Global Information Assurance Certification Paper

Nftables as a Second Language

GIAC (GSEC) Gold Certification

Author: Kenton A. Groombridge, kgroombr@comcast.net
Advisor: Christopher Walker
Accepted: May 10th 2015

Abstract

The iptables Linux kernel firewall has been around for a long time and many Linux users are well versed in it, but now a new player in town, nftables, is now merged into the Linux kernel source and is touted to replace iptables. "What, another thing I have to learn" you say? Don't fret, rather than take on nftables as a completely new language, use what you know of iptables and apply it.

1. Introduction

The iptables kernel firewall was released in 1998 and is considered the standard for configuring Linux firewalls. Since it has been around for seventeen years, as the writing of this document, it has it has proven to be a very capable, effective firewall. The command "iptables" is only one component of the iptables version of the Linux firewall. Although the term "iptables" is often considered just a command, it refers to the command and the kernel options with which it interacts as the command itself is useless without selecting, building, and installing the Linux kernel components for iptables.

The nftables kernel firewall application is relatively new as it was released with Linux kernel version 3.13 on 19 January 2014. It is also a capable, effective firewall, and, like iptables, it relies on selecting the required optional kernel components when building the kernel. Nftables is under active development and not all features of iptables incorporated, but it is sufficient for most host-based firewall applications.

Both iptables and nftables are capable of more than just filtering packets as they can also perform Network Address Translation (NAT), and packet mangling. It is up to the user to choose between the two; however, though not recommended, iptables and nftables can both be used simultaneously. Typical users will find nftables to be more flexible with a more intuitive syntax.

The iptables firewall has several predefined tables and filters. With nftables, nothing is preconfigured, so it is up to the user to configure it to their choosing. Some might consider this a step backward, but realize that preconfigured tables and chains that are not used reduces the available memory and performance of the system. Iptables, by default, counts the number of packets and bytes per chain and per rule (displayable with the '-v' switch). Again, having a feature and never using it impacts memory and performance. Nftables makes counters optional

and is easily added on a per rule basis. Iptables was limited to having one action per rule, while nftables can have multiple actions per rule.

A common iptables rule was usually littered with many switches and options preceded by either a dash '-' or double dash '--'. Although it made it simpler for the programmer of the command to parse the command line, it reduced the readability of the rule. The nftables command uses a flow similar to that of the Berkeley Packet Filter (BPF) syntax which more closely resembles a sentence rather than a Linux/UNIX command. Although the readability is improved, it may have been better to actually utilize the BPF syntax to lessen nftables learning curve.

Although this document is focused to those with experience with iptables so that they can utilize what they already know and start using nftables, it will provide value to those with no iptables experience as it covers several scenarios in the usage of nftables.

2. Getting Started

Although iptables is a mature product, nftables is under active development and is changing rapidly. The Linux kernel used in these examples is 3.19.1. The iptables version is 1.4.21, and the nftables version is 0.4. Discrepancies in the functionality of the tools used in this document may differ due to functionality differences that may occur with each version.

Before using iptables or nftables, the optional kernel components must be configured, built, and installed. For those that find tinkering with kernel options challenging, it is recommended to research what options to build for the planned usage; however, to quickly get started, configure the kernel for automatic module loading/unloading and select all options under "Network Packet Filtering Framework (NetFilter)" as modules. Build and install the kernel and modules, then boot from the newly built kernel. Doing this will enable the system to load the appropriate module(s) when entering iptables and nftables commands.

Kenton A. Groombridge, kgroombr@comcast.net

It is possible to use iptables and nftables as the same time; however, this is highly discouraged. Since both of will be acting on packets, this may cause conflicts and perform unintended actions.

Again, the goal is to take information known from iptables and apply it to nftables. To facilitate this, the intended action followed by the iptables and nftables version of performing that action along with dialog explaining the commands. By comparing the commands side by side will assist in translating from iptables to nftables.

3. Iptables and nftables Terminology

To ease the understanding iptables and nftables, it is best to grasp some of the terminology used. Fortunately much of the terminology is interchangeable between the two.

3.1 Tables

Table are containers for chains.

Iptables has five predefined tables: filter, mangle, nat, raw, and security. The filter table is the table where actions related to firewalling typically take place. The filter table is the default, so if a table isn't specified with the iptables command, then the filter table is employed. The nat, mangle, raw, and security tables are beyond the scope of this document.

For nftables, there are no default tables so they must be created as needed. Nftables uses what are called families which can be any of: arp, bridge, inet, ip, and ip6. When not specifying a table family in an nftables command, the default is ip which is used for chains containing IPv4 rules. The ip6 table family is used for chains containing IPv6 rules. The inet table family is used for chains containing both IPv4 and IPv6 rules. The arp and bridge families will not be discussed as they are beyond the scope of this document.

3.2 Chains

Chains are containers for rules.

With iptables, there are applicable preconfigured chains for each table type. The source of the packet determines which chain will be used to process the packet. There are five base chains, but some may not be pertinent to specified table. The five iptables base chains are:

PREROUTING: Packets entering the system traverse this chain before a routing decision is made.

INPUT: Packets addressed to the local system will traverse this chain. It is used to control external connections to the local system.

FORWARD: Packets not addressed for the local system that are to be routed traverse this chain.

