THE LINUX FOUNDATION OPEN SOURCE SUMMIT NORTH AMERICA | Embedded Linux Conference North America

Introduction to NFTables

Mike Anderson
Director of Technology
The PTR Group, LLC

THE LINUX FOUNDATION

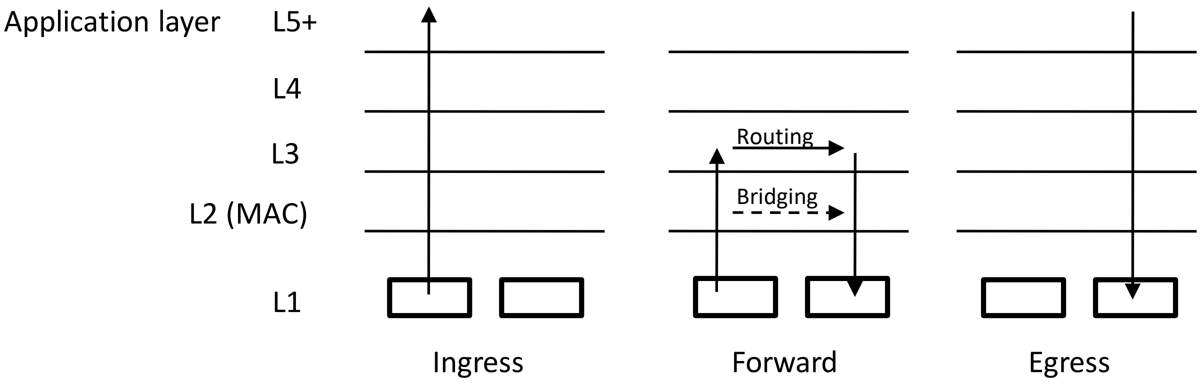
Speaker/Author Details

- **Website:**
  - **http://www.theptrgroup.com**
- **Email:**
  - **mailto:mike@theptrgroup.com**
- **Linked-in:**
  - **https://www.linkedin.com/in/mikeandersonptr**
- **Twitter:**
  - **@hungjar**
- **PTR is now a subsidiary of** Huntington Ingalls Industries

What We Will Talk About…

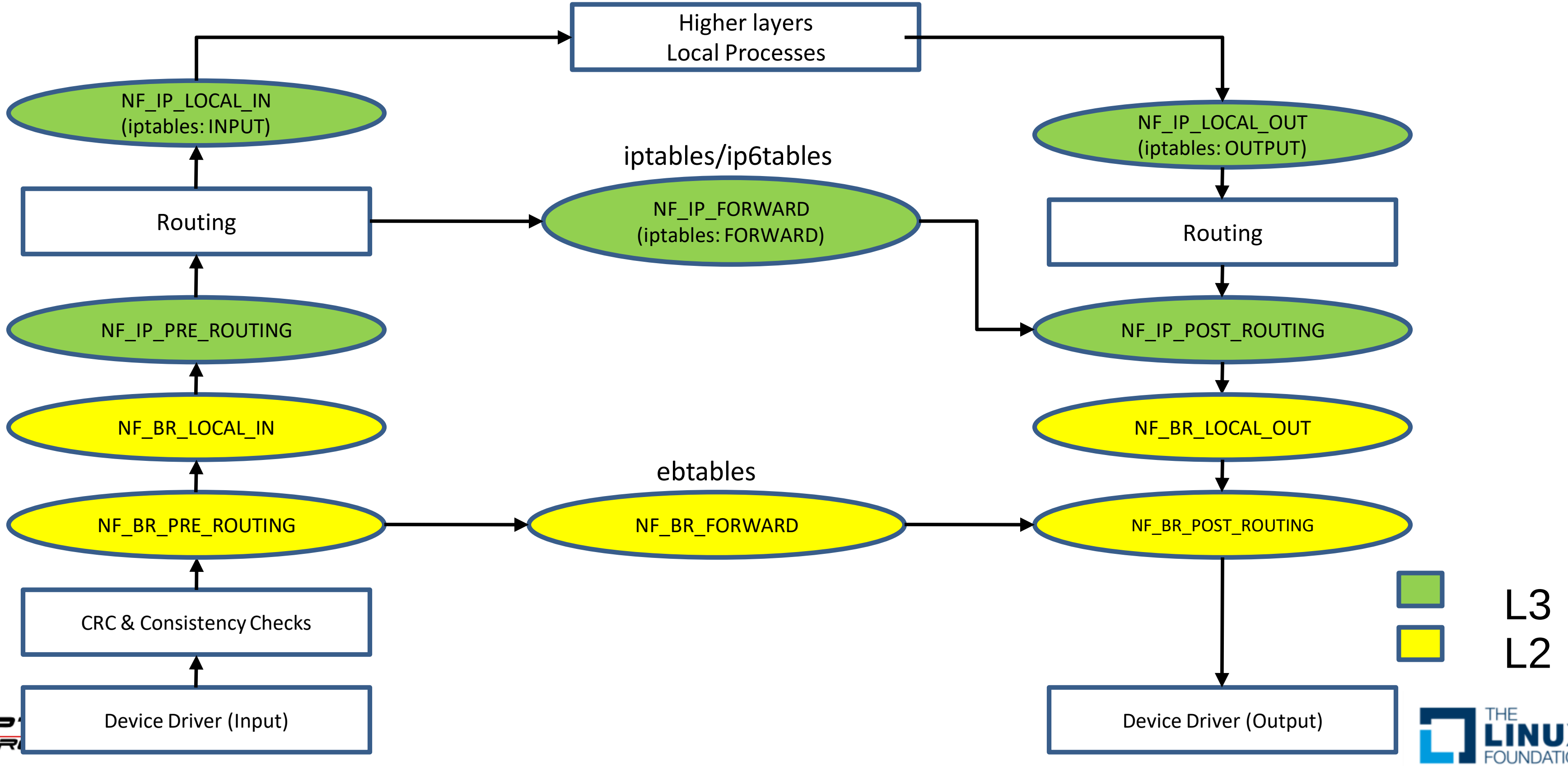
- Linux packet handling hooks in the stack
- Stateless and stateful firewalls
- Connection tracking
- Xtables packet flow
- nftables architecture
- Installation in kernel and userspace
- Approach and usage
- Converting existing Xtables firewalls to nftables
- Finding documentation
- Summary

