacklist for miscreants. *nftfw* glues these rules together and loads them into the system’s kernel to act as your firewall.

There are five control directories in */etc/nftfw* (or it may be */usr/local/etc/nftfw* on your machine). You tell *nftfw* to make a rule by adding a file, that’s often empty, to one of these directories. *nftfw* uses the file name to understand what you are asking, and if needed, will use the file contents to configure the rules it makes.

Here are the directories:

- *incoming.d* The incoming directory provides information to permit selected inbound connections through the firewall. It contains files that add rules giving the ports that should be available to external users. When a file has no content, its rule applies to all comers. Adding IP addresses into the file constrains the rule to apply only to those addresses.
- *outgoing.d* The outgoing directory behaves in the same way as the incoming rule set except that it’s designed to filter outbound connections. When a file has no content, its rule applies to all destination IP addresses. If needed, adding specific destination addresses as content to the file modifies the rule.
- *whitelist.d* The whitelist directory contains files named for IP addresses, it makes rules to inspect inbound connections to the system. Packets from these addresses can always access the machine. Adding port numbers as contents to the a file modifies the rules allowing the IP address to only access certain services. There’s also an automatic scanner looking for successful logins into your machine that will create files in this directory.
- *blacklist.d* The blacklist directory has similar contents to the whitelist but will block any attempt to access the system from the IP address. Adding port numbers as contents to the files modifies the rules to only block access to those services. There’s an automatic system that looks in log files for people doing bad things and adds their IP address into this directory.
- *blacknets.d* The blacknets directory contains files ending in *.nets*, each file can contain a list of IP network address ranges in CIDR format. Ranges enable the firewall to use fast logical operations on numbers to see if an IP address should be be blocked rather than needing a single rule for each IP. Using blacknets, it’s possible to cheaply stop access to your server from one or more countries, or from other large organisations with a diverse address range.

All of these directories create a list of rules. The order of the list is important. The firewall passes each packet from one rule to the next trying to match the data in the packet with the tests in the rule. Some rules will be looking for matching IP addresses, some for ports and some for both addresses and ports. When the firewall finds a match, the rule tells it to make one of two decisions: accept the packet or reject it.

The firewall is a filter, continuing with testing until it finds a decision. For inbound packets, the firewall passes the packet into:

- the whitelist rules accepting good guys, then
- the blacknet rules blocking a wide range of addresses, then
- the blacklist rules rejecting known bad guys, then
- and finally the incoming rules making decisions about all others.
- If the packet falls out the bottom, then it's automatically rejected.

For outbound packets, the firewall will accept packets that have no match with any rule.

If you consider the volume of packets travelling through any machine, firewall testing seems to be placing a lot of processing between the inbound network interface and the program ready to receive the next packet for that port. Processing is considerably reduced by knowing whether the packet is already part of a conversation or is just starting one. An early rule in the firewall accepts packets that are part of a conversation, applying the testing only when the conversation starts. The firewall filters out most packets before starting testing against the rules from the directories.

incoming.d

Here's what is present in the *incoming.d* directory for a standard installation:

```
$ ls incoming.d
05-ping        10-https        30-imaps  50-smtps
06-ftp-helper  20-ftp          40-pop3   50-submission
07-ssh         21-ftp-passive  40-pop3s  60-sieve
10-http        30-imap         50-smtp   99-drop
```

None of these files have any content, they are just a filename. Each filename is a two digit number, a minus sign and a name. The number provides sorting value for the files and firewall entries made from this setup will appear in the order that you see. As we've seen, order is important.

There are three possible types for 'name' section of the filename:

- A port number;
- the name of the service in */etc/services*;
- the name of an action file found in the directory *rule.d*.

If the name part is a port number or a service name, then *nftfw* uses the default *accept* action from *rule.d* to make the necessary rules. The action is given the port number from the name or from the matching entry in */etc/services*. For example, the file *10-http* will create an accept rule for port 80, *10-https* will create an accept rule for port 443.

It could be that you are not using the POP protocol for inbound mail delivery and you can remove *40-pop3* and *40-pop3s*. These services will still be available, but perhaps you don't want to offer them to the world.

If you want a service only to work for a limited number of specific IP addresses, just add the addresses one per line to the file in *incoming.d*. You may add a domain name instead of an IP address, and *nftfw* will lookup the name, translating it to actual IP addresses (both types of address: IPv4 and IPv6 if available). It's a good idea to run a caching nameserver on your machine if using names, to prevent slower offsite lookups.