OUTPUT: Packets originating from the local system to be sent outward will traverse this chain. It is used to control external connections from the local system.

POSTROUTING: Packets exiting the system traverse this chain after a routing decision has been made.

Base chains are chains that are connected via netfilter hooks, and through this connection, they will process traffic. Iptables allows the creation of user-defined chains. User-defined chains are not associated with any netfilter hook; therefore, by default, do not process traffic but can be used to arrange rule sets and then using jump-to chains.

As there are no preconfigured tables with nftables, there are also no preconfigured chains. Base chains and non-base chains can be created depending on the intended purpose. Base chains are the same as those with iptables and what iptables refers to as user-defined chains are called

non-base chains with nftables. There are three nftables chain types and their name suits their description.

filter: Packets to be filtered will require this chain.
route: Packets that require packet mangling will need this chain.
nat: Packets requiring NAT will utilize this chain.

Only certain hooks are available depending on the type of chain selected. There are five hooks that are available with nftables: prerouting, input, forward, output, and postrouting. These hooks names are the same as the five types of iptables chains, except they are in lowercase, and the function of each is identical to that of its counterpart in iptables.

3.3 Rules

Rules dictate what to match and what actions are performed on packets when there is a match.

The majority of the difference in the syntax between iptables and nftables is with the rules. When a packet matches a rule, the action specified on that rule will mandate what happens to the packet. This works the same for both iptables and nftables. One major difference is that iptables has a feature called “default policy” that doesn't exist in nftables.

By default, with iptables, if all the rules in the chain are traversed and there is no match, then the packet is allowed, but this can be overridden by changing, with the appropriate command, the default policy to drop packets not matching any previous rule. This is often preferred for security reasons so that any packet not explicitly allowed, will be denied or dropped.

Although it is stated that nftables doesn't have a default policy, it may better be described that nftables has a default policy that cannot be changed. With nftables, if all the rules in the chain are traversed and there is no match, then the packet is allowed; however, there is no command that can change this behavior. If it is required to drop all packets that are not matched against any rule in the chain, then a rule must be placed at the end of that chain to drop all remaining packets.

4. Common Firewall Configurations

Covering every possible syntax and keyword comparing iptables to nftables isn't feasible for the scope of this document so it will be limited to discussing utilizing these tools in creating basic host firewall rules. Even limiting to this, the following examples only touch the surface of the possibilities, but it covers a foundation of examples that can be utilized in nearly all types of rule writing.

Creating a host firewall is one of the most common applications of iptables or nftables. Before implementing firewalls on a Linux system, careful planning should take place before hacking away at the keyboard. When no jump chains are used, both iptables and nftables each traverse the rule chain from top to bottom, one by one and if a match is found, the rules action is performed and subsequent rules are not examined. If a rule allowing a packet is listed before another rule blocking the same packet, then that packet will be allowed. If the intension was to block that packet, then the rules will have to be reordered so the rule blocking the packet will precede the rule allowing the packet.

Tables 1 through 4 found in the appendix are given to assist in the translation of iptables command options to nftables command options. The tables do not contain a comprehensive list of options, but cover the more common ones which are used in the following examples. For a complete list of command options, refer to the appropriate manual page or online documentation.

Now that some of the peculiarities between iptables and nftables has been hashed out, it is time to write some configurations. The IP and network addresses used for these configurations are completely arbitrary other than RFC 1918 addresses are used to ensure anonymity.

Creating tables and chains

As discussed earlier, iptables has predefined tables and chains that are immediately available when the applicable kernel modules are loaded so rules can created without any previously entered commands. These predefined tables and chains cannot be created or removed as needed; however, iptables does support the creation of user-defined chains. For example, to perform logging with iptables, a user-defined chain can be created that will contain the rules that will perform logging. Logging with iptables will be discussed later. Any name can be used, but style prescribes that it should be uppercase as it is common practice to use uppercase for user-defined names. Iptables is case sensitive so the name "LOGGING" is not the same as "logging".

```
# iptables -N LOGGING       # Create a new user-defined chain named LOGGING
```

With nftables, there are no preconfigured tables and chains, so they will first need to be created. This may appear like this is a step backward as tables and chains will have to be created before ever writing a rule, but the added flexibility eventually pays off.

First a table for the appropriate address family must be created. The command is "nft" followed by "add table" to add or create a new table. This is then followed by the address family of "ip" as this table will be used for IPv4 addresses. As "ip" is the default family, this is optional, but again, should be added for readability and consistency. Then lastly, is the name of the table "FILTER". This name can be any name and most of the documentation uses lowercase for table names. It is highly recommended that all user chosen names be uppercase as it will greatly add to the readability of the configuration and ensures there is no confusion between keywords and user chosen names.

Kenton A. Groombridge, kgroombr@comcast.net

```
# nft add table ip FILTER      # Create table name FILTER
```

Once the table is created, a chain referencing an existing table must be created. The command is "nft" followed by "add chain" to add or create a new chain, followed by the address family of "ip" as this table will be used for IPv4 addresses. This is then followed by the name of the previously created table "FILTER" as the chain name must be placed within an existing table. If creating a non-base chain as in the previous iptables example, this is all that is required:

```
# nft add chain ip FILTER LOGGING          # Create non-base chain named LOGGING
```

Most of the time, a base chain will be created. The syntax is the same as creating a non-base chain except an additional section called the chain configuration will be appended to the command. The chain configuration portion is required when creating a base chain so that it registers via the appropriate hook so that traffic will be inspected.