Understanding Network Traffic Terms



- The paths taken in the stack by network frames differ for the inbound (ingress), routed/bridged and outbound (egress) data streams
- There are multiple decision points along the way that determine the fate and path of network frames
- These include the potential of simply dropping the packet at several decision points

Hooks Within the Stack



Example Hook Module

- Here is an example of hooking in the stack to simply drop all UDP traffic

```
// 'Hello World' v2 netfilter hooks example that drops UDP (protocol 17)
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
#include <linux/skbuff.h>
#include <linux/udp.h>
#include <linux/ip.h>

static struct nf_hook_ops nfho;   //net filter hook option struct
struct sk_buff *sock_buff;
struct udphdr *udp_header;         //udp header struct (not used)
struct iphdr *ip_header;           //ip header struct

int init_module() {
        nfho.hook = hook_func;
        nfho.hooknum = NF_IP_PRE_ROUTING;
        nfho.pf = PF_INET;
        nfho.priority = NF_IP_PRI_FIRST;
        nf_register_hook(&nfho);

        return 0;
}
```

(source http://www.paulkiddie.com)

Example Hook Module #2

```
unsigned int hook_func(unsigned int hooknum, struct sk_buff **skb,
                    const struct net_device *in,
                    const struct net_device *out,
                    int (*okfn)(struct sk_buff *)) {
    sock_buff = *skb;
     // grab network header using accessor
    ip_header = (struct iphdr *)skb_network_header(sock_buff);

    if(!sock_buff) { return NF_ACCEPT;}
    // grab transport header
    if (ip_header->protocol==17) {
        udp_header = (struct udphdr *)skb_transport_header(sock_buff);
        // log we've got udp packet to /var/log/messages
        printk(KERN_INFO "got udp packet \n");
        return NF_DROP;
    }
        return NF_ACCEPT;
}

void cleanup_module() {
        nf_unregister_hook(&nfho);
}
```

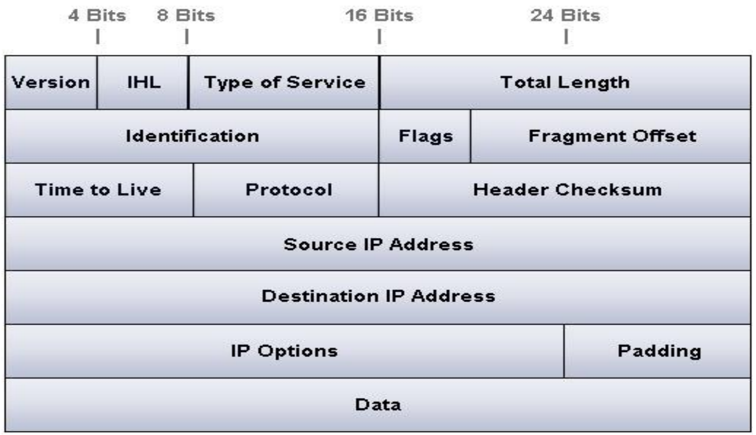
Network Packet Filtering

Source: youtube.com

- Within each operating system with network connectivity, we must take into account the evil cyber wonks
  - They have nothing better to do with their time but attack the unsuspecting
- Generally, the first layer of a defense-in-depth strategy is to try to block evil-doers with a packet filter
  - Drops/blocks packets before they can enter the network stack
  - Packet filters are typically stateless or stateful
    - Stateless filters are also known as network-layer firewalls
    - Stateful filters are also known as circuit-level firewalls

Packet Filters

- Validates a packet based mostly on the contents of its IP header
  - IP Addresses
  - Protocol
  - Type of service
  - Hardware Interface
  - Direction
- Each packet is an entity onto themselves
  - I.e., the filter is stateless
- Little visibility into the packet payload/data
- Packet filters *can* look into the TCP header
  - Typically only for the TCP/UDP port, however



Source: learn-networking.com

Well Known Port Examples

- A Linux system has 65535 ports
  - Ports are bound using the `bind()` call for servers or automatically for clients
- Not all are used
- Modification of well-known ports (1-1023) requires superuser permissions
  - Reserved for privileged system processes