The *07-ftp-helper* is not a service name. It's an action file in *rule.d* named *ftp-helper.sh*. It adds in the essential glue that makes the *ftp* server work. It's not needed it you don't support *ftp*.

If you need to create a special script for a standard service, then you can do so. *nftfw* gives precedence to action files in *rule.d* with the same name as a service.

There are several unused rules in the *rule.d* directory, a text file called *README* lists them.

outgoing.d

There are no files *outgoing.d* directory for a standard installation. There are some essential rules built into *nftables* template for IPv6.

Files in the directory behave the same as those in *incoming.d*, except that when they contain IP addresses, then those addresses will match the destination IP address of the packet that's tested.

The default setting *outgoing.d* is: if no rule matches the filtered packet then it's accepted and passed on. Adding a file that uses a service name or a port number will load a 'reject' rule, for example (you may not want to do this):

```
40-ssh
42-80
```

will block your machine from sending to an external *sshd* server, and also to any external websites. If you want to block access to a specific site offering these services, then you can add IP addresses to the file.

blacklist.d

To blacklist an address, create a file in *blacklist.d* named by the IP address to add the address into the firewall. The*touch* command is a simple w ay to make an empty file.

```
$ touch 198.51.100.40 '2001:db8::6|64'
$ ls blacklist.d
198.51.100.40   2001:db8::6|64  203.0.113.204.auto
```

The first IP address is a version 4 address, and the second a version 6 address. It's usual to match only the top half of an IPv6 address, this is normally written `2001:db8::6/64` but we can't use '/' in a Linux file system, and the convention is to replace this by the vertical bar '|' symbol. Note we have to quote the address when used on the command line. The blacklist scanner (see below) creates files ending in *.auto*, you should leave these alone, the scanner will take care of them.

An empty blacklist file will block access to all ports to the IP address given by the name. Adding port numbers into the file, one per line, will restrict access to only those port numbers. The word 'all' can also be added, blocking all incoming ports. If a file contains 'all', other port numbers are ignored.

?If your system doesn't have *systemd* active directory installed, you will need to run

```
$ sudo nftfw load
```

after you've changed the directory contents, or wait for the next automatic update.

whitelist.d

The whitelist directory follows the same basic pattern used to manage the blacklist directory. To whitelist an address, create a file named by the IP address. The file content can hold ports if you only want to allow access to specific services.

The whitelist scanner will create files ending in *.auto*. See below.

If your system doesn't have *systemd* active directory installed, you will need to run

```
$ sudo nftfw load
```

after you've changed the directory contents, or wait for the next automatic update.

blacknets.d

The only rule about filenames in the *blacknets.d* directory is that they should end with *.nets*. Each file should contain a list of network address ranges, one per line, in CIDR format. See Wikipedia if you need more information on the notation. The files support comments starting with '#'.

Mostly, you can find suitable lists on the web, see Getting CIDR Lists for how to install them. However, there's nothing to stop you creating your own lists.

Starting with *nftfw*

The root user runs_nftfw_ from the command line to create the firewall:

```
$ sudo nftfw load
```

It can be run from the *systemd* daemon whose *nftfw.path* service (when installed) triggers a call to *nftfw* when files change in one of the action directories. As a catch-all,*cron* will run the command once an hour.

The *load* command tries not to make changes to the kernel's tables unless it has to. It creates a set of files that are needed for the complete ruleset and then compares those files with the set that was made on the last run. If the files are identical, no change is needed. The blacklist and whitelist rule sets can be replaced without changing the whole ruleset. The blacklist rules change most frequently, and an update to the blacklist is done without disturbing the remainder of the firewall. In general, all the rules maintain counts of matches, and these counts are reloaded when the complete firewall is replaced. By updating only parts of the firewall, these counts become a good indicator of activity on your system.

The *-f* or *–full* flag to the *load* command forces the new files to be loaded without reference to the last run. Consider using the *-f* flag to force a full reload if you think that the firewall isn't working.

nftfw uses a configuration file in */etc/nftfw/config.ini* to supply tailoring of various aspects of its operation. The distributed version contains a complete set of the default settings, with the values commented out. The *-i* flag to *nftfw* lists all the values that are in force.

nftfw will write error messages into */var/log/syslog* using the standard *syslog* mechanism. To get an listing of what *ntfw* is doing, set *loglevel* in the config file to *INFO*.

There are various manual pages - see Manual page index.

If you are migrating from another firewall system to *nftfw*, now's the time to look at the Migrating to nftfw section of the Installation document.