The chain configuration is enclosed in braces "{}". It follows a "keyword value" pattern. The first keyword in the chain configuration is "type" is followed by the value for the type of chain, in this case "filter" which creates a filter chain for filtering packets. This, in turn is followed by the keyword "hook", followed by value "input" so that this chain is registered with the kernel's input hook in order for traffic to be inspected as it enters this system. Then this is followed by the keyword "priority" which is followed value of priority which is a signed 32 bit number from -2147483648 to 2147483647. The chain configuration portion must be terminated with a semicolon ";", but because the semicolon has special meaning and is parsed by the bash shell used in this example, it must be escaped by preceding it with a backslash "\".

```
# nft add chain ip FILTER INPUT { type filter hook input priority 1 \; }
```

Chains are evaluated from the lowest to highest priority value. In essence the lower the number, the higher priority. In this previous example, the priority value of one "1" was chosen as it is an appropriate value to use for filtering (See Table 5: Chain Priorities). The priority number used should consider internal operations. For example, when using connection tracking, if specifying a priority value of -401or lower in a rule, then the rule will be evaluated before defragmentation takes place. Having this ability provides great power with nftables rules, but it may cause unintended consequences if not aware of the chain priority values and their function.

Nftables chains must be created for every type of chain required. If there will be input and output filtering, then two chains, one for input, and one for output filtering require creation. Multiple chains can be placed within one table so both chains can reference the FILTER table in this example. Not all examples in this document require an input and or output chain, but the following can be used to create the base structure in so that the examples do not cause an error due to a missing tables or chains:

```
# nft add table ip FILTER
# nft add chain ip FILTER INPUT { type filter hook input priority 1 \; }
# nft add chain ip FILTER OUTPUT { type filter hook output priority 1 \; }
```

Flushing rules from tables and chains

Before writing a new set of rules, it is good practice to start with a clean slate. With iptables, to flush all the rules from all the chains, use the -F switch by itself, and to remove only rules from specified chains, use the "-F" switch followed by the specified chain. One disadvantage to iptables is that there is no consistency in character case of the command. The table name "filter", must be in lowercase, and the chain name "INPUT" or "OUTPUT" must be in uppercase. This make differentiating user defined names from keywords which makes the command slightly more difficult to read to an inexperienced user.

```
# iptables -t filter -F              # Flush all rules from all chains
# iptables -t filter -F INPUT   # Flush all rules only from the INPUT chain
# iptables -t filter -F OUPUT # Flush all rules only from the OUTPUT chain
```

Because nftables doesn't have default tables and chains, they will have to be created before flushing them. This may sound counterintuitive, but there may be existing rules from a previous configuration that would be easier to flush and start over than to edit or the rules are no longer required.

```
# nft flush table ip FILTER    # Flush all rules from all chains under the FILTER table
# nft flush chain ip FILTER INPUT      # Flush all rules only from the INPUT chain
# nft flush chain ip FILTER OUTPUT  # Flush all rules only from the OUTPUT chain
```

Block all connections from a single IP address

Accomplishing this with iptables is fairly straight forward. The command is "iptables", followed by "-t filter" to specify the filter table. Because the filter table is the default table, this is optional and doesn't have to be included in the command unless specifying a table other than the filter table. It is good practice to use this to add to the readability of the command and to ensure consistency. The "-A INPUT" specifies to append this rule to the INPUT chain. The "-s 10.10.10.10" is the source IP address that must match for this rule to be evaluated true and then "-j DROP" is the action taken if the rule is evaluated true. If the default policy is not changed from accept to drop, then all packets not matching this rule will be allowed.

```
# iptables -t filter -A INPUT -s 10.10.10.10 -j DROP
```

Achieving this with nftables is quite simple once the table and chain are created. Start with the command "nft", followed by "add rule" to add a rule followed by "ip FILTER INPUT" which specified the address family, the table name, and the chain name. This is the followed by

the match (See Table 5: Common Matches for a subset of common matches). For this example the goal is the match the source IP address, so the match is "ip" followed by "saddr". This is then followed by "10.10.10.10" as the source IP address of 10.10.10.10 must match in order for this rule to evaluate true. The last item on the line is the action to take if this rule evaluates true and that is "drop" to drop the packet. Since nftables doesn't have the ability to change the policy, all other packets not matching this rule will be allowed.

```
# nft add rule ip FILTER INPUT ip saddr 10.10.10.10 drop
```

When considering that a user must create the tables and chains before entering rules, the nftables example makes iptables look much more appealing, but the tables and chains only have to be created one time. When researching nftables online, a considerable amount of the documentation doesn't discuss the creation of the tables and chains, but it is a requirement as no rule can be added until the base framework of tables and chains is first established.