Protocol	Port	Protocol	Port
FTP	20, 21	NNTP	119
SSH, SFTP, SCP	22	IMAP	143
Telnet	23	SNMP	161
SMTP	25	LDAP	389
TACACS	49	ISAMP (VPN)	500
DNS	53	Syslog	514
TFTP	69	LDAP/TLS	636
HTTP	80	L2TP	1701
Kerberos	88	PPTP	1723
POP3	110	Remote access	3389

Source: certapps.com

Other Ports

- Registered Ports (1024 -> 49151) are for applications that do not need superuser privileges
  - OpenVPN:1194
  - NFS: 2049
  - SVN: 3690
- Ephemeral Ports (49152 -> 65535) are for applications that need a temporary communications port
- Many Linux kernels use the port range (32768 -> 61000) for sockets

Problems With Port Blocking

- FTP example
  - Client sends command from an arbitrary port to port 21 on the server
  - Server sends data from port 20 to the client on a dynamically allocated port
  - Depending on the firewall configuration (usually default deny) the dynamically-allocated port is probably blocked

Example Packet Filtering Rules

- Packet filter behavior is defined by the use of rules

Policy/Rule	Firewall Setting
No outside Web access	Drop all outgoing packets to any IP address using port 80
Prevent IPTV from eating up the available bandwidth	Drop all incoming UDP packets except DNS and router broadcasts
Prevent your network from being used for a DoS attack	Drop all ICMP packets going to a 'broadcast' address (e.g. 222.22.255.255)
Prevent your network from being tracerouted	Drop all outgoing ICMP

Source: cis.poly.edu

Packet Filter Rule Definitions

- Rules are processed from top to bottom until match is found
- If no rule matches, the default policy is followed

Action	Source Address	Dest Address	Protocol	Source Port	Dest Port
Deny	222.22/16	outside of 222.22/16	TCP	> 1023	80
Allow	outside of 222.22/16	222.22/16	TCP	80	>1023
Allow	222.22/16	outside of 222.22/16	UDP	>1023	53
Allow	outside of 222.22/16	222.22/16	UDP	53	>1023
Deny	All	All	All	All	All

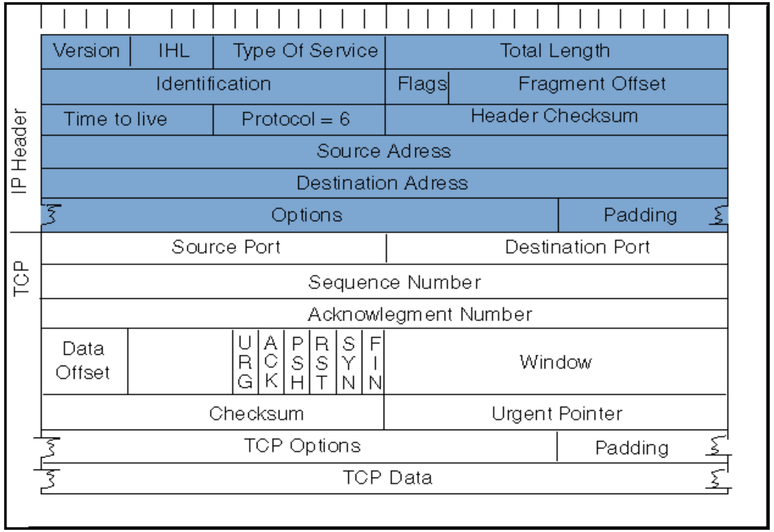
Source: cis.poly.edu

Stateless Filters: Pros & Cons

- Advantages
  - Effective against worms and Trojan horses
  - Can be very fast if filtering rules are not too complex
  - Built into Linux kernel
- Disadvantages
  - Does not protect against attacks with malformed packets (e.g., spoofing)
  - Does not protect against protocol-based attacks (e.g., buffer overflows)
  - Not effective against attacks using authorized channels (e.g., email viruses)
  - Cannot enforce some policies like excluding certain users
  - Rules can get complicated and difficult to test
- Necessary but not sufficient
  - Packet filters are the first line of defense in a multi-layered approach

Stateful Filters

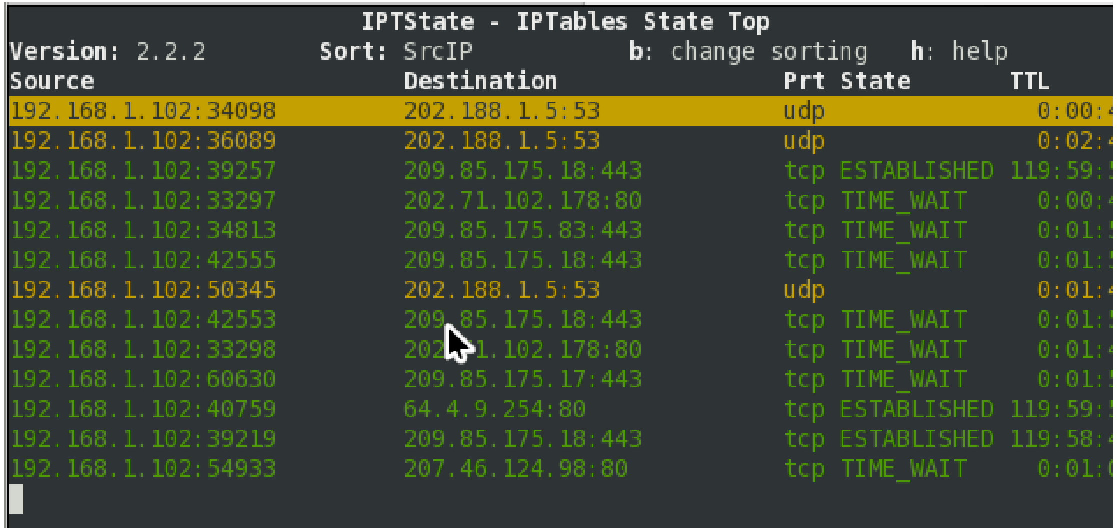
- Extension of basic packet filters
- Remembers the state of communication sessions
- Using TCP header information, it permits connections only from trusted clients
- Can configure to only allow in packets from established sessions