Blacklist

nftfw contains a scanner whose job is to watch log files and add offending IP addresses into *blacklist.d*. The scanner is started by:

```
$ sudo nftfw blacklist
```

and that's usually run from *cron* every 15 minutes. If it makes any changes, it will reload the firewall, usually only changing the parts of the firewall that it uses.

The rules for scanning are supplied by a set of files in */etc/nftfw/patterns.d*. Files here are named by *'pattern-name'.pattern*. Here's part of my *apache2.pattern* file:

```
#
#  The file we scan for patterns
file  = /var/log/apache2/access.log
# the ports we block, can be the word all
ports = 80,443
#
#  The patterns we use
__IP__.*wget%20http
__IP__.*XDEBUG_SESSION_START=phpstorm.*$
__IP__.* CONNECT .*$
__IP__.*%20UNION.*$
```

Comments are useful, and comment lines are shown by putting a # at the start of the line.

The file is split into two sections. The first couple of statements set the file to be scanned and the ports to be blocked if this pattern is matched. The rest of the file contains several regular expressions that match lines in the logfile.

The file statement can contain shell ‘glob’ patterns, for example on a Symbiosis/Sympl system, you can scan all the website log files by using:

```
file=/srv/*/public/logs/access.log
```

When scanning files, *nftfw* remembers the last position it reached in the a file and restarts from that position. When adding a new expression to capture some bad line you’ve spotted in a log, it can sometimes be confusing that the bad guy doesn’t suddenly appear in the blacklist. *nftfw* supplies a way round this using the ‘ports’ statement, see below.

The regular expressions contain the ‘magic’ string `__IP__` (two underscores at each end) matching an IPv4 or IPv6 address in the line. If you are a *regex* novice, then `.*` matches ‘anything’ and the $ matches the end of the line. Most matches can be specified using these simple constructs. Character case is ignored when the expression is matched against lines from the file.

Regular expressions use several characters to mean something special, and if you want to match these characters in a line from a log then they must be preceded by a backslash (). The characters are:

```
\ . ^ $ * + ? ( ) { } [ ] |
```

It’s important to use backslash before these characters to make sure that the expression means what you want it to mean. You can use any of the Python regular expression syntax to match lines, but *nftfw* will complain if unescaped (..) strings appear, these indicate a ‘match group’, there should only be one - the `__IP__` . You *can* use ‘non-capturing groups’. For example, to provide alternation, an expression containing `(?:word1|word2)` will match ‘word1’ or ‘word2’ at that position in the line.

nftfw will ignore matched addresses that are not ‘global’, so if your system is a gateway with a local network running from it, local network addresses are not blacklisted. It also ignores addresses found in the *whitelist.d* directory. Using the whitelist avoids any entry for ‘good’ addresses appearing in the blacklist database.

When *nftfw* finds a match, it stores the IP address, the ports, the time of first encounter, the time of last encounter and the matching pattern name in a database. It also stores a running count of matches found and the number of ‘incidents’, the latter is the number of runs of the scanner it has taken to make the match count.

If the match count is over a threshold number, defaulting to 10, and settable in the *config.ini* file, the scanner will add a file to the *blacklist.d* directory, triggering a run of *nftfw load* to put the ip address into the blacklist blocking against the nominated ports. When the match count gets over a second threshold, default 100, and also settable in *config.ini*, the blacklist entry is promoted to blocking all ports.

The current state of the blacklist can be seen by using

```
$ nftfwls
```

and the *-a* option to this command lists all the entries in the database. *nftfwls* has a manual page, see the Manual page index.

The ports=update option

Once in the firewall, miscreant sites will continue to knock at the door. They are generally robots, and it’s good to know if they are still knocking so we can keep them in the firewall. A firewall rule with logging will write a record into */var/log/syslog* and the database is updated using the special port value of *update*. Here’s *blacklist-update.pattern*:

```
#  Blacklist feedback pattern
#
#  The file we scan for patterns
#
file  = /var/log/syslog
#  Just update the database
ports = update
#
#  The patterns we use
#  depends on Blacklist Logging in place
.*kernel.*Blacklist.*SRC=__IP__.*$
```

when detected, the port action just updates the database counts and time, so you can track if the bad sites have really gone away.