Block all connections from a subnetwork

With iptables this is identical to blocking a single IP address with the exception of using the CIDR notation or dotted octed notation with the IP address to indicate how many bits in the address must match. Dotted octet notation is also be used with the forward slash "/" as a separator.

```
# iptables -t filter -A INPUT -s 10.10.10.0/24 -j DROP
        or
# iptables -t filter -A INPUT -s 10.10.10.0/255.255.255.0 -j DROP
```

This will be identical to the configuration with nftables that blocks a single IP address with the exception of specifying the CIDR notation to indicate how many bits in the network

address must match. Unlike iptables, nftables does not have the ability to use dotted octet notation for masks.

```
# nft add rule ip FILTER INPUT ip saddr 10.10.10.0/24 drop
```

Block all connections from a subnetwork to a port

When using iptables this is identical to previous example with the addition of adding a destination port with the "--dport" option.

```
# iptables -t filter -A INPUT -p tcp --dport ssh -s 10.10.10.0/24 -j DROP
```

For this example a negative chain priority will be used to demonstrate some of the quirks that have to performed in certain instances. When entering commands from the command line, the nftables command "nft" will attempt to interpret the negative priority as a command line switch as it begins with a dash "-". In order for nftables to understand that this is a negative priority, a double dash "--" surrounded by spaces must precede the negative priority value to signify the end of command line options. This example doesn't assume that the table and chains have been created as command line vs interactive mode will be compared:

```
# nft add table ip FILTER
# nft add chain ip FILTER INPUT { type filter hook input priority -- -1; }
# nft add rule ip FILTER INPUT tcp dport ssh ip saddr 10.10.10.0/24 drop
```

Interactive mode simplifies the process as the command "nft" is only entered at the beginning with the "-i" switch to signify interactive mode. After that the commands are entered without having to repeatedly type the "nft" command . It makes entering negative chain priorities simpler as a double dash "--" is not needed , and, the backslash is not required to escape the semicolon as the bash shell no longer parses the command line. When done, the "quit" command is used to exit from interactive mode.

```
# nft -i
nft> add table ip FILTER
nft> add chain ip FILTER INPUT { type filter hook input priority -1; }
nft> add rule ip FILTER INPUT tcp dport ssh ip saddr 10.10.10.0/24 drop
nft> quit
```

Port numbers or port names listed in /etc/services can be used with both iptables and nftables commands so it is personal preference on how they are entered. Resolvable port names are shown by default, but can be displayed by number with optional command line arguments which will be discussed later.

Allow only packets from new or existing connections to be allowed into the system

Since this example states that only internally originated connections are to be allowed back into the system, it implies that all other inbound traffic should be dropped. With iptables, in this case, it makes sense to change the default policy, with the "-P" switch, to drop on the input chain. It doesn't hurt to explicitly set the default policy to accept on the output chain even though it is the default. To evaluate traffic based on the state, the "-m state" loads the xt_conntrack kernel module and dependent modules so that the system performs stateful inspection of the traffic. Using "-m state" requires another "--state' option" which is a list of states to match. Valid states for iptables is NEW for new sessions, ESTABLISHED for existing connections, RELATED which is traffic that is related to but does not belong to the existing connection, and INVALID for traffic that could not be identified. Since this example specifies to only allow return traffic from internally established connections, then the states will be ESTABLISHED and RELATED. Iptables requires multiple states to be separated by commas.

```
# iptables -t filter -P INPUT DROP
# iptables -t filter -P OUTPUT ACCEPT
# iptables -t filter -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Nftables starts to show its value when it comes to stateful firewall rules. As the previous example interactive mode is used to save on typing. The table and chain are created followed by

the rule. The match for nftables is "ct state" which states connection tracking, "ct", based on state of the traffic which is then followed by the state of the traffic. For nftables, the state names are exactly the same as iptables except they are specified in lower case. Although not required, if it is desired to count packets and bytes, it must be specified with the target "counter". In this example the rule will count the number of packets and bytes that are accepted by this rule. Because nftables doesn't have the ability to change the default policy of accept, then a rule must be added to drop all other traffic.

```
# nft -i
nft> add rule ip FILTER INPUT ct state established,related counter accept
nft> add rule ip FILTER INPUT drop
nft> quit
```

The output of configured tables/chains/rules between iptables and nftables is considerably different.

```
# iptables -L
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere            state RELATED,ESTABLISHED

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

```
# nft list chain ip FILTER INPUT
table ip FILTER {
        chain INPUT {
                type filter hook input priority -1;
                ct state established,related counter packets 0 bytes 0 accept
                drop
        }
```

```
}
```

Saving and importing rules

Nftables produces a clean, organized output. With nftables it would be simpler to take this command output, and reproduce the commands required to produce the same output, but isn't required since it is possible to save the configuration and import it in later. With iptables the commands to save and restore are iptables-save and iptables-restore respectively. Nftables uses the same nft command to export and import:

```
# nft list table ip FILTER > filtertable.fw      # Redirect output to a file
# nft -f filtertable.fw                          # Use the -f switch to read in file
```

Specifying interfaces

There are many services running on systems that must communicate via the loopback interface so it is important to allow this. There are different options depending on the direction of the traffic with respect to the interface. With iptables, the -i option is used to specify “in” or reading from the interface, and the -o option is used to specify “out” or writing to the interface:

```
# iptables -A INPUT -i lo -j ACCEPT
# iptables -A OUTPUT -o lo -j ACCEPT
```

Nftables similarly uses different options depending on the direction. If reading from the interface use iif and if writing to the interface use oif. Assuming the INPUT and OUTPUT chains are created, the nftables commands are:

```
# nft add rule ip FILTER INPUT iif lo accept
# nft add rule ip FILTER OUTPUT oof lo accept
```

These command are not limited to just using the loopback interface as any available interface name, found via the ifconfig command, can be specified.