Source: cisco.com

Connection State Table

- Keeps a dynamic table of active sessions
- Assigns connection states to each packet
  - **`NEW`**, **`ESTABLISHED`**, **`RELATED`**, **`INVALID`**
- Linux uses the **`conntrack`** mechanism to maintain the connection table



IPTState - IPTables State Top

Version: 2.2.2   Sort: SrcIP   b: change sorting   h: help

Source	Destination	Prt	State	TTL
192.168.1.102:34098	202.188.1.5:53	udp		0:00:
192.168.1.102:36089	202.188.1.5:53	udp		0:02:
192.168.1.102:39257	209.85.175.18:443	tcp	ESTABLISHED	119:59:
192.168.1.102:33297	202.71.102.178:80	tcp	TIME_WAIT	0:00:
192.168.1.102:34813	209.85.175.83:443	tcp	TIME_WAIT	0:01:
192.168.1.102:42555	209.85.175.18:443	tcp	TIME_WAIT	0:01:
192.168.1.102:50345	202.188.1.5:53	udp		0:01:
192.168.1.102:42553	209.85.175.18:443	tcp	TIME_WAIT	0:01:
192.168.1.102:33298	202[illegible]1.102.178:80	tcp	TIME_WAIT	0:01:
192.168.1.102:60630	209.85.175.17:443	tcp	TIME_WAIT	0:01:
192.168.1.102:40759	64.4.9.254:80	tcp	ESTABLISHED	119:59:
192.168.1.102:39219	209.85.175.18:443	tcp	ESTABLISHED	119:58:
192.168.1.102:54933	207.46.124.98:80	tcp	TIME_WAIT	0:01:

Source: johnboy60.com

Stateful Filters: Pros & Cons

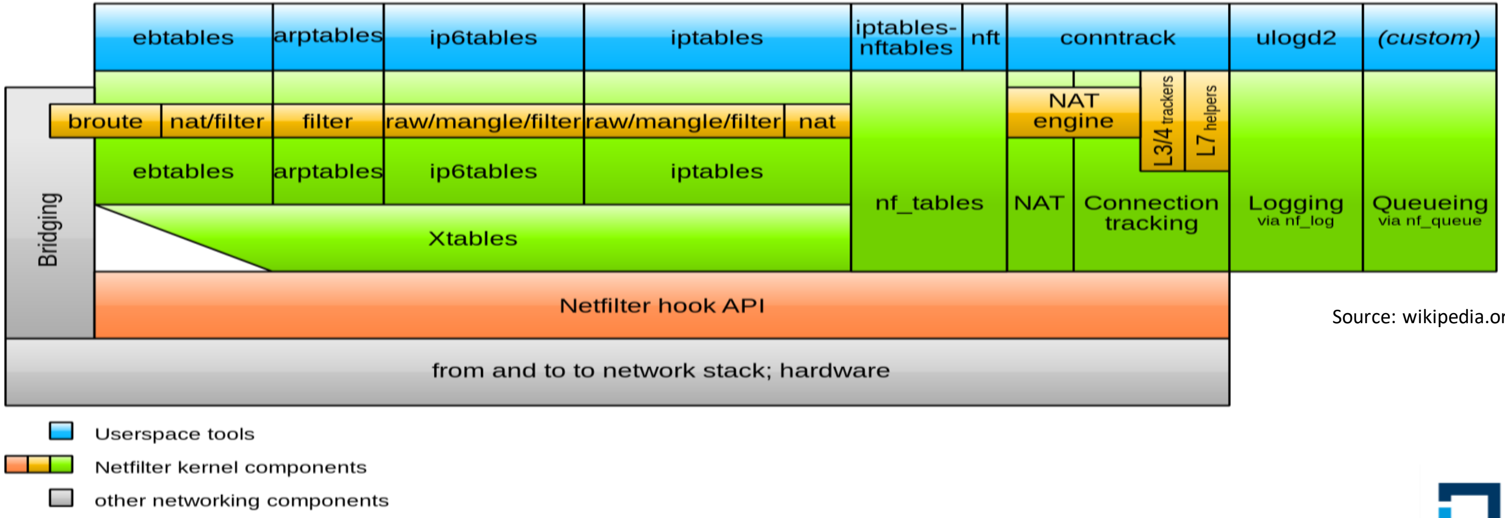
- Advantages
  - Relatively easy to implement
    - Standard protocols are very easy to configure
  - Among the fastest filter types
    - Needs to check packet rules for connection requests (OSI layer 7)
    - Otherwise only needs to check state table (OSI layer 4 and below)
  - Protects against 'answer' session exploits and some DoS like SYN-flooding
  - Built into Linux kernel
- Disadvantages
  - Does not protect against attacks with malformed packets (e.g., spoofing)
  - Does not protect against protocol-based attacks (e.g., buffer overflows)
  - Not effective against attacks using authorized channels (e.g., email viruses)