Testing regular expressions

It's often the case that you see a line in a logfile and think 'I ought to have a rule for that'. Testing the new line you've just added can be difficult because the logfile reader remembers file positions and won't see the line again unless the action re-occurs.

nftfw can use a special pattern file, setting *ports=test*, that allows you to test the regular expression against some known content to see if it matches. Copy the pattern file that you want check out to a new file, here's a copy of *apache2.pattern* in *apache2-test.pattern*.

```
#
#  The file we scan for patterns
file  = /var/log/apache2/access.log
# the ports we block, can be the word all
ports = test
#
#  The patterns we use
__IP__.*TESTING EXPRESSION
```

The blacklist scanner will ignore any pattern file with *ports=test*, but it can be used with the single package selection option to *nftfw*. Here's the testing command:

```
$ sudo nftfw -x -p apache2-test blacklist
```

will use data from *apache2-test.pattern* and will scan the named log file. The-*x* flag scans the log file from the beginning and will not update the stored file position. The command will print a table with any matching IP addresses, along with a match count. The command can be re-run if matches fail after adjusting the regular expression in the pattern file.

Whitelist

The whitelist scanner is started by

```
$ sudo nftfw whitelist
```

and is usually run from *cron* every 15 minutes, usually 5 minutes after the blacklist run. If it makes any changes, it will reload the firewall, usually only changing the parts of the firewall that it uses.

It's job is a little simpler than the task faced by the blacklist, it adds the IP addresses used by users of the machine that have logged in from a global IP address into the *whitelist.d* directory.

The whitelist command looks in the system's *wtmp* file that records all user logins and system reboots. It can be set to look in the*wtmp* file that just contains toda y's activity by changing the *wtmp* value in _config.

Rules - rule.d

The *rule* directory contains small shell scripts that translate firewall actions named in the *incoming.d* and *outgoing.d* directories into nftables command lines. Default rules are also used for the whitelist and blacklist generation. Note the coding and management of these files are different from Symbiosis, but the same idea is there, a shell file allows easy additions by users. The files do not run any commands, they output *nftables* statements to *nftfw* which stores them and passes the completed file into the *nft* command.

nftfw runs the scripts though the shell and captures the output text, appending it to a file holding nftables commands. The system calls each action file twice, once for IPv4 and again for IPv6. The processing script uses environment variables to pass parameters into the shell. The parameters are:

- DIRECTION - is set to either ‘incoming’ or ‘outgoing’. The value is most often used to select whether the rule should apply to source or destination IP addresses.
- PROTO - is set to either ‘ip’ or ‘ip6’. These names not only supply the protocol type, but also are the names of the two main tables that form the basic framework, one for each protocol type.
- TABLE - is the name of the table, this is usually ‘filter’, again defined by the basic framework.
- CHAIN - the chain used to add the rule to.
- PORTS - selects the ports that the rule should apply to, the value can be empty, a single port, or an *nftables* anonymous set, several ports separated by commas and wrapped in {} braces.
- IPS - where the rule needs to apply to specific IP addresses, the command will have IPS set. Like the PORTS value, ir can be empty, a single port, or an *nftables* anonymous set, several ports separated by commas and wrapped in {} braces.
- COUNTER - adds the ‘counter’ statement to the rule. The value is usually empty, or the word ‘counter’
- LOGGER - Finally the LOGGER value is either empty or contains ‘log prefix “String”’, adding a space after any supplied string from *nftfw*.

There are several examples of these scripts in the */etc/nftfw/rule.d* and the README file in that directory explains what they do.

Rules - local.d

If you want to supply your own rules, or override standard rules in *rule.d*, new and alternative rules can be placed in *local.d*.

Other documents

All documents can be found on the web from the *nftfw* website.

See documents in the *docs* directory:

- Installing *nftfw*
  - Full installation of the system for Debian Buster.
- Installation Instructions
  - For those who want a bare bones list of tasks.
- Installing Geolocation
  - Installing Geolocation, adding country detection to *nftfwls*, which is optional but desirable.
- Getting CIDR lists

  - How to get CIDR files for use with the *blacknet* feature..
- sympl-email-changes - changes to Sympl buster email installation
  - I’ve added a repository that steps through the changes I make to the standard *exim4*/*dovecot* systems on Sympl to improve feedback and detection of bad IPs.
- Updating *nftfw*
  - How to update *nftfw*.
- How do I.. or Quick User’s Guide
  - Answers a bunch of questions about the system.
- Manual Page index
  - Manual Page index

Acknowledgement

All of this is made possible by shamelessly borrowing ideas from Patrick Cherry who created the Symbiosis hosting package for Bytemark of which the firewall system is part.