Kenton A. Groombridge, kgroombr@comcast.net

Specifying ranges and multiple IP address and ports

If individual rules had to be written by specifying single source and destination addresses and ports, then it would take a considerable amount of time to write rules and the length of the rule list would be grow quite large very quickly; therefore, it is common to define ranges and/or multiple discontiguous IP addresses and ports. Not only does it save keystrokes, but it adds to the readability of the rules.

Without the specified kernel modules, iptables has the ability to comma separate IP addresses when entering rules. It will create as many rules as required for every possible combination of communications. For example, this command:

```
# iptables -t filter -A INPUT -s 10.1.1.1.1,10.1.1.2 -d 10.0.0.1,10.0.0.2 -j ACCEPT
```

Will create these four rules:

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  10.1.1.1            10.0.0.1
ACCEPT     all  --  10.1.1.1            10.0.0.2
ACCEPT     all  --  10.1.1.2            10.0.0.1
ACCEPT     all  --  10.1.1.2            10.0.0.2
```

One of the main goals of nftables was to add to the readability of the rules, so allowing ranges and discontiguous addresses and ports were incorporated early on and all that needs to be done is to enclose them in braces. Assuming the chain "FILTER INPUT" has been created, the previous iptables command can be accomplished with nftables with the following:

```
# nft add table ip FILTER
# nft add chain ip FILTER INPUT { type filter hook input priority 1 \; }
# nft add rule ip FILTER INPUT ip saddr { 10.1.1.1,10.1.1.2 } \
        ip daddr { 10.0.0.1,10.0.0.2 } accept
```

Because of the length of this command, it has been broken up into two parts. The first line is terminated with a backslash "\". In UNIX, this called a command continuation as the command isn't completed until a newline (return is pressed) is encountered. This is often used in scripts and documentation for readability. The command may have been entered on a single line without the backslash.

```
# nft list table ip FILTER
table ip FILTER {
      chain INPUT {
            type filter hook input priority 1;
            ip saddr { 10.1.1.1, 10.1.1.2} ip daddr { 10.0.0.1, 10.0.0.2} accept
```

Iptables provides the ability to specify ranges and discontiguous addresses and ports; however, the proper kernel components must be installed.

These iptables and nftables commands will allow destination traffic to TCP ports 22, 80, and 443 into the system:

```
# iptables -t filter -A INPUT -p tcp --match multiport --dports 22,80,443 -j ACCEPT
```

```
# nft add rule ip FILTER INPUT tcp dport { 22,80,442 } accept
```

This iptables command will allow destination traffic to TCP ports 20 through 80 into the system (note that iptables uses a colon ":" as the range separator and nftables uses a dash "-"):

```
# iptables -t  filter -A INPUT -p tcp --match multiport --dports 20:80 -j ACCEPT

#  nft add rule ip FILTER INPUT tcp dport { 22-80 } accept
```

These iptables and nftables commands will allow outgoing traffic to destination IP addresses 10.1.1.1 through 10.1.1.20:

```
# iptables -t filter -A OUTPUT -m iprange --dst-range '10.1.1.1-10.1.1.20' -j ACCEPT

#  nft add rule ip FILTER OUTPUT ip daddr { 10.1.1.1-10.1.1.20 } accept
```

These commands are not limited to just one option as combinations of source and destination IP addresses and ports can be used to get the desired effect.

5. Adding Comments to Rules

Just like programmers comment code so that it can be understood by others, firewall rule lists should also be commented especially if there are multiple people that manage the rules. Even with the most simple of rule lists, it is highly recommended to comment each rule, not so much of what the rule does, but why it is there.

Throughout this document comments have been added to commands to explain their function using the format "# and some text". The commands will execute properly even if "# and some text" is appended to the command because the shell will not process it because it understands that it as a shell comment. The problem with this is that it is recognized by the shell as a comment, but not by iptables and nftables; thus, the comment isn't saved with the rules.

Iptables allows for attaching comments to rules as long as the appropriate module is built and installed. Like most iptables rules, using comments is a bit convoluted:

```
# iptables -A INPUT -p tcp --dport 22 -j DROP -m comment --comment "disallow ssh"
```

The “-m comment” tells iptables to use the comment module, and “--comment “disallow ssh” is the actual command that adds a comment to the rule. The comment must be entered within double quotes. Comments are displayed with normal output when listing output with the “-L” switch:

```
# iptables -L INPUT
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       tcp  --  anywhere             anywhere            tcp dpt:ssh /* disallow ssh */
```

The current nftables documentation doesn't appear to have any information regarding adding comments, but digging into the source code of nftables v0.4 reveals code supporting the addition of comments which can be added as follows:

```
# nft add rule FILTER INPUT tcp dport ssh drop comment \"disallow ssh\"
```

This isn't quite as convoluted as the iptables version. The quotes must be escaped when entering commands from the shell and not escaped if entering commands in interactive mode. The comments entered are displayed when listing output:

```
# nft list table FILTER
table ip FILTER {
        chain INPUT {
                tcp dport ssh drop comment "disallow ssh"
```

6. Logging

Many would call logging the most important thing to do when configuring a firewall which is difficult to argue; however, rules by themselves server little purpose until put together in a working rule list. Each rule is a piece of the puzzle, and unless the puzzle is complete, there is an exposed hole.

Basic Logging

Iptables includes a LOG target that can be used to log packets. What differentiates the LOG target from others targets is that it returns control back to the calling rule list for further processing. Essentially it acts like a subroutine in a program. There are a couple approaches to logging packets. One way would be to create two identical rules with the exception of the targets. The first rule performs the logging action, and the next rule jumps to the appropriate action (ACCEPT or DROP).