Bridging Firewalls and Brouters

- At L2, Linux has another set of packet filtering code known as ebtables
  - Three separate tables: filter, NAT and broute
- Often used for rewriting MAC addresses ala NAT
- In the broute table, we decide to either bridge or route the packet by forwarding it to L3
  - `DROP` rule says to route the frame to L3 and `ACCEPT` rule says to bridge the frame

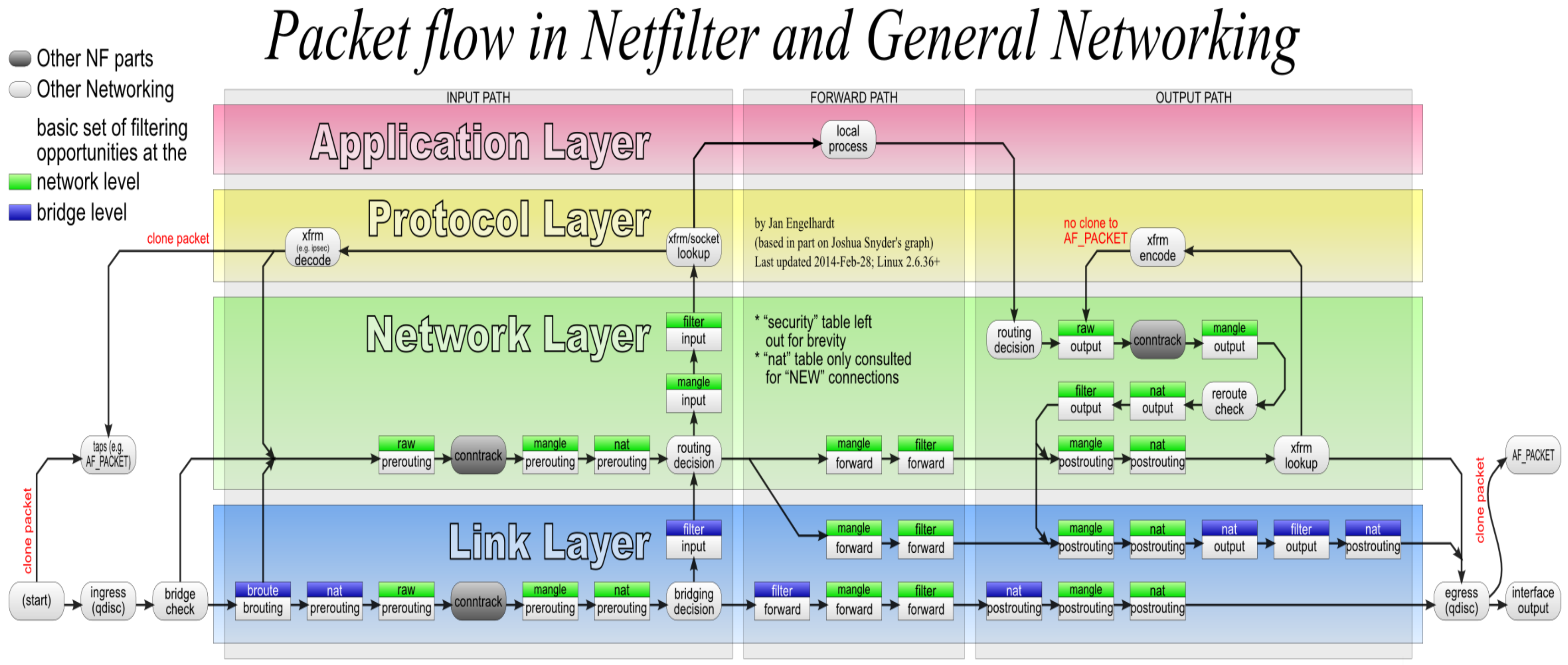
Netfilter Architecture

- ebtables: Manages ruleset for Ethernet packet frames
- arptables: Manages ruleset for ARP packet frames
- iptables/ip6tables: iptables for IPv4 and IPv6 respectively
- conntrack: Manage in-kernel connection state table



Source: wikipedia.org

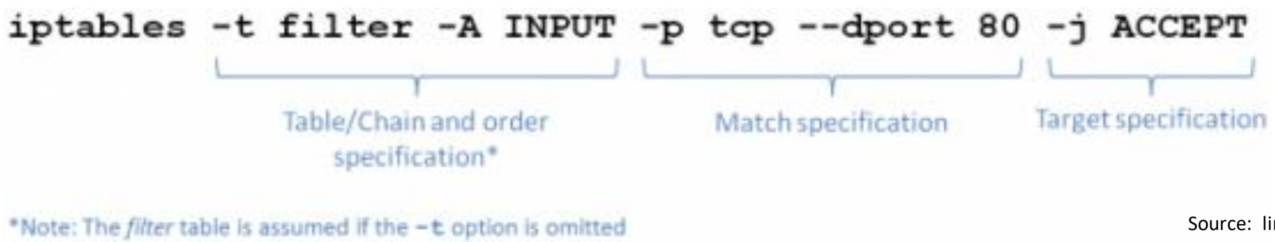
Xtables Packet Flow



Problems with Xtables

- The Xtables mechanism has been in use since the 2.4 kernel
- Defining both stateless and stateful firewall rules can be tedious due to the number of rules that need to be written
- The order of the rules is important
  - Any change to the rules require the entire table be reloaded
- When being used on a multi-tenant server, the creation and searching of the rules for each client starts to slow exponentially as the number of rules increases
- In addition, there is considerable duplicated code between the protocol stacks