```
# iptables -t filter -A OUTPUT -d 10.0.0.1 -j LOG    # Log, continue to next rule
# iptables -t filter -A OUTPUT -d 10.0.0.1 -j DROP # Drops the same packet
```

The disadvantage to this is that it causes the packet to be evaluated twice, once to log, and once to perform the action. Evaluating rules takes time and processing power, and keeping the rule list as short as possible ensures efficiency.

The second approach is to create a user-defined chain specifically for logging that logs the packet and performs the appropriate action (ACCEPT or DROP).

```
# iptables -t filter -N LOGGING                          # Create non-base chain for logging
# iptables -t filter -A LOGGING -j LOG                   # Add rule to logging chain to LOG
# iptables -t filter -A LOGGING -j DROP     # Add rule to logging chain to DROP
# iptables -t filter -A OUTPUT -d 10.0.0.1 -j LOGGING  # Jump LOGGING if match
```

With this version, the packet is evaluated once on the base chain OUTPUT, and if the destination address is 10.0.0.1, then it jumps to the non-base chain LOGGING which first logs the packet, and second drops the packet. Utilizing this method, although a bit more convoluted, is more efficient than the previous version that had to evaluate the packet twice.

One major advantage to using nftables is the ability to have multiple targets per rule. The same rule written with nftables is much simpler to implement:

```
# nft add rule ip FILTER OUTPUT ip daddr 10.0.0.1 log drop
```

Logging with custom output

Log output can be cryptic, and the log output is often modified to add keywords to make the log output easier to read. Doing this with iptable and nftables is straight forward. With iptables, the non-base chain LOGGING will need to be created, and the matching packet will use the jump chain of LOGGING:

```
# iptables -t filter -A LOGGING -j LOG --log-prefix "Packet Drop: "
# iptables -t filter -A LOGGING -j DROP
```

As before, nftables allows the entire configuration in one command. Like entering comments with nftables, the quotes around the custom message must be escaped, unless utilizing interactive mode:

```
# nft add rule ip FILTER OUTPUT ip daddr 10.0.0.1 log prefix \“Packet Drop: \“ drop
```

Logging with rate limiting

Utilizing rate limiting gives the ability to create a threshold to only log a stipulated amount of matching packets within a specified time frame. This is important since it prevents

log files from growing at a rapid rate which produces log pollution, and denial of service (DoS) conditions where the system is logging at such a rate that it can't perform other tasks or, if the logging directory is part of root file system, it exhausts all available space on root which causes the system to freeze or crash.

With iptables, utilizing the user-defined chain LOGGING, is fairly simple to implement. A logging rule can be created to log only up to two events per minute. The "-m limit" tells iptables to use the nft_limit module and the "--limit 2/m" will allow up to two events per minute.

```
# iptables -t filter -A LOGGING -m limit --limit 2/m -j LOG
```

The nftables version is similar except is reads a bit easier. The "limit rate" tells nftables that what follows is a number/timeframe of how much to limit. The timeframe must be spelled out with nftables as in "2/minute". Here is an example of the same command added to a non-base chain named LOGGING:

```
# nft add rule ip FILTER LOGGING limit rate 2/minute accept
```

7. Editing rules

Editing rules has its challenges, but unfortunately the method that nftables utilizes to edit rules is a step back from that of iptables; however, it is only slightly more complicated than that of iptables.

Rather than focus on rule writing, a simple input filter will be used that has three rules each allowing a single destination port of 22, 23, and 25. They are entered in that order so the effects of the editing can be more easily seen. Each example will build upon the previous output.

Listing line numbers and rule handles

In order to edit the rules, the position where to insert, delete or replace has to be specified. Iptables utilizes line numbers which can be displayed with the --line-numbers option accompanied with the list option "-L".

```
# iptables -t filter -L -n --line-numbers
Chain INPUT (policy ACCEPT)
num   target     prot opt source               destination
1     ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:22
2     ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:23
3     ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:25
```

In the iptables output the first column "num" is the line number for the rule. A new switch "-n" was added to not resolve addresses or port names.

The nftables command uses the "-a" switch to output what it calls the "rule handle". Although it looks like line numbering, it isn't, and how it operates will be described a bit later.

```
# nft list table FILTER -nna
table ip FILTER {
        chain INPUT {
                type filter hook input priority 0;
                tcp dport 22 accept # handle 2
                tcp dport 23 accept # handle 3
                tcp dport 25 accept # handle 4
        }
}
```

This nftables command uses the "-n" and "-nn" switch like tcpdump. "-n" disables address resolution, and "-nn" disables address and port resolution. There is no column for the rule handle, but at the end of each rule is listed "# handle num" where "num" is the rule handle. With nftables version 0.4, if switches are to be used, they must be used in command line mode. Unfortunately, attempting to use a switch in interactive mode will throw an error. Notice in the

output that the handle number starts with 2 (do not get "priority 0" confused with a rule handle of the first rule).

Deleting rules

The switch to delete lines with iptables is "-D". In order to delete line number 2 with iptables, the command will be:

```
# iptables -t filter -D INPUT 2
# iptables -t filter -L -n --line-numbers
Chain INPUT (policy ACCEPT)
num   target     prot opt source               destination
1     ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:22
2     ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:25
```

One thing to notice with the output following the deletion of line number 2, is that iptables reordered the rules, so what was once line number 3 is now line number 2. In fact, iptables keeps the line numbers sequential.

In order to delete rules in nftables, the command option is "delete" followed by the table and chain name, then followed by the keyword "handle" then the rule handle.

```
# nft delete rule FILTER INPUT handle 3
# nft list table FILTER -nna
table ip FILTER {
        chain INPUT {
                type filter hook input priority 0;
                tcp dport 22 accept # handle 2
                tcp dport 25 accept # handle 4
        }
}
```

Unlike iptables, there is no renumbering of rules. The way nftables works, is once a rule is given a handle, it keeps that handle indefinitely until something forces it to change such as deleting the tables and chains, reentering them manually or reading them from a file.

Inserting rules

To insert a rule with iptables the command switch is "-I" and specify the line number where the new rule should be placed. If a line number isn't specified, the rule will be inserted at the beginning of the list. To place a rule between line numbers one and two, then specify line two, and what was once line two will become line three:

```
# iptables -t filter -I INPUT 2 -p tcp --dport 23 -j ACCEPT
# iptables -t filter -L -n --line-numbers
Chain INPUT (policy ACCEPT)
num   target     prot opt source               destination
1     ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:22
2     ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:23
3     ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:25
```

Using nftables previous output where rule handle three was deleted, to insert the same rule between two and four, a person may think that the rule handle to specify would be rule handle three as that is between two and four. Not exactly. Like iptables, the rule handle where the rule is to be inserted is specified. If a non-existent rule handle is specified, nftables with throw an error. Like iptables, if the rule handle position is omitted, the rule will be place at the beginning of the rule list. Unlike iptables where the syntax of the command is consistent, nftables now uses a new keyword "position" followed by the rule handle where to insert the rule:

```
# nft insert rule FILTER INPUT position 4 tcp dport 23 accept
# nft list table FILTER -nna
table ip FILTER {
        chain INPUT {
                type filter hook input priority 0;
                tcp dport 22 accept # handle 2
                tcp dport 23 accept # handle 5
```

```
            tcp dport 25 accept # handle 4
        }
}
```

This example of nftables confirms that statement that rule handles never change. Do not get rule handles confused with how nftables traverses the rule list. The rules are still evaluated sequentially from top to bottom. Rule handles are an nftables mechanism to uniquely identify rules and not an assignment of order.

Replacing rules

With iptables, the "-R" switch will replace a rule. The line number for the rule to be replaced is specified followed by the superseded rule:

```
# iptables -t filter -R INPUT 3 -p tcp --dport 80 -j ACCEPT
# iptables -t filter -L -n --line-numbers
Chain INPUT (policy ACCEPT)
num   target     prot opt source               destination
1     ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:22
2     ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:23
3     ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0           tcp dpt:80
```

Nftables doesn't currently include the ability to replace rules and it is unknown if this functionality will be added. In order to perform the same function as replacement, the rule to be replaced can be deleted and the rule to replace can be inserted into the rule chain in the same location where the previous rule was deleted.

8. Conclusion

There is little doubt that the iptables components in the Linux kernel will remain for many more years, and whether it is to be replaced by nftables is yet to be determined. For those versed in iptables, there is no immediate reason to learn nftables, but for the Linux newcomer, nftables should be considered as it appears to have an easier learning curve. As nftables matures

and features are added, it will become more appealing and may eventually overtake iptables as the favored Linux firewall.

Kenton A. Groombridge, kgroombr@comcast.net

9. References

nftables HOWTO (2014, November 19). Retrieved April 27, 2015, from http://wiki.nftables.org/wiki-nftables/index.php/Main_Page

Russell, R. (2002, January 24). Retrieved Retrieved March 14, 2015, from http://www.netfilter.org/documentation/index.html#documentation-howto

Nftables - Gentoo Wiki. (2015, January 28). Retrieved March 14, 2015, from http://wiki.gentoo.org/wiki/Nftables

Kenton A. Groombridge, kgroombr@comcast.net

10. Appendix

Description	iptables	nftables
Add (append)	-A	add
Delete	-D	delete
Flush	-F	flush
Insert	-I	insert
List	-L	list
Policy	-P	N/A
Replace	-R	N/A

Table 1: Command Options

Description	iptables	nftables
Protocol	-p	N/A
Source Address	-s	saddr
Destination Address	-d	daddr
Source Port	--sport	sport
Destination Port	--dport	dport
Jump Target	-j	N/A
Match Name	-m	N/A

Table 2: Parameters

Kenton A. Groombridge, kgroombr@comcast.net

Description	iptables	nftables
Accept (Allow) Packet	ACCEPT	accept
Silently Drop Packet	DROP	drop
Reject Packet & Respond	REJECT	reject
Destination NAT	DNAT	dnat
Source NAT	SNAT	snat
Masquerade NAT	MASQUERADE	masquerade
Log Packet	LOG	log
Count packets and bytes	N/A	counter

Table 3: Target Options

Match	Arguments	Description
ip	saddr	Source address
	daddr	Destination address
tcp	sport	Source port
	dport	Destination port
udp	sport	Source port
	dport	Destination port
ct	state	State of the connection

Table 4: Common Matches[1]

1 Nftables - Gentoo Wiki. (n.d.). Retrieved March 15, 2015, from http://wiki.gentoo.org/wiki/Nftables#Matches

Kenton A. Groombridge, kgroombr@comcast.net

Priority Name/Value	Function
NF_IP_PRI_CONNTRACK_DEFRAG (-400)	Priority of defragmentation
NF_IP_PRI_RAW (-300)	Traditional priority of the raw table placed before connection tracking operation
NF_IP_PRI_SELINUX_FIRST (-225):	SELinux operations
NF_IP_PRI_CONNTRACK (-200)	Connection tracking operations
NF_IP_PRI_MANGLE (-150)	Mangle operation
NF_IP_PRI_NAT_DST (-100)	Destination NAT (DNAT)
NF_IP_PRI_FILTER (0)	Filtering operation, the filter table
NF_IP_PRI_SECURITY (50)	Place of security table where secmark can be set for example
NF_IP_PRI_NAT_SRC (100)	Source NAT (SNAT)
NF_IP_PRI_SELINUX_LAST (225)	SELinux at packet exit
NF_IP_PRI_CONNTRACK_HELPER (300)	Connection tracking at exit

Table 5: Chain Priorities[2]

2 Configuring chains - nftables HOWTO. (n.d.). Retrieved from http://wiki.nftables.org/wiki-nftables/index.php/Configuring_chains#Base_chain_types

Last Updated: July 21st, 2017

Upcoming Training

SANSFIRE 2017	Washington, DC	Jul 22, 2017 - Jul 29, 2017	Live Event
SANS Prague 2017	Prague, Czech Republic	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Salt Lake City 2017	Salt Lake City, UT	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS New York City 2017	New York City, NY	Aug 14, 2017 - Aug 19, 2017	Live Event
Community SANS Omaha SEC401*	Omaha, NE	Aug 14, 2017 - Aug 19, 2017	Community SANS
Community SANS Trenton SEC401	Trenton, NJ	Aug 21, 2017 - Aug 26, 2017	Community SANS
Virginia Beach 2017 - SEC401: Security Essentials Bootcamp Style	Virginia Beach, VA	Aug 21, 2017 - Aug 26, 2017	vLive
SANS Virginia Beach 2017	Virginia Beach, VA	Aug 21, 2017 - Sep 01, 2017	Live Event
SANS Chicago 2017	Chicago, IL	Aug 21, 2017 - Aug 26, 2017	Live Event
SANS Adelaide 2017	Adelaide, Australia	Aug 21, 2017 - Aug 26, 2017	Live Event
Community SANS Pasadena SEC401 @ NASA	Pasadena, CA	Aug 23, 2017 - Aug 30, 2017	Community SANS
Mentor Session - SEC401	Minneapolis, MN	Aug 29, 2017 - Oct 10, 2017	Mentor
SANS San Francisco Fall 2017	San Francisco, CA	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS Tampa - Clearwater 2017	Clearwater, FL	Sep 05, 2017 - Sep 10, 2017	Live Event
Mentor Session - SEC401	Edmonton, AB	Sep 06, 2017 - Oct 18, 2017	Mentor
SANS Network Security 2017	Las Vegas, NV	Sep 10, 2017 - Sep 17, 2017	Live Event
Mentor Session - SEC401	Ventura, CA	Sep 11, 2017 - Oct 12, 2017	Mentor
Community SANS Albany SEC401	Albany, NY	Sep 11, 2017 - Sep 16, 2017	Community SANS
Community SANS Columbia SEC401	Columbia, MD	Sep 18, 2017 - Sep 23, 2017	Community SANS
Community SANS Dallas SEC401	Dallas, TX	Sep 18, 2017 - Sep 23, 2017	Community SANS
SANS London September 2017	London, United Kingdom	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Baltimore Fall 2017	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Copenhagen 2017	Copenhagen, Denmark	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS Boise SEC401	Boise, ID	Sep 25, 2017 - Sep 30, 2017	Community SANS
Baltimore Fall 2017 - SEC401: Security Essentials Bootcamp Style	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	vLive
Community SANS New York SEC401	New York, NY	Sep 25, 2017 - Sep 30, 2017	Community SANS
Rocky Mountain Fall 2017	Denver, CO	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS Sacramento SEC401	Sacramento, CA	Oct 02, 2017 - Oct 07, 2017	Community SANS
SANS DFIR Prague 2017	Prague, Czech Republic	Oct 02, 2017 - Oct 08, 2017	Live Event
Community SANS Charleston SEC401	Charleston, SC	Oct 02, 2017 - Oct 07, 2017	Community SANS