Example of Dropping Malformed Packets



Source: linuxjournal.com

- Deal with oddball packets

```
$ iptables -A INPUT -i eth0 -p tcp -m tcp
 --tcp-flags FIN,SYN,RST,PSH,ACK,URG NONE -j DROP
$ iptables -A INPUT -i eth0 -p tcp -m tcp
 --tcp-flags FIN,SYN FIN,SYN -j DROP
$ iptables -A INPUT -i eth0 -p tcp -m tcp
 --tcp-flags SYN,RST SYN,RST -j DROP
$ iptables -A INPUT -i eth0 -p tcp -m tcp
 --tcp-flags FIN,RST FIN,RST -j DROP
$ iptables -A INPUT -i eth0 -p tcp -m tcp
 --tcp-flags FIN,ACK FIN -j DROP
$ iptables -A INPUT -i eth0 -p tcp -m tcp
 --tcp-flags ACK,URG URG -j DROP
```

Source: typepad.com

Enter nftables

Source: nftables.org

- In 2009, the nftables project was created by Patrick McHardy to address the perceived problems of netfilter code duplication for each protocol and that of the Xtables mechanism slowing down packet handling
- In the mean time, the **`ipset`** command was introduced to simplify the creation of efficient look-up tables for “sets” of addresses
  - Made it more efficient to match rules against large numbers of IP addresses like from a blacklist of evil IPs
- nftables languished because it addressed a problem that had apparently already been solved
- Then, in 2013, the nftables project was revived by Pablo Neira Ayuso and the code made it into mainline for the 3.13 kernel
  - Driven by the explosion of the use of containers and VMs which made the existing Xtables solutions untenable because large sets of rules would come and go rapidly

nftables Architecture

- In order to simplify all of the Xtables commands into a generic syntax with a common API and significantly reduce the amount of duplicated code, nftables borrows the interpreter VM concept from BPF
  - The VM in the kernel space runs bytecode compiled in userspace via the **`nft`** CLI
- No separate APIs for iptables, ip6tables, arptables, ebtables, etc.
  - However, no predefined tables like iptables' INPUT, OUTPUT, FORWARD, etc.
  - So, you have to essentially start from scratch each time the system starts
- Uses the existing NF hooks and **`conntrack`** mechanism
  - Remains compatible with iptables via some translators
- Arithmetic, bitwise and comparison operators can be used for deciding the fate of packets
  - Supports arbitrary offsets into packets for evaluations
- Like Xtables, nftables are a sequence of Table->Chain->Rule tuples

Verify that nftables is in the Kernel

- Because nftables is an alternative to Xtables, your kernel may not have nftables enabled

  # lsmod | grep ^nf

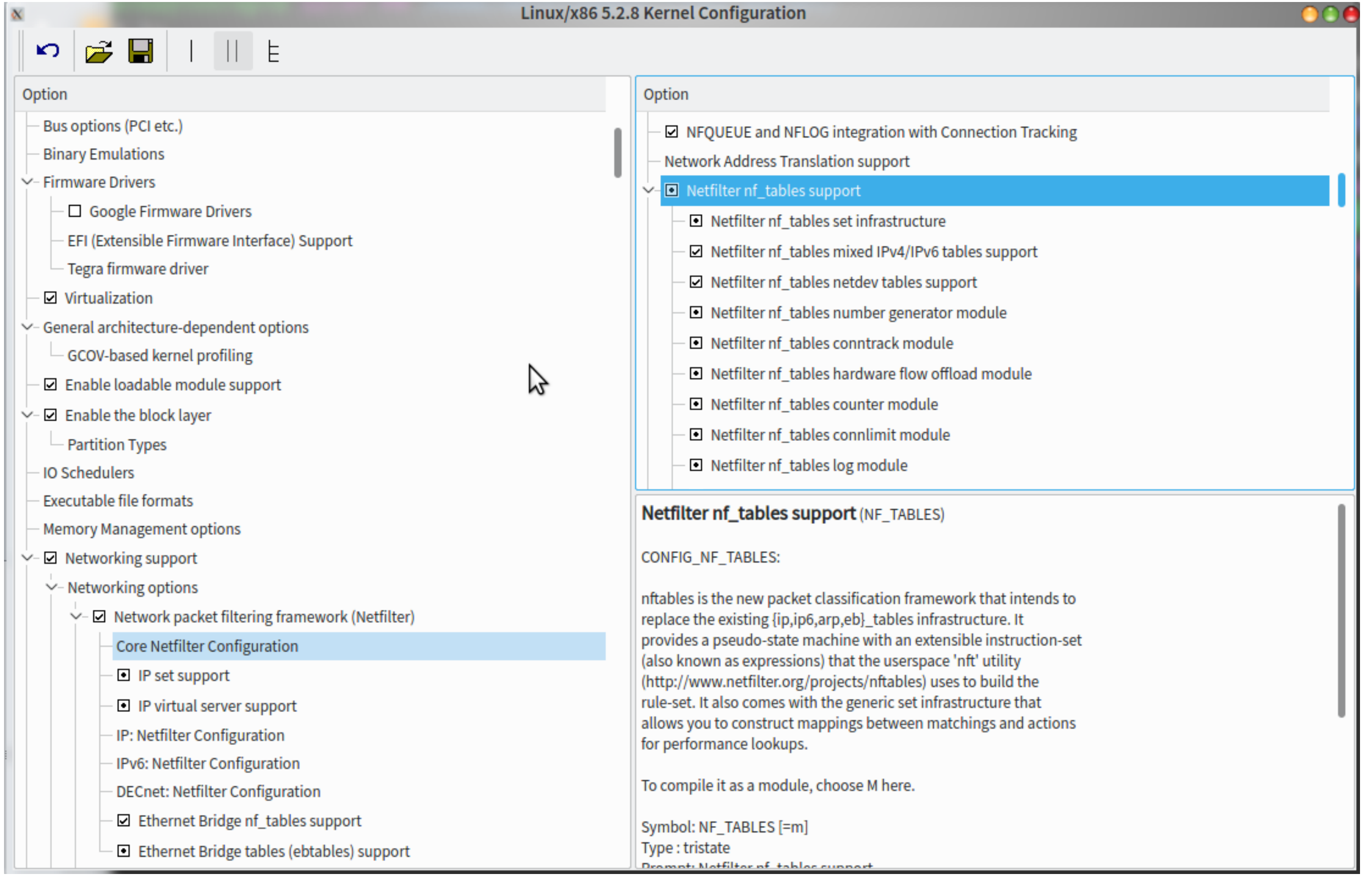
  <lots of modules beginning with nf>

- It's also possible that your kernel may have nft statically linked
  - Try to create a table and chain using:

```
# nft add table inet filter
# nft add chain inet filter input
```

  - If these succeed, then nftables is installed

Kernel Configuration



Usage

- You will need to install the nftables and iptables-nftables-compat packages
  - You need the **`nft`** CLI and the iptables-to-nftables translators and their libraries
- The **`nft`** CLI is both a compiler and decompiler of the bytecode to/from the kernel VM
- Any changes made from the command line are transient and will be lost on the next reboot
  - Can be saved to a file and reloaded on the next reboot
    - i.e., if loaded from a file, these are considered “permanent”
    - The file **`/etc/nftables.conf`** is automatically loaded by **`systemd`**
- There are commands that will take iptables/ip6tables commands and show the equivalent nftables command
- Also, there are iptables syntax-compatible commands that use the nftables framework
  - E.g., **`iptables-compat`**, **`arptables-compat`**, **`ebtables-compat`**, etc.

Coexistance of Xtables and nftables

- You can have both Xtables and nftables active simultaneously
  - Not recommended as this makes it nearly impossible to debug
- To disable iptables/ip6tables, use:

```
# iptables -F; iptables -L
# ip6tables -F; ip6tables -L
```

- To disable nftables, use:

```
# nft flush ruleset
```

Source: sfgate.com

Basic Approach

- The sequence of tasks in nftables is to create a table(s), then chain(s), then rule(s)
- Each command should include an address family
  - Defaults to the **`ip`** family if none specified
- The address families are:
  - **`ip`** IPv4 address family
  - **`ip6`** IPv6 address family
  - **`inet`** Internet (IPv4/IPv6) address family
  - **`arp`** ARP address family (IPv4 ARP packets)
  - **`bridge`** Bridge address family (L2)
  - **`netdev`** Netdev address family, handling packets from ingress

CLI vs. File

- It is possible to enter all of the elements of the tables via the **`nft`** CLI
  - However, some of the options can be tricky to enter from the command line due to the shell's line interpreter

```
# nft add chain ip traffic-filter output \
{ type filter hook output priority 0 \; policy accept\; }
```

- Alternatively, you can put the commands into a formatted file and import the file using:

```
# nft -f <filename>
```

Sample IPv4/IPv6 Combined Firewall

```
#!/sbin/nft -f

flush ruleset

table inet filter {
   chain input {
       type filter hook input priority 0; policy drop;
       ct state invalid counter drop comment "early drop of invalid packets"
       ct state {established, related} counter accept comment "accept all connections related to connections made by us"
       iif lo accept comment "accept loopback"
       iif != lo ip daddr 127.0.0.1/8 counter drop comment "drop connections to loopback not coming from loopback"
       iif != lo ip6 daddr ::1/128 counter drop comment "drop connections to loopback not coming from loopback"
       ip protocol icmp counter accept comment "accept all ICMP types"
       ip6 nexthdr icmpv6 counter accept comment "accept all ICMP types"
       tcp dport 22 counter accept comment "accept SSH"
       counter comment "count dropped packets"
   }

   chain forward {
       type filter hook forward priority 0; policy drop;
       counter comment "count dropped packets"
   }

   # If you're not counting packets, this chain can be omitted.
   chain output {
       type filter hook output priority 0; policy accept;
       counter comment "count accepted packets"
   }
}
```

Source: cbronline.com

Example Bridge w/ Failed Translation

```
# ebtables-nft -L
Bridge table: filter Bridge chain: INPUT, entries: 0, policy: ACCEPT
Bridge chain: FORWARD, entries: 2, policy: ACCEPT
--802_3-type 0x0001 -j CONTINUE
--mark 0x1 -j CONTINUE
Bridge chain: OUTPUT, entries: 0, policy: ACCEPT

# nft list ruleset
table bridge filter {
    chain INPUT {
        type filter hook input priority -200; policy accept;
    }
    chain FORWARD {
        type filter hook forward priority -200; policy accept;
        #--802_3-type 0x0001 counter packets 0 bytes 0
        #--mark 0x1 counter packets 0 bytes 0
    }
    chain OUTPUT {
        type filter hook output priority -200; policy accept;
    }
}
```

Translating from iptables to nftables

- It is possible to migrate your existing iptables firewalls to nftables using the following sequence:

```
# iptables-save > iptables.b4nft
# iptables-restore-translate -f iptables.b4nft > ruleset.nft
# nft -f ruleset.nft
```

Source: migrationboutique.com

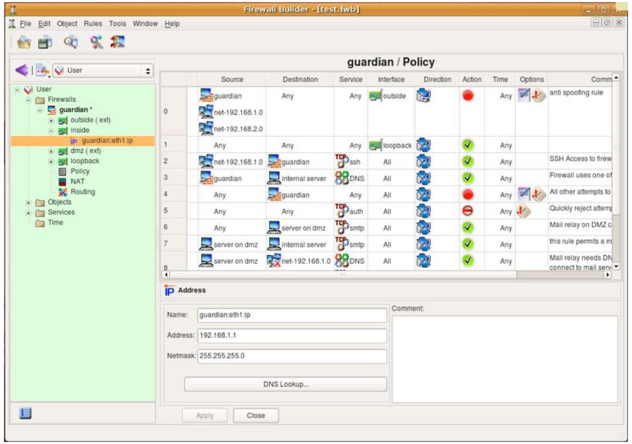
- Then, you can move these commands into the `/etc/nftables.conf` so **systemd** will load them at boot
- You can also translate iptables commands one-by-one:

```
# iptables-translate -A INPUT -i eth0 -p tcp -m tcp \
--tcp-flags FIN,SYN,RST,PSH,ACK,URG NONE -j DROP

nft add rule ip filter INPUT iifname eth0 tcp flags &\
(fin|syn|rst|psh|ack|urg) == 0x0 counter drop
```

GUI Firewall Builder for nftables?

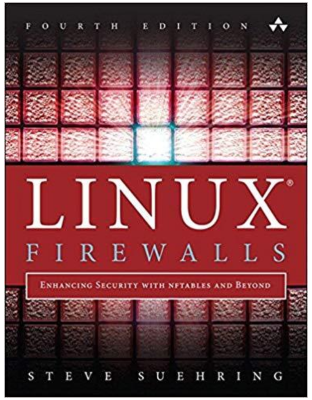
- The syntax for nftables is relatively complex and not well documented
  - So, is there a GUI that outputs the rules in the correct format?
- Well, sort of…
  - fwbuilder is used by Red Hat and claims to have output compatible with nftables
    - But, there's nothing obvious in the GUI for nftables
  - And, any of the iptables-based firewall builders could output iptables commands that could be converted to nftables using the techniques presented earlier



Source: sourceforge.net

Finding Documentation

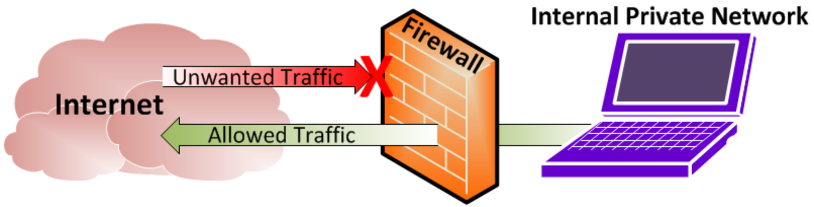
- Unfortunately, there's not a lot of documentation on nftables
- The nftables HOWTO can be found here:
  https://wiki.nftables.org/wiki-nftables/index.php/Main_Page
- ArchLinux also has a wiki:
  https://wiki.archlinux.org/index.php/Nftables
- Fortunately, there's also a book:
  Linux Firewalls: Enhancing Security with nftables and Beyond, Steve Suehring, Addison-Wesley Professional, 2015, ISBN10:0134000021
- And, there is the **`nft`** manual page ☺



Source: amazon.com

Summary

- nftables is heralded as the future for Linux firewalls
  - In spite of being in mainline for over 5 years, there's still very little documentation
- nftables addresses many of the problems encountered with Xtables with respect to:
  - Different syntax with the various tools
    - One unified syntax across L2-L4
  - Code duplication between address families
    - Single kernel-based VM ala BPF
  - Stack slow downs due to large numbers of rules
    - Rules can be aggregated with fast lookup
  - Requiring reloading all of the rules if something needed to be modified
    - The ability to modify any rule in any chain without needing to reload
- Fortunately, there are several compatibility commands and layers that allow you to continue with the known Xtables syntax and have that converted to nftables
- The bottom line is that the learning curve for nftables is steep
  - But, the benefits appear to be worth the effort



Source: tunnelsup.com

Embedded Linux Conference

